Social, Legal, Ethical, Empathetic, and Cultural Rules: Compilation and Reasoning

Nicolas Troquard, Martina De Sanctis, Paola Inverardi,
Patrizio Pelliccione, Gian Luca Scoccia

Gran Sasso Science Institute
L'Aquila, Italy
firstname.surname@gssi.it

Abstract

The rise of AI-based and autonomous systems is raising concerns and apprehension due to potential negative repercussions stemming from their behavior or decisions. These systems must be designed to comply with the human contexts in which they will operate. To this extent, Townsend et al. (2022) introduce the concept of SLEEC (social, legal, ethical, empathetic, or cultural) rules that aim to facilitate the formulation, verification, and enforcement of the rules AI-based and autonomous systems should obey. They lay out a methodology to elicit them and to let philosophers, lawyers, domain experts, and others to formulate them in natural language. To enable their effective use in AI systems, it is necessary to translate these rules systematically into a formal language that supports automated reasoning. In this study, we first conduct a linguistic analysis of the SLEEC rules pattern, which justifies the translation of SLEEC rules into classical logic. Then we investigate the computational complexity of reasoning about SLEEC rules and show how logical programming frameworks can be employed to implement SLEEC rules in practical scenarios. The result is a readily applicable strategy for implementing AI systems that conform to norms expressed as SLEEC rules.

1 Introduction

The rise of AI-based and autonomous systems is posing concerns and causing apprehension due to potential negative repercussions arising from their behavior or decisions. Philosophers, engineers, and legal experts are actively exploring avenues to regulate the actions of these systems, which are becoming increasingly integrated into our daily existence. One way to exert control over these systems involves the identification of rules dictating their conduct during collaboration and interaction with humans.

Normative multi-agent systems (Chopra et al. 2018) are systems where the behaviour of the participants is affected by the norms established therein. Townsend et al. (2022) propose a methodology to elicit normative rules for multi-agent systems. The process follows an iterative approach which is:

1. identify high-level normative principles,
2. identify robot/system capabilities,

3. create a preliminary normative rule,
4. identify conflicts between normative rule and robot/system capabilities, and if any:
   - resolve normative conflicts,
   - refine normative rule,
   - go back to 4.

In the process, philosophers, ethicists, lawyers, domain experts, and other professionals can extract normative rules from a specific domain. These rules are referred to as *SLEEC rules*, which stands for *social, legal, ethical, empathetic, and cultural* rules. These rules are aimed to play a crucial role for facilitating the creation of tools and instruments to engineer systems that conform adequately to the established norms, while also aiding in the verification of existing system behavior. However, the SLEEC rules obtained through elicitation are typically expressed in natural language. Therefore, to be employed effectively in AI systems, these rules need to be translated into a machine-interpretable language. Fortunately, the elicitation process generates rules that are characterized by a clear pattern.

The primary objective of this paper is to present a systematic approach for translating SLEEC rules into classical logic, thereby facilitating their seamless integration in SLEEC-sensitive AI systems.

Example 1 *Townsend et al. (2022) describe the rule elicitation methodology on a running example in the domain of nursing homes. At the end of the iterative process, the following SLEEC rule is obtained.*

> *When the user tells the robot to open the curtains then the robot should open the curtains, UNLESS the user is 'undressed' in which case the robot does not open the curtains and tells the user 'the curtains cannot be opened while you, the user, are undressed,' UNLESS the user is 'highly distressed' in which case the robot opens the curtains.*

For clarity of exposition and concision, we will follow suit and illustrate the present work with this running example.

Compilation and Reasoning. This paper delves into an exploration of the suitability of classical logic and logic programming frameworks for representing and reasoning about Townsend et al.'s rules, emphasizing the simplicity and practicality of these approaches within this context.

"*Open the curtains!*" (sensing a)

User is dressed (sensing d)

$\texttt{rules} \wedge a \wedge d \models o$

I must open the curtains (deciding o)
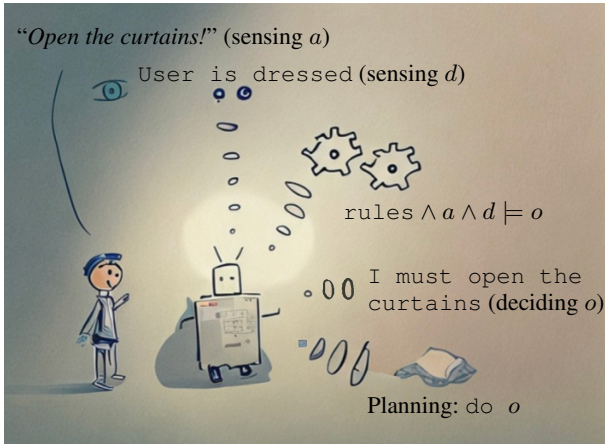
Planning: do o

Figure 1: SLEEC-compliant AI systems.

In computer science, a *compilation* is a translation of computer code in a language into computer code in another language. By *normative rule compilation* we intend a *translation of SLEEC rules* from natural language *into a logical language*.

Example 1 (continue) *After disambiguation and identification of the relevant pieces of information, the logical form of the SLEEC rule is ($\neg\neg$ is deliberate):*

$$(a \wedge \neg\neg d \to o) \wedge (a \wedge \neg d \wedge \neg h \to n \wedge s) \wedge (a \wedge \neg d \wedge h \to o) \ .$$

We will provide the details of the successive steps in due time.[1]

Compiling SLEEC rules into a formal language such as classical logic offers numerous advantages. Firstly, it endows SLEEC rules with a precise semantics, enabling unequivocal determinations of their consistency and whether specific outcomes are necessary consequences of the set of rules. Secondly, it facilitates machine processing of SLEEC rules, allowing the integration of off-the-shelf theorem provers, reasoners, and logic programming frameworks (e.g., PRO-LOG, Answer Set Programming)[2] for embedding SLEEC-compliant decision-making modules into AI systems.

Example 1 (continue) *Figure 1 illustrate the central role of reasoning with SLEEC rules. The robot senses the user asking 'open the curtains!', setting the propositional variable a to true. The robot senses that the user is dressed, setting the propositional variable d to true. The propositional variable o stands for the obligation to open the curtains. Automated reasoning is then performed to decide whether the formalised rule (see before), a, and d together entail o. Since it is case, the robot concludes that it ought to open the curtains. It can now perform some planning task to ensure, e.g., that sometime in the future the curtains are opened.*

---

[1] For impatient readers, reference Section 4.2 to understand the reading of the propositional variables, and Example 5.

[2] Robot programming platforms already propose wrapped logic programming frameworks for easier integration, e.g., KnowRob/RosProlog (Tenorth and Beetz 2017) and ROSoClingo (Andres et al. 2015).

The literature in multi-agent systems, knowledge representation, and normative reasoning offers numerous ad-hoc formalisms to represent and reason about norms (Meyer and Wieringa 1994; Gabbay et al. 2013, 2021). These specialized formalisms for normative reasoning hold a great value for representing complex norms *that have not been elicited into SLEEC rules*. However, we will demonstrate that the specific rules proposed by Townsend et al. (2022) can be effectively represented using a simple classical logic approach. Furthermore, implementing these rules in logic programming frameworks such as PROLOG or Answer Set Programming (ASP) is a straightforward process. Thus, simple formalisms, supported by a wide range of readily-available software support, are sufficient for developing SLEEC-sensitive AI systems. Yaman et al. (2023) have presented SLEECVAL, a tool designed for the validation of SLEEC rules through the mapping of rules to tock-CSP processes. Subsequently, a CSP refinement checker can be employed to detect redundant and conflicting rules. A tool to identify conflicts, redundancies, and concerns, aimed at assisting experts during SLEEC rule elicitation, is also presented by Feng et al. (2023). In contrast, our logic-based compilation is straightforward and allows for versatile reasoning instead of mere verification. We can effortlessly verify the consistency of a set of SLEEC rules and integrate it into a robot development platform to perform decision-making tasks, determining the obligations that arise from a given set of rules and observations as in Figure 1.

Outline. The reader unfamiliar with propositional logic or logic programming can find some basic definitions in Section 2. In Section 3, we conduct a linguistic analysis of the SLEEC rules pattern, which justifies the translation of SLEEC rules into classical logic. In Section 4 we briefly explain how to take care of ambiguities and to identify the relevant pieces of information in SLEEC rules. In Section 5, we define the logical translation of SLEEC rules, thus also establishing a formal semantics. In Section 6, we study the computational complexity of reasoning about SLEEC rules. In Section 7, we demonstrate how logic programming frameworks can be employed to implement SLEEC rules in practical scenarios. We conclude in Section 8.

2 Elements of Logic and Logic Programming

To maintain self-contained clarity, we lay out the basic definitions of classical propositional logic, along with brief elements of logic programming. This foundational knowledge will suffice for understanding the forthcoming arguments.

2.1 Propositional Logic

Let AP be a non-empty enumerable set of propositional variables. A *propositional formula* ϕ over AP is defined by the following grammar: $\phi ::= p \mid \neg\phi \mid \phi \vee \phi$, where $p \in AP$. As it is usual, when ϕ and ψ are propositional formulas, we define $\phi \wedge \psi = \neg(\neg\phi \vee \neg\psi)$, and $\phi \to \psi = \neg\phi \vee \psi$, and $\top = q \vee \neg q$ for some $q \in AP$.

A *positive literal* is a propositional variable p and a *negative literal* is the negation of a propositional variable $\neg p$. A *clause* is a disjunction of literals. A *formula in CNF* is a

conjunction of clauses. A *formula in 3CNF* is a conjunction of clauses of three literals. A *Horn formula* is a conjunction of clauses containing at most one negative literal.

An *interpretation* over AP is a function $v : AP \longrightarrow \{true, false\}$. It can be extended to complex formulas as follows:

- $v(\neg\phi) = true$ iff $v(\phi) = false$.
- $v(\phi \lor \psi) = true$ iff $v(\phi) = true$ or $v(\psi) = true$.

A *model* of a formula ϕ over AP is an interpretation v over AP such that $v(\phi) = true$. A propositional formula ϕ is *satisfiable* if there is an interpretation v such that $v(\phi) = true$. Let Γ be a set of propositional formulas, and ϕ be a propositional formula. We say that Γ *entails* ϕ if for all interpretations v, if $v(\psi) = true$ for all $\psi \in \Gamma$, then $v(\phi) = true$. We write $\Gamma \models \phi$ when it is the case. We write $\phi \equiv \psi$ when the two formulas ϕ and ψ are logically equivalent, that is, when $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$.

2.2 Logic Programming

Logic programming is a programming paradigm using logical formulas to describe a domain.

A program is a list of clauses of the form:

```
H :- P1, ..., Pn.
```

where `H` is the *head*, and each `Pi` is part of the *body*. It can be read as "if `P1` and ... and `Pn` then `H`". The head and the parts of the body can be an arbitrary predicate with arbitrary variables. E.g.,

```
motherof(X,Y) :- woman(X), childof(Y,X).
```

captures the fact that if `X` is a woman, and `Y` is the child of `X`, then `X` is the mother of `Y`. A fact is a clause without a body. E.g.,

```
woman(jane).
childof(teddy,jane).
```

expresses that `jane` is a woman and `teddy` is the child of `jane`. All predicate names must appear in a rule head (or in a fact).

The comma `,` represents a conjunction, while the semicolon `;` represents a disjunction. The negation (as failure) is written `\+` in PROLOG and `not` in ASP. Finally, a line starting with `%` is a comment.

In PROLOG, we can 'query' the knowledge base. E.g.,

```
motherof(M,C).
```

will enumerate all couples (`M`, `C`) such that `M` is the mother of `C`. In our example, it will find exactly one solution: `M = jane, C = teddy`.

3 Unless... In Which Case...

We assume that we are given a SLEEC rule, elicited following the methodology of Townsend et al. (2022). The pivotal logical connective in SLEEC rules is the word 'unless'. And importantly, 'unless' is typically followed by 'in which case'. The aim of this section is to define the meaning of the formulas of the form ϕ UNLESS ψ IN WHICH CASE χ where ϕ, ψ, and χ are arbitrary propositional formulas, and to justify the proposed definition. This justification is crucial for instilling trust in the compilation process, which is

of utmost importance for applications in social contexts. We first look at 'unless' in isolation, and then provide a logical definition of the construction 'unless... in which case...'.

3.1 'Unless' as XOR

To persuade both non-logicians and logicians alike of the correct logical interpretation of 'unless', it is helpful to take a moment and consider what it is not. Indeed, it is extremely tempting to interpret ϕ UNLESS ψ as an 'exclusive or' (Barker-Plummer, Barwise, and Etchemendy 2011). That is, $(\phi \lor \psi) \land \neg(\phi \land \psi)$, and is equivalent to the biconditional $\neg\psi \leftrightarrow \phi$. The biconditional $\leftrightarrow$ can also be read 'if and only if', and written IFF in English sentences.

For simplicity of exposition, we will step away from SLEEC rules statements for a moment. Consider the statement $1a$, and its XOR reading ($1b$).

$1a$ Ana is in Florence UNLESS she is in Rome.

$1b$ Ana is in Florence IFF she is not in Rome.

  $1b1$ If Ana is not in Rome, then Ana is in Florence. (✓)

  $1b2$ If Ana is in Rome, then Ana is not in Florence. (✓)

The statement $1b$ can be decomposed into the statements $1b1$ and $1b2$. Both are sensible consequences of $1b$. But only because of some background knowledge that a person cannot be in two places at the same time. To make the issue clearer, consider the statement $2a$, and its XOR reading ($2b$).

$2a$ Ana attends the meeting UNLESS she is in Rome.

$2b$ Ana attends the meeting IFF she is not in Rome.

  $2b1$ If Ana is not in Rome, then Ana attends the meeting. (✓)

  $2b2$ If Ana is in Rome, then Ana does not attend the meeting. (❓)

The statement $2b$ can be decomposed into the statements $2b1$ and $2b2$. The statement $2b1$ is a sensible consequence of $2b$. But what about the statement $2b2$? It is in fact *only* a *Gricean conversational implicature* (Grice 1975), and not part of the meaning of the original statement. To see this, one can apply Grice's *cancellation principle*. It postulates that if a conclusion is not part of the meaning of an assertion, it can be 'cancelled' without contradiction by some further elaboration. E.g.,

$3a$ Ana attends the meeting UNLESS she is in Rome, but *if she is in Rome she may still attend remotely*.

  $3b1$ If Ana is not in Rome, then Ana attends the meeting. (✓)

  $3b2$ If Ana is in Rome, then Ana does not attend the meeting. (✗)

It is clear that the statement $3b1$ is a consequence of the statement $3a$. But the statement $3b2$ is not, because the further elaboration 'but if she is in Rome she may still attend remotely' makes it clear that even if Ana is in Rome, it is possible that she attends the meeting.

Back to Example 1, 'if the user is undressed, then the robot does not open the curtains' is *only a conversational implicature* and is not part of the meaning of 'the robot opens the curtains UNLESS the user is undressed'. Indeed,

we could add 'and if the user is undressed, then it depends on whether the user is highly distressed'. And it is precisely what the full statement of the rule takes care of by adding "UNLESS the user is 'highly distressed' in which case the robot opens the curtains". It takes away the suggestion that 'if the user is undressed, then the robot does not open the curtains'.

3.2 The Logical Form of 'Unless'

The linguistic argument above serves to refute the right-to-left implication of the XOR/IFF formulation of 'unless'. In fact, Quine (1959) contends that ϕ UNLESS ψ must be logically interpreted as $\neg\psi \to \phi$. It is exactly the left-to-right implication of the XOR/IFF formulation above. We adopt it as a definition.

Definition 1 ϕ UNLESS $\psi = \neg\psi \to \phi$.

Translating 'unless' statements into classical logic becomes a very simple rewriting process.

Example 2 *The sentence "the robot should open the curtains, UNLESS the user is 'undressed'" is logically equivalent to the sentence "IF the user is not 'undressed' THEN the robot should open the curtains".*

With a working logical definition of 'unless', we effectively establish a semantic condition for it.

Fact 1 *Let v be an interpretation. We have*

$v(\phi \text{ UNLESS } \psi) = true \text{ iff } v(\neg\phi) = false \text{ or } v(\psi) = true.$

It comes with certain simple properties, some of which might appear surprising.

Fact 2 *The following properties hold:*

1. *equivalence to logical OR:* ϕ UNLESS $\psi \equiv \phi \vee \psi$.
2. *associativity:* $(\phi$ UNLESS $\psi)$ UNLESS $\chi \equiv \phi$ UNLESS$(\psi$ UNLESS $\chi)$.
3. *commutativity:* ϕ UNLESS $\psi \equiv \psi$ UNLESS ϕ.

Despite their surprising nature, these properties of 'unless' will not pose a problem, especially in the broader context of SLEEC rules.

3.3 Unless... in Which Case...

The SLEEC rules do not use the word 'unless' in isolation, but instead in a construction that could be the ternary Boolean expression:

$$\phi \text{ UNLESS } \psi \text{ IN WHICH CASE } \chi \ .$$

It is a stronger statement than just "ϕ UNLESS ψ", specifying not only what should be the case when ψ is not true (i.e., ϕ), but also what should be the case when ψ is true (i.e., χ). It can thus be interpreted as the conjunction of ϕ UNLESS ψ, and IF ψ THEN χ. Hence, we obtain the following definition:

Definition 2 ϕ UNLESS ψ IN WHICH CASE χ = $(\phi$ UNLESS $\psi) \wedge (\psi \to \chi)$.

Using Definition 1, it is of course equivalent to $(\neg\psi \to \phi) \wedge (\psi \to \chi)$. Clearly, we have ϕ UNLESS $\psi \equiv \phi$ UNLESS ψ IN WHICH CASE $\top$, where $\top$ is the tautology.

This logical definition allows us to establish a semantic condition for it.

Fact 3 *Let v be an interpretation. We have*

$$v(\phi \text{ UNLESS } \psi \text{ IN WHICH CASE } \chi) = true \text{ iff}$$
$$v(\neg\psi) = false \text{ or } v(\phi) = true \text{ and}$$
$$v(\neg\psi) = true \text{ or } v(\chi) = true \ .$$

The following example will aid in clarifying the intuitions.

Example 3 *The sentence "the robot should open the curtains, UNLESS the user is 'undressed' in which case the robot does not open the curtains and tells the user 'the curtains cannot be opened while you, the user, are undressed,'" is logically equivalent to the sentence "IF the user is not 'undressed' THEN the robot should open the curtains, AND, IF the user is 'undressed' THEN the robot does not open the curtains AND tells the user [...]".*

After what we said about the commutativity of UNLESS (Fact 2.3) it would be a *mistake* to think that ϕ UNLESS ψ IN WHICH CASE χ is equivalent to ψ UNLESS ϕ IN WHICH CASE χ. Hence, the order of occurrence of the different conditions and exceptions in the SLEEC rule *do* have an importance.

4 Identifying the Pieces of Information in SLEEC Rules

In order to formalise a SLEEC rule, one needs to attach a logical term or variable to every *piece of information*. A piece of information in a SLEEC rule is an atomic statement that occurs in the rule. For instance, in Example 1, 'the user tells the robot to open the curtains' and 'the robot does not open the curtains' are two pieces of information. We identify all the pieces of information of Example 1 in Section 4.2

On close inspection, we can see that the statements in SLEEC rules can be partitioned into two kinds: statements whose truth value is established by sensing the world (e.g., the user tells the robot to open the curtains, or the user is undressed), and statements whose truth value is established by the existence of an obligation for the robot to act (e.g., the robot should open the curtains, or the robot tells the user something).

Some further considerations should be taken into account. A logical term must be attached to only one piece of information. Due to the ambiguity of natural languages, this may require a preliminary analysis to disambiguate some statements in the rules. We illustrate it in Section 4.1.

4.1 Disambiguation of Other Words: Tell and Tell

The rule of the running example contains two occurrences of the word 'tells'.

> [...] the user *tells* the robot to open the curtains [...] the robot [...] *tells* the user 'the curtains cannot be opened [...]'.

They must be interpreted in two different ways.

> [...] the user *asks* the robot to open the curtains [...] the robot [...] *says [to]* the user 'the curtains cannot be opened [...]'.

It is up to the modeller to avoid this sort of ambiguity. When they remain, it is up to the engineer in charge of formalising the rules to detect them.

4.2 Determining the Pieces of Information in the Domain

Once the statements in the SLEEC rules are disambiguated, we are ready to establish a one-to-one mapping between the set of atomic statements in the rules and a set of logical variables.

In Example 1, we can informally partition the pieces of information into two sets: the variables whose value is known from *sensing* (a, d, h),

a user asks the robot to open the curtains,

d user is dressed,

h user is highly distressed,

and the variables whose value is determined by the existence of an *obligation to act* (o, n, t),

o obligation to open the curtains,

n obligation not to open the curtains,

s obligation to says to the user 'the curtains cannot be opened [...]'.

This partitioning is helpful to make sense of the rules, but it is inconsequential for our solution.

5 Compilation of SLEEC Rules Into Logic

We present a general pattern that SLEEC rules follow, and using the observations of Section 3, we explain how they must be translated into classical logic.

5.1 General Pattern for SLEEC Rules

Definition 3 *A (general pattern of a) SLEEC rule is an expression of the form*

> *IF C_0 THEN O_0,*
> *UNLESS C_1 IN WHICH CASE O_1,*
> *UNLESS C_2 IN WHICH CASE O_2,*
> *...*
> *UNLESS C_n IN WHICH CASE O_n.*

where C_i and O_i, $0 \le i \le n$, are arbitrary propositional formulas over a set of propositional variables AP.

Typically, in Definition 3 C_i is a Boolean condition about sensed variables, and O_i about obligations to act.

Example 4 *After substituting disambiguated words, we obtain the following intermediate forms for Example 1.*

> *IF the user* asks *the robot to open the curtains [a] THEN the robot should open the curtains [o], UNLESS the user is 'undressed' [$\neg d$] IN WHICH CASE the robot does not open the curtains [n] AND* says to *the user 'the curtains cannot be opened while you, the user, are undressed,' [s] UNLESS the user is 'highly distressed' [h] IN WHICH CASE the robot opens the curtains [o].*
>
> $\Downarrow$
>
> *IF a THEN o,*
> *UNLESS $\neg d$ IN WHICH CASE $n \wedge s$,*
> *UNLESS h IN WHICH CASE o.*

The latter follows the pattern identified in Definition 3.

5.2 Translation Into Classical Logic

Prior to presenting the propositional logic form of SLEEC rules, we provide some insights into its derivation. The general pattern of SLEEC rules can be expressed in a parenthesized manner as follows:[3]

> IF C_0 THEN (O_0,
> UNLESS C_1 IN WHICH CASE (O_1,
> UNLESS C_2 IN WHICH CASE (O_2,
> ...
> UNLESS C_n IN WHICH CASE (O_n) ...))).

Applying Definition 2 n times, and *uncurrying* (that is, applying the left-to-right implication of the following logical equivalence $(p \rightarrow (q \rightarrow r)) \equiv (p \wedge q \rightarrow r)$), we get the following simplified representation:

> IF C_0 AND NOT C_1 THEN O_0,
> IF C_0 AND C_1 AND NOT C_2 THEN O_1,
> IF C_0 AND C_1 AND C_2 AND NOT C_3 THEN O_2,
> ...
> IF C_0 AND C_1 AND C_2 AND ... AND C_{n-1} AND NOT C_n THEN O_{n-1},
> IF C_0 AND C_1 AND C_2 AND ... AND C_{n-1} AND C_n THEN O_n.

Finally, we obtain the following definition of the compilation of a SLEEC rule into classical propositional logic.

Definition 4 *Let AP be a set of propositional variables. Let C_i and O_i, $0 \le i \le n$, be arbitrary propositional formulas over AP. Let*

$$\sigma = \textit{IF } C_0 \textit{ THEN } O_0,$$
$$\textit{UNLESS } C_1 \textit{ IN WHICH CASE } O_1,$$
$$\textit{UNLESS } C_2 \textit{ IN WHICH CASE } O_2,$$
$$...$$
$$\textit{UNLESS } C_n \textit{ IN WHICH CASE } O_n.$$

be a SLEEC rule. We define the compilation *of σ, noted* compile(σ), *as follows:*

$$\left[\bigwedge_{0 \le i \le n-1} \left(\left(\bigwedge_{0 \le j \le i} (C_j) \wedge \neg C_{i+1} \right) \rightarrow O_i \right) \right]$$

$$\wedge \left[\left(\bigwedge_{0 \le j \le n} (C_j) \right) \rightarrow O_n \right] \ .$$

We are finally ready to give the last step in the compilation of the SLEEC rule of Example 1.

Example 5 *The SLEEC rule of Example 1 is compiled as follows.*

$$\textit{IF } a \textit{ THEN } o,$$
$$\textit{UNLESS } \neg d \textit{ IN WHICH CASE } n \wedge s,$$
$$\textit{UNLESS } h \textit{ IN WHICH CASE } o \ .$$

---

[3]A possible alternative is IF C_0 THEN $(\dots((O_0,$ UNLESS C_1 IN WHICH CASE $O_1),$ UNLESS C_2 IN WHICH CASE $O_2),$ $\dots$ UNLESS C_n IN WHICH CASE $O_n)$, for which a similar analysis can be done.

$\Downarrow$ *compile*

$(a \wedge \neg\neg d \to o) \wedge (a \wedge \neg d \wedge \neg h \to n \wedge s) \wedge (a \wedge \neg d \wedge h \to o)$.

The semantics of SLEEC rules follows.

Fact 4 *Let σ be a SLEEC rule as in Definition 3, and let v be an interpretation. We have: $v(\sigma) = true$ iff*

$$\big[\forall 0 \le i \le n-1, (\exists 0 \le j \le i, v(C_i) = false$$
$$or\ v(C_{i+1}) = true\ or\ v(O_i) = true)\big]\ and$$
$$\big[\exists 0 \le i \le n, v(C_i) = false\ or\ v(O_n) = true\big]\ .$$

The size of the compiled form of a SLEEC rule is quadratic in the size of the SLEEC rule.

Fact 5 $|compile(\sigma)| = O(|\sigma|^2)$.

This is instrumental to establish the complexity results of Section 6.

6 Complexity of Reasoning About SLEEC Rules

Since we are dealing with a special form of propositional formulas, we provide a few observations on the theoretical complexity of reasoning about SLEEC rules.

Proposition 1 *Let AP be a set of propositional variables. Deciding whether an obligation ϕ (more generally, any propositional statement over AP) is entailed by a set Γ of SLEEC rules is CONP-complete, even if the terms $C_0, \ldots C_n$ and $O_0, \ldots O_n$ in every rule are restricted to be literals in $\{p, \neg p \mid p \in AP\}$.*

Proof *For membership, consider the complement problem, asking whether the set of rules Γ does not entail the statement ϕ. This can be solved by the following algorithm: compile the SLEEC rules following Definition 4 to obtain a conjunction of propositional formulas with only a quadratic blowup in size (Fact 5); non-deterministically choose a valuation; check whether it satisfies all rules in Γ and does not satisfy ϕ. It is correct and sound, and runs in polynomial time on a non-deterministic Turing machine. This means that our problem is in CONP.*

For hardness, we can polynomially reduce 3SAT. For $p \in AP$, define $\sim p = \neg p$ and $\sim \neg p = p$. An instance of 3SAT is a formula of the form:

$$(l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \ldots \wedge (l_{k,1} \vee l_{k,2} \vee l_{k,3})\ ,$$

with every $l_{i,j} \in \{p, \neg p \mid p \in AP\}$. Each clause $(l_{i,1} \vee l_{i,2} \vee l_{i,3})$ can be captured by the SLEEC rule corresponding to

   IF $\sim l_{i,1}$ THEN $l_{i,3}$,
   UNLESS $l_{i,2}$ IN WHICH CASE $\top$

Indeed, its logical form is $(\sim l_{i,1} \wedge \sim l_{i,2} \to l_{i,3}) \wedge (\sim l_{i,1} \wedge l_{i,2} \to \top)$, which is equivalent to $l_{i,1} \vee l_{i,2} \vee l_{i,3}$. It means that deciding an entailment problem from an arbitrary 3CNF over AP with k clauses can be reduced to deciding an entailment problem from k SLEEC rules where every term is a literal. Entailment for 3CNF (3SAT) is CONP-hard, so hardness follows. $\square$

A consequence of Proposition 1 is that reasoning with the rules of (Townsend et al. 2022) is (worst-case) hard unless one imposes drastic restrictions. An example of restriction that makes the entailment of an obligation from a set of SLEEC rules easier is when all C_i are negative literals and all O_i are positive literals.

Proposition 2 *Let AP be a set of propositional variables. Deciding whether an obligation o is entailed by a set Γ of SLEEC rules is in PTIME when the terms $C_0, \ldots C_n$ of all rules are all negative literals in $\{\neg p \mid p \in AP\}$ and all $O_0, \ldots O_n$ of all rules are positive literals in $\{p \mid p \in AP\}$.*

It follows from a simple reduction to HORNSAT which is in PTIME (Dowling and Gallier 1984) (see appendix of (Troquard et al. 2023) for details).

Another consequence of Proposition 1 is that there is no reason to restrict ourselves to $C_0, \ldots C_n$ and $O_0, \ldots O_n$ being only literals. We can use complex C_i with no impact on the computational complexity. Since C_i describe states of affairs, it is very convenient to be able to write them as arbitrary propositional formulas. We can do so without impacting the complexity of reasoning.

We can use complex O_i. For instance, if two obligations o_1 and o_2 should be triggered at the same time, it would not alter the complexity of reasoning to simply write $o_1 \wedge o_2$ for some O_i. In the rule of Example 1, we have: "in which case the robot does not open the curtains and tells the user 'the curtains cannot be opened while you, the user, are undressed'". We could use a single atom $o_{n,s}$ to represent the obligation to not open the curtains and the obligation to say something to the user, but it does not hurt the computational complexity of reasoning if we use two obligations n and s, and simply identify a O_i with $n \wedge s$. We will make good use of this.

7 Reasoning With SLEEC Rules

We build upon the example provided in Example 1 to demonstrate the application of the compiled logical form of a SLEEC rule for automated normative reasoning. We give an overview of the reasoning tasks in propositional logic, Answer Set Programming, and in PROLOG.

In ASP and PROLOG (Section 7.2 and Section 7.3), we use the following predicates:

- `a(X,Y)`: user X **a**sks to open Y.
- `s(X,Y)`: obligation to **s**ay Y to X.
- `o(X)`: obligation to **o**pen X.
- `n(X)`: obligation to **n**ot open X.
- `d(X)`: X is **d**ressed.
- `h(X)`: X is **h**ighly distressed.

We start with reasoning in propositional logic.

7.1 Propositional Reasoning

The SLEEC rule of Example 1 is compiled into the following propositional formula: $(a \wedge \neg\neg d \to o) \wedge (a \wedge \neg d \wedge \neg h \to n \wedge s) \wedge (a \wedge \neg d \wedge h \to o)$. Let us note it ϕ_{SLEEC}. Any off-the-shelf SAT solver can now be used for reasoning.

Suppose that user asks, user is dressed, user is highly distressed. Together with the SLEEC rule, it *entails* the obligation to open the curtains. This is captured by:

$$(a \wedge d \wedge h) \quad \wedge \quad \phi_{SLEEC} \models o \ .$$

Now suppose that user asks, user is not dressed, user is not highly distressed. Then, together with SLEEC rule, it *entails* the obligation to not open the curtains and say why:

$$(a \wedge \neg d \wedge \neg h) \quad \wedge \quad \phi_{SLEEC} \models n \wedge s \ .$$

To ensure the soundness of our solution, it is imperative that it does not propose any spurious obligations. Fortunately, our solution meets this requirement, as no additional obligations are entailed by the previous formulas. For instance, $(a \wedge d \wedge h) \wedge \phi_{SLEEC} \not\models n \vee s$, because the interpretation $v = \{a \mapsto true, d \mapsto true, h \mapsto true, \boldsymbol{n} \mapsto \boldsymbol{false}, o \mapsto true, \boldsymbol{s} \mapsto \boldsymbol{false}\}$ is a model of $(a \wedge d \wedge h) \wedge \phi_{SLEEC}$. Similarly, we have that $(a \wedge \neg d \wedge \neg h) \wedge \phi_{SLEEC} \not\models o$, because the interpretation $v' = \{a \mapsto true, d \mapsto false, h \mapsto false, n \mapsto true, \boldsymbol{o} \mapsto \boldsymbol{false}, s \mapsto true\}$ is a model of $(a \wedge \neg d \wedge \neg h) \wedge \phi_{SLEEC}$.

7.2 Logic Programming: Answer Set Programming

Example 1 can be formalised in ASP as follows.

```
o(Y) :- a(X,Y), d(X).
o(Y) :- a(X,Y), not d(X), h(X).
n(Y) :- a(X,Y), not d(X), not h(X).
s(X, Y) :- a(X,Y), not d(X), not h(X).
a(user,curtains).
d(someoneelse).
h(user).
```

We have chosen to represent a situation where the user is not dressed (`d(user)` is not present) but highly distressed (`h(user)`). The clause `d(someoneelse)` is present for technical reason, because all predicates must appear in the head of a rule. We obtain the expected model in the ASP system clingo (5.7.0):

```
Solving...
Answer: 1
d(someoneelse) a(user,curtains)
o(curtains) h(user)
SATISFIABLE
```

That is, if the user asks the robot to open the curtains, and the user is highly distressed, then the only model is when the robot has the obligation to open the curtains (`o(curtains)`).

7.3 Logic Programming: PROLOG

A straightforward implementation of Example 1 in PROLOG is as follows.

```
o(Y) :- a(X,Y), d(X),
        write("I have the obligation to open "),
        write(Y).
o(Y) :- a(X,Y), \+ d(X), h(X),
        write("I have the obligation to open "),
        write(Y).
n(Y) :- a(X,Y), \+ d(X), \+ h(X),
        write("I have the obligation not to open "),
```

```
        write(Y).
s(X, Y) :- a(X,Y), \+ d(X), \+ h(X),
        write("I can't open "), write(Y),
        write(" while you "), write(X),
        write(" are undressed").
a(user,curtains).  % user asks robot open the curtains
%d(someoneelse).
d(user).           % user is dressed
h(someoneelse).
%h(user).           % user is highly distressed
```

We simply enriched the rules with `write` predicates to provide more informal details in the query answers. Clauses with `someoneelse` are used to ensure that all predicates appear in the head of a rule.

To get all the obligations, we can ask the PROLOG query:

```
o(Z) ; n(Z) ; s(X, Y).
```

In SWI-Prolog (9.1.20), it yields

```
I have the obligation to open curtains
Z = curtains
```

Similar to the machine-processable input program, the output of the reasoning tasks is also machine processable. Notably, the conclusions drawn from the reasoning can be seamlessly transferred to other components of a robot platform, such as a planning module.

Alternatively, if the user is not dressed (that is if the clause `d(user)` is not present) then the same query is answered with:

```
I have the obligation not to open curtains
Z = curtains
I can't open curtains while you user are undressed
X = user,
Y = curtains
```

If the user is not dressed but highly distressed (that is if the clause `d(user).` is not present, but the clause `h(user).` is present), then the same query is answered with:

```
I have the obligation to open curtains
Z = curtains
```

8 Conclusions

We have bridged the crucial divide that separates the norms articulated as SLEEC rules by philosophers, ethicists, lawyers, domain experts, and other professionals from the realization of agents compliant with these norms on robot development platforms.

In this study, we have conducted a linguistic and logical analysis of the social, legal, ethical, empathetic, and cultural (SLEEC) rules obtained through the elicitation method outlined by Townsend et al. (2022). We have established a precise logical representation of SLEEC rules and investigated their computational characteristics. Furthermore, we have demonstrated how this definition can be effortlessly employed for SLEEC-compliant automated reasoning by leveraging SAT solvers or logic programming frameworks, all of which are conveniently readily-available on robot development platforms.

Ethical Statement

Acknowledgements

References

Andres, B.; Rajaratnam, D.; Sabuncu, O.; and Schaub, T. 2015. Integrating ASP into ROS for Reasoning in Robots. In *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, 69–82. Springer.

Barker-Plummer, D.; Barwise, J.; and Etchemendy, J. 2011. *Language, Proof and Logic*. CSLI Publications.

Chopra, A.; van der Torre, L.; Verhagen, H.; and Villata, S., eds. 2018. *Handbook of Normative Multiagent Systems*. London, UK: College Publications. ISBN 978-1-84890-285-9.

Dowling, W. F.; and Gallier, J. H. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3): 267–284.

Feng, N.; Marsso, L.; Yaman, S. G.; Townsend, B.; Cavalcanti, A.; Calinescu, R.; and Chechik, M. 2023. Towards a Formal Framework for Normative Requirements Elicitation. In *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*, 1776–1780. IEEE.

Gabbay, D.; Horty, J.; Parent, X.; van der Meyden, R.; and van der Torre, L., eds. 2013. *Handbook of Deontic Logic and Normative Systems*. London, UK: College Publications. ISBN 978-1-84890-132-2.

Gabbay, D.; Horty, J.; Parent, X.; van der Meyden, R.; and van der Torre, L., eds. 2021. *Handbook of Deontic Logic and Normative Systems, Volume 2*. London, UK: College Publications. ISBN 978-1-84890-363-0.

Grice, H. P. 1975. Logic and Conversation. In Cole, P.; and Morgan, J. L., eds., *Syntax and Semantics: Vol. 3: Speech Acts*, 41–58. New York: Academic Press.

Meyer, J.-J. C.; and Wieringa, R. J., eds. 1994. *Deontic Logic in Computer Science: Normative System Specification*. John Wiley and Sons Ltd. ISBN 0471937436.

Quine, W. V. O. 1959. *Methods of Logic*. New York: Henry Holt and Company, Inc. Revised Edition.

Tenorth, M.; and Beetz, M. 2017. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence*, 247: 151–169.

Townsend, B.; Paterson, C.; Arvind, T. T.; Nemirovsky, G.; Calinescu, R.; Cavalcanti, A.; Habli, I.; and Thomas, A. 2022. From Pluralistic Normative Principles to Autonomous-Agent Rules. *Minds and Machines*, 32(4): 683–715.

Troquard, N.; De Sanctis, M.; Inverardi, P.; Pelliccione, P.; and Scoccia, G. L. 2023. Social, Legal, Ethical, Empathetic, and Cultural Rules: Compilation and Reasoning (Extended Version). arXiv:2312.09699.

Yaman, S. G.; Burholt, C.; Jones, M.; Calinescu, R.; and Cavalcanti, A. 2023. Specification and Validation of Normative Rules for Autonomous Agents. In Lambers, L.; and Uchitel, S., eds., *Fundamental Approaches to Software Engineering*, 241–248. Cham: Springer Nature Switzerland.