Towards Efficient Norm-Aware Robots' Decision Making Using Datalog

Mahrokh Mirani[1], Franco Raimondi[1] and Nicolas Troquard[1]

[1]Gran Sasso Science Institute (GSSI), Viale Luigi Rendina 26-28, 67100 L'Aquila, Italy

Abstract

Social, Legal, Ethical, Empathetic, and Cultural (SLEEC) requirements are a key concern for the implementation of autonomous agents, such as robots. Existing work has focused on the definition of domain-specific languages for SLEEC rules, and on approaches to ensure the consistency of these rules.

Motivated by the fact that normative deliberation is likely to happen on board, in devices with limited resources, we investigate methods for the *efficient* computation of obligations that arise from SLEEC rules.

More in detail, we show that encoding SLEEC rules in Datalog can provide a scalable solution for the computation of obligations. We demonstrate this through two use cases: we compute the obligations of a robotic assistant in a care home, and the obligations of an online personal assistant performing charity donations on behalf of a user. Our results show that a declarative approach can scale both in Boolean and in arithmetic domains to very large instances.

Keywords

Normative rules, Efficient reasoning, Datalog

1. Introduction

The current trend in autonomous, AI-based robotic applications to support daily human activities, together with new regulations coming into effect, such as the EU AI Act[1], are attracting a growing interest around the issue of specifying and reasoning about *norms* that regulate human - (autonomous) robot interactions (see [1] and references therein). One approach that has been suggested to manage these norms is the adoption of a domain-specific language and an elicitation methodology for Social, Legal, Ethical, Empathetic and Cultural (SLEEC) rules. More in detail, Townsend et al. [2] propose a methodology to elicit normative rules for multi-agent systems. The process follows an iterative approach, in which a default rule is progressively refined to accommodate exceptions. Following it, philosophers, ethicists, lawyers, domain experts, and other professionals can derive normative rules specific to a particular domain.

Townsend et al. [2] illustrate the elicitation process on a running example and obtain the rule presented in Example 1. We employ Boolean atomic propositions in square brackets to capture specific facts. Notice also the presence of the keyword UNLESS introducing what is called a *defeater* for the preceding obligation; see Section 2 for additional details.

Example 1. *When the user tells the robot to open the curtains [ask] then the robot should open the curtains [open], UNLESS the user is 'undressed' [¬dressed] in which case the robot does not open the curtains [not_open] and tells the user 'the curtains cannot be opened while you, the user, are undressed,' [say] UNLESS the user is 'highly distressed' [highly_distressed] in which case the robot opens the curtains [open].*

3rd Workshop on Bias, Ethical AI, Explainability and the Role of Logic and Logic Programming (BEWARE24), co-located with AIxIA 2024, November 25-28, 2024, Bolzano, Italy

mahrokh.mirani@gssi.it (M. Mirani); franco.raimondi@gssi.it (F. Raimondi); nicolas.troquard@gssi.it (N. Troquard)
https://orcid.org/0009-0003-9052-5408 (M. Mirani); https://orcid.org/0000-0002-9508-7713 (F. Raimondi); https://orcid.org/0000-0002-5763-6080 (N. Troquard)

[1]https://eur-lex.europa.eu/eli/reg/2024/1689/oj

Towards tractable norm-aware decision making. The rules produced by the elicitation are expressed in natural language. Previous research demonstrated how to compile them into formal representations using a domain-specific language [3]. However, general reasoning with these formalizations is computationally intractable. Troquard et al. [4] show that reasoning with propositional SLEEC rules is NP-hard in general. They also propose implementations of the SLEEC rules in Answer Set Programming and Prolog, which are in general Σ_2^p-hard and undecidable, respectively. Feng et al. [5] translate SLEEC rules into a predicate logic that is also undecidable.

As the amount of data to be processed in a system grows, the less likely a norm-informed decision will be possible, especially when time performance is a concern.

Contribution. This work investigates the application of logic programming, and in particular Datalog, to the problem of computing the obligations that arise from SLEEC rules. Logic programming has been proposed in the past to express norms in AI systems to achieve transparency and explainability [6]. In our work we focus on efficiency, and we do so by identifying specific fragments of the general syntax of SLEEC rules: in particular, we assume (1) that we can partition the domain into observations and obligations, and (2) that obligations take effect immediately and do not have time constraints. Formally, given a set of SLEEC rules, given a set of *observations* A obtained through sensors, and given an obligation $B(\vec{c})$, does $B(\vec{c})$ hold? We call this problem Obligation Inference. We provide a translation from SLEEC rules to Datalog and we employ Soufflé[2] to validate our approach on two use cases: a robot in a care home (Boolean domain, scaling on the number of rooms and the number of patients) and a software agent acting on behalf of a user to select donations for charities (numeric domain, scaling on the number of users and the number of charities). Our preliminary results show that the approach can scale to very large instances (millions of users and rooms; thousands of charities).

The rest of the paper is organised as follows. We provide a formalization of SLEEC rule fragment we address, its translation into classical logic formulae, and the Datalog encoding in Section 2. We report experimental results in Section 3 and we conclude in Section 4.

2. Making Obligation Inference scale

As discussed in [2] and in [3], SLEEC rules can be expressed in natural language or in a domain-specific language using the pattern **if** `condition` **then** `obligation` **UNLESS** `condition` **in which case** `obligation` **UNLESS** `condition` **in which case** `obligation`... We formalize this syntax as follows:

Definition 1. *SLEEC rules are expressions of the form*

IF A_0 *THEN* B_0,
UNLESS A_1 *IN WHICH CASE* B_1,
UNLESS A_2 *IN WHICH CASE* B_2,
...
UNLESS A_n *IN WHICH CASE* B_n.

where, A_i *and* B_i *are arbitrary formulae. We make the additional assumption that* A_i *and* B_i *are Boolean expressions built using the standard connectives (conjunction, disjunction, negation, etc.), in which the atoms are either Boolean atoms (i.e,* $p, q, \dots$*) or atomic predicates for first order relations (e.g.,* $R(x, y)$*,* $x \leq y$*, etc.). We use the notation* $SR = \{A_i, B_i\}_{i \in \{0,\dots,n\}}$ *to denote a generic SLEEC rule, representing the* n *pairs of condition (or defeater) and obligation.*

To illustrate this with propositional atoms, consider the SLEEC rule of Example 1. We have that all expressions are Boolean and $A_0 = ask$, $B_0 = open$, $A_1 = \neg dressed$, $B_1 = not\_open \wedge say$, $A_2 = highly\_distressed$, and $B_2 = open$. Notice that the atom not_open should be read as "the robot has an obligation *not* to open the curtains", and is therefore treated differently from the atomic proposition $open$.

[2] https://souffle-lang.github.io/

In the following, we exploit the simple fact that the A_i's are about states of affairs that are 'sensed', and the B_i's are obligations, usually in the form of actions required from the robot, and thus are constituted of Boolean atoms and atomic predicates from two independent domains, SENSING and OBLIGATIONS, such that SENSING $\cap$ OBLIGATIONS $= \emptyset$.

Compiling SLEEC rules into classical logic. We first remark that the semantics of SLEEC rules adopted by Feng et al. [7, 5] induces a preference ordering in the obligations. As an example, consider again Example 1: the last obligation corresponding to opening the window when the user is highly distressed has a "higher priority" than the previous obligations. Thus, we compute obligations starting from the highest index. Formally:

Definition 2. *Given a SLEEC rule $SR = \{A_i, B_i\}_{i \in \{0,\dots,n\}}$, the corresponding logic encoding is as follows:*

$$
\begin{gathered}
A_0 \wedge A_n \rightarrow B_n \\
\bigwedge \\
A_0 \wedge \neg A_n \wedge A_{n-1} \rightarrow B_{n-1} \\
\bigwedge \\
\dots \\
\bigwedge \\
A_0 \wedge \neg A_n \wedge \neg A_{n-1} \wedge \dots \wedge \neg A_1 \rightarrow B_0
\end{gathered}
$$

Considering again Example 1, the corresponding logic translation is:

$$
\begin{aligned}
&((ask \wedge highly\_distressed) \rightarrow open) \\
&\wedge ((ask \wedge \neg highly\_distressed \wedge \neg dressed) \rightarrow (not\_open \wedge say)) \\
&\wedge ((ask \wedge \neg highly\_distressed \wedge \neg\neg dressed) \rightarrow open)
\end{aligned}
$$

(where the double negation for $dressed$ is left for clarity).

In terms of complexity, notice that the size of this translation is polynomial in the size of the rule, specifically $O(|SR|^2)$.

Compiling SLEEC rules into Datalog. For our Datalog translation we use the tool Soufflé[3]. Given the translation of SLEEC rules into logic in Definition 2, the Datalog translation follows a nearly identical pattern. We introduce a relation for each atomic proposition or predicate A_i^j and for each obligation B_k^l. Boolean expressions over atoms in SENSING and over OBLIGATIONS are translated directly, with the only additional assumption that negations should be *stratified*[4]. The relations corresponding to the atoms in SENSING are user-provided through facts, while the relations corresponding to the atoms in OBLIGATIONS are obtained through rules and are written to a file.

As a concrete example, the following Datalog snippet is a possible encoding of Example 1:

```
// These are atoms from Sensing
.decl user(u: symbol) // u is a user
.decl ask(action: symbol) // the user asks to perform an action
.decl dressed(u: symbol) // the user is dressed
.decl highly_distressed(u: symbol) // the user is distressed
.input user
.input ask
.input dressed
```

[3] https://souffle-lang.github.io

[4] See https://souffle-lang.github.io/rules#negation-in-rules. In practice, since the relations in SENSING and OBLIGATIONS are over external entities such as windows and users, stratification is usually possible. See examples in Section 3.

```
.input highly_distressed

// These are atoms from Obligations
.decl open(window: symbol) // the robot has an obligation to open the
    window for the user
.decl not_open(window: symbol) // the robot has an obligation not to open
    the window for the user
.decl say(message: symbol) // the robot has an obligation to tell the
    message to the user

// This is the encoding of the rules
open("window") :- ask("open window"), highly_distressed(u), user(u).
not_open("window") :- ask("open window"), !highly_distressed(u), !dressed(u
    ), user(u).
say("cannot open window") :- ask("open window"), !highly_distressed(u), !
    dressed(u), user(u).
open("window") :- ask("open window"), !highly_distressed(u), dressed(u),
    user(u).
```

3. Experimental Results

To examine the applicability of our translation of the SLEEC rules to Datalog we conduct two experiments. The first one is an extension of the Robot Assisted Dressing from Townsend et al. [2] and encoded in Example 1, which only includes Boolean conditions: we extend the example by considering multiple users and multiple rooms. The second example is a charity donation program which also includes arithmetic conditions. We investigate the scalability of these two programs with respect to the input data. The experiments are carried out on a machine with M1 Pro chip with eight cores and 16 GB RAM, running macOS Sonoma (14.5) and Soufflé version 2.4.1.

3.1. Robot Assisted Dressing

In order to show pertinent and significant experiments, we extend Example 1 from the literature to handle a varying number of participating objects (rooms and users). In this variant of the scenario, there are multiple users and multiple rooms, and a single robot. Each user and window are in a given room, encoded as $in\_room(x, room)$. The problem can be elicited as follows:

Example 2. *When a user u_i asks the robot has an obligation to open the curtains of window w_j (encoded as $ask(u_i, w_j)$) then the robot should open the curtains of window w_j ($open(w_j)$), UNLESS the user u_i is 'undressed' ($\neg dressed(u_i)$) in which case the robot has an obligation not to open the curtains of window w_j ($not\_open(w_j)$) and tells the user 'the curtains of window w_j cannot be opened ($say(u)$), UNLESS the user u_i is 'highly distressed' ($highly\_distress(u_i)$) in which case the robot opens the curtains of window w_j ($open(w_j)$), UNLESS the user u_i is not in the same room as window w_j in which case the robot does not open the curtains of window w_j ($not\_open(w_j)$) and tells the user 'the curtains of window w_j cannot be opened ($say(u)$).*

Applying the compilation presented in Section 2, one obtains the following rules (informally):

- if user asks, and user and window are not in the same room, then not open and say something
- if user asks, user is highly distressed, and user and window are in the same room, then open
- if user asks, user is not dressed, user is not highly distressed, and user and window are in the same room, then not open and say something
- if user asks, user is dressed, user is not highly distressed, and user and window are in the same room, then open

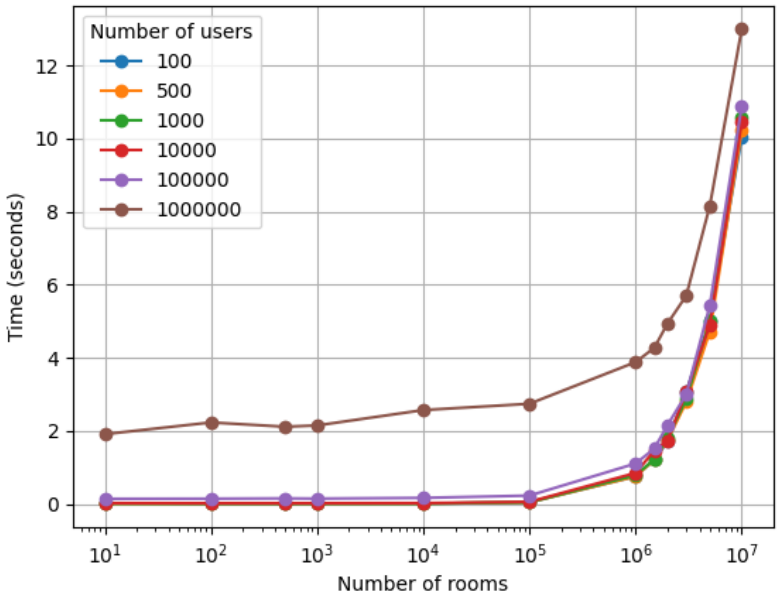

Figure 1: Execution time of assisted dressing robot example

The following Datalog snippet reports the key rules employed in the experiments:

```
.decl aux_same_room(thing1: symbol, thing2: symbol) // thing1 and thing2
    are in the same room
.decl aux_ask_open(user: symbol, robot: symbol, window: symbol) // user
    asks robot to open curtains of window

aux_same_room(x,y) :- in_room(x,room), in_room(y,room).
aux_ask_open(u,r,w) :- robot(r), user(u),ask(u,a,w), a="open".

not_open(r,w,u) :- aux_ask_open(u,r,w), !dressed(u), !highly_distressed(u),
     aux_same_room(u,w).
not_open(r,w,u) :- aux_ask_open(u,r,w), !aux_same_room(u,w).
say(r,w,u) :- aux_ask_open(u,r,w), !dressed(u), !highly_distressed(u),
    aux_same_room(u,w).
say(r,w,u) :- aux_ask_open(u,r,w), !aux_same_room(u,w).
open(r,w,u) :- aux_ask_open(u,r,w), dressed(u), !highly_distressed(u),
    aux_same_room(u,w).
open(r,w,u) :- aux_ask_open(u,r,w), highly_distressed(u), aux_same_room(u,
    w).
```

Figure 1 reports the execution time for this example as a function of the number of users and the number of rooms. The number of rooms is plotted using a logarithmic scale on the X-axis. As shown in the picture, Soufflé can analyze 10^6 users in 10^7 rooms in approximately 12 seconds. The execution time increases linearly with the number of rooms (notice that the plot of Figure 1 has a logarithmic scale on the X-axis). Observe also that in this example the number of rules remains constant.

3.2. Charity Donation

In this novel scenario, a person configures a bot to represent them online. When choosing a charity among various options to make a donation, the bot must choose the charity with the minimum number of donations so far. In the case that there is more than one charity with the minimum amount of donations, the person has an ordered list that shows his/her preference over the charities. Assuming

that these charities are sorted according to the preference (charity 1 is the least preferred), we can elicit the problem as follows:

Example 3. *When the user wants to donate money then donate to charity 1, UNLESS charity 2 received the minimum amount of donations in which case donate to charity 2, UNLESS charity 3 has received the minimum amount of donations in which case donate to charity 3, . . . , UNLESS charity n has received the minimum amount of donations, in which case donate to charity n.*

Applying the compilation presented in Section 2, *in the case of three charities* one obtains the following rules (informally):

- if user wants to donate, and charity 3 received the minimum amount of donations, then donate to charity 3;
- if user wants to donate, charity 2 received the minimum amount of donations, and charity 3 has not received the minimum amount of donations, then donate to charity 2;
- if user wants to donate, charity 2 has not received the minimum amount of donations, and charity 3 has not received the minimum amount of donations, then donate to charity 1;

This example can be scaled both in the number of users that have previously donated and also in the number of charities which, differently from the previous example, results in an increasing number of rules. We formulated this problem in Datalog as explained previously. In this specific case, we use a variable of type `number` to count the donations received by each charity, and the aggregation functions `count` and `min` to express the rules. As an example, the rule to donate to charity 2 is as follows:

```
bot_donates_to("c2") :- count_for_charity("c2",t2), minimum_val(t2),
                        count_for_charity("c3",t3), !minimum_val(t3).
```

where `count_for_charity("c2",t2)` is a predicate that encodes in `t2` the number of users that have donated for the second charity, and `minimum_val(t2)` is true if `t2` is the minimum value for charity donations.

Figure 2 shows the execution time for different numbers of users and charities. On the one hand, it can be observed that changes in the number of users don't affect performance significantly: for a given number of charities, the execution time for users from 100 to 1,000,000 is nearly identical. On the other hand, increasing the number of charities has a more significant impact due to an increase in the size of the SLEEC rule, reflected in an increased number of Datalog rules.

Overall, considering both the first and the second scenarios, we can conclude that Datalog seems to have excellent scalability performance in terms of the size of the input facts (also called data complexity). In terms of number of rules (program complexity), the performance does not seem to scale as well, even if Datalog can still handle hundreds of rules, up to 1,000 in less than 15 minutes, on a standard laptop and without optimizations nor pre-compilation.

4. Conclusion

There is an increasing need for autonomous systems capable of efficient norm-following decision making. Previous research has shown how to formalise Townsend et al. [2]'s SLEEC rules in various logics and how to reason about the consistency of these rules. In these logics the decision problem Obligation Inference considered here remains computationally complex, or even undecidable, in general. Obligation Inference asks whether an obligation follows from a set of SLEEC rules and a set of observations. In this paper, we assume that SLEEC rules are consistent and we proposed to translate SLEEC rules into Datalog programs, for rules not including time and built using either Boolean atoms or atomic relations.

We illustrated our approach with two scenarios and evaluated the performance of the Datalog solver Soufflé. These preliminary experiments indicate that Datalog is well-suited for representing SLEEC principles, for inferring autonomous agents' obligations, and with excellent scalability features.

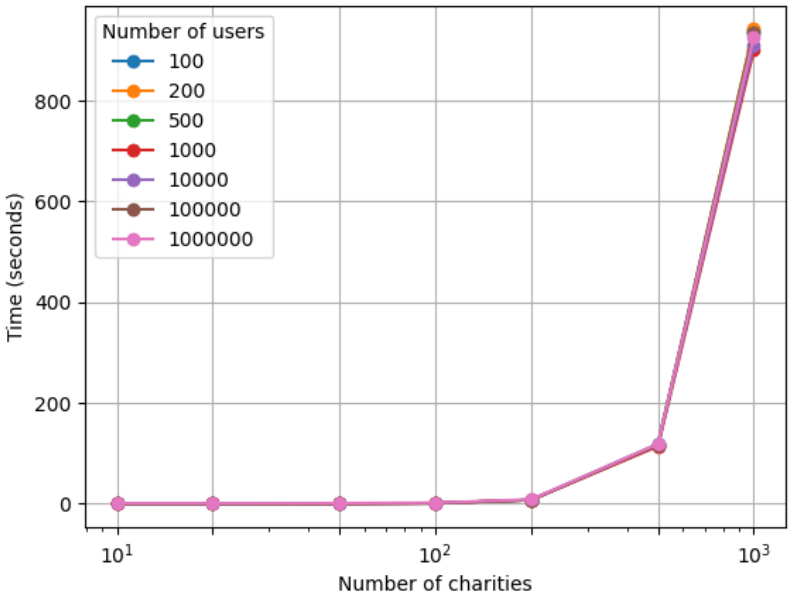

Figure 2: Execution time of charity donation example

In the future, we plan to apply our approach to more scenarios in order to understand the possible limitations. We also plan to investigate the complexity of the Obligation Inference in various fragments, possibly extended with time. We also intend to integrate it into robotic platforms to perform the computation of obligations at run-time, for instance as a ROS module [8].

References

[1] M. De Sanctis, P. Inverardi, P. Pelliccione, Do modern systems require new quality dimensions?, in: A. Bertolino, J. Pascoal Faria, P. Lago, L. Semini (Eds.), Quality of Information and Communications Technology, Springer Nature Switzerland, Cham, 2024, pp. 83–90.

[2] B. Townsend, C. Paterson, T. T. Arvind, G. Nemirovsky, R. Calinescu, A. Cavalcanti, I. Habli, A. Thomas, From pluralistic normative principles to autonomous-agent rules, Minds and Machines 32 (2022) 683–715. URL: https://doi.org/10.1007/s11023-022-09614-w. doi:10.1007/s11023-022-09614-w.

[3] S. G. Yaman, C. Burholt, M. Jones, R. Calinescu, A. Cavalcanti, Specification and validation of normative rules for autonomous agents, in: L. Lambers, S. Uchitel (Eds.), Fundamental Approaches to Software Engineering, Springer Nature Switzerland, Cham, 2023, pp. 241–248.

[4] N. Troquard, M. De Sanctis, P. Inverardi, P. Pelliccione, G. L. Scoccia, Social, legal, ethical, empathetic, and cultural rules: Compilation and reasoning, in: M. J. Wooldridge, J. G. Dy, S. Natarajan (Eds.), Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, AAAI Press, 2024, pp. 22385–22392. URL: https://doi.org/10.1609/aaai.v38i20.30245. doi:10.1609/AAAI.V38I20.30245.

[5] N. Feng, L. Marsso, S. G. Yaman, Y. Baatartogtokh, R. Ayad, V. O. de Mello, B. A. Townsend, I. Standen, I. Stefanakos, C. Imrie, G. N. Rodrigues, A. Cavalcanti, R. Calinescu, M. Chechik, Analyzing and debugging normative requirements via satisfiability checking, in: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal,

April 14-20, 2024, ACM, 2024, pp. 214:1–214:12. URL: https://doi.org/10.1145/3597503.3639093. doi:10.1145/3597503.3639093.

[6] A. Dyoub, S. Costantini, F. A. Lisi, Logic programming and machine ethics, Electronic Proceedings in Theoretical Computer Science 325 (2020) 6–17. URL: http://dx.doi.org/10.4204/EPTCS.325.6. doi:10.4204/eptcs.325.6.

[7] N. Feng, L. Marsso, S. G. Yaman, B. Townsend, A. Cavalcanti, R. Calinescu, M. Chechik, 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, IEEE, 2023, pp. 1776–1780.

[8] Stanford Artificial Intelligence Laboratory et al., Robotic operating system, 2018. URL: https://www.ros.org.