Handling Change in Normative Specifications

(Extended Abstract)

Duangtida Athakravi, Domenico Corapi, Alessandra Russo
Department of Computing,Imperial College London
{da407, d.corapi, a.russo}@imperial.ac.uk

Marina De Vos, Julian Padget
Dept. of Computer Science,
University of Bath
{mdv,jap}@cs.bath.ac.uk

Ken Satoh
National Institute of
Informatics
ksatoh@nii.ac.jp

ABSTRACT

Normative frameworks provide a means to address the governance of open systems, by offering a mechanism to express responsibilities and permissions of the individual participants with respect to the entire system without compromising their autonomy. Careful design is crucial if it is to meet its requirements. Tools that support the design process can be of great benefit. In this paper, we describe a method for choosing the appropriate change in the normative specification, using impact analysis of the critical consequences being preserved or rejected by the change.

Categories and Subject Descriptors

H.4 [**Programming Techniques**]: Logic Programming

General Terms

Theory, Verification, Algorithms, Design

Keywords

Design, Normative systems, Inductive Logic Programming

1. INTRODUCTION

Normative frameworks provide a powerful tool for governing open systems by providing guidelines for the behaviour of the individual components without regimentation [4]. Using a formal declarative language to specify the behaviour of a normative system gives the system's designer a means to verify the compliance of the system with respect to desirable behaviours or properties [2, 1]. However, when errors are detected, the identification of what changes to make is often a difficult and error-prone manual process. Corapi et al [3] have shown how Inductive Logic Programming (ILP) can support the elaboration of normative specifications, modelled in Answer Set Programming (ASP), by learning possible changes that would make partial normative specification consistently compliant with given use-cases.

This paper addresses the problem of how to choose between alternative changes by analysing their impact on the specification. We use a notion of *relevant literals*, i.e. elements of the domain that are critical for discriminating between the suggested changes. We describe how these literals are computed using test generation [5]. Ranking them is based on the number of changes that they would discard. This ranking ensures that the revisions are discriminated against in the most effective way.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

2. HANDLING CHANGE

Our framework for handling change during the elaboration of normative specifications addresses the limitation of our previous work [3], where it is the designer's responsibility to choose the most appropriate revision from multiple possible revisions computed by a learner. In real applications the number of suggested changes can be large, making an automated criteria essential for selecting the most effective change. As shown in Figure 1, our framework combines the approach in [3], with two additional steps for computing and scoring relevant literals. The most highly ranked literal is then queried to the designer, who can then specify its truth value. Based on the answer of the designer, those changes that are refuted by the relevant literals are discarded. Literals that are dependent on the highly ranked one could be used to further reduce the hypothesis space.

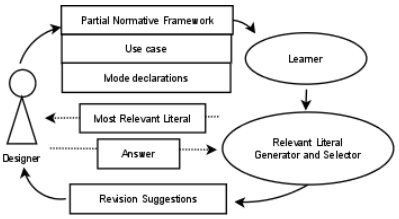

Figure 1: Framework for handling change

2.1 Abducing Relevant Literals

Our definition of relevant literal is based on the notion of relevant test given in [5].

Definition 1. (*Relevant Literal*) Let $\langle T, O\rangle$ be a use-case consisting of a trace T and desired outcome O, given the existing (partial) normative specification Σ, and the set of hypothesis representing the suggested revisions HYP. A literal l is relevant if:

1. $\Sigma \wedge T \wedge O \wedge H$ is satisfiable for all $H_i \in HYP$
2. $T \wedge O \wedge l$ is an abductive explanation for $\bigvee_{H_i \in HYP} \neg H_i$
3. $\Sigma \wedge T \wedge O \not\models \bigvee_{H_i \in HYP} \neg H_i$
4. $T \wedge O \wedge l$ is not an abductive explanation for $\neg H_i, \forall H_i \in HYP$

Conditions (1) and (3) state that all suggested revisions satisfy the use-case and are consistent with the normative specification. These are guaranteed by the correctness of the learner. Conditions (2) and (4) ensure that some but not all suggested revisions are rejected by the relevant literal. These are captured by integrity constraints used as goals of the following abductive program.

Let N_B be the static part of the normative specification, N_T the part containing rules that are revisable, $\langle T, O\rangle$ the use-cases applied for learning suggested revisions S, and let $C_H/2$ be the function that combines N_T with suggested revisions by representing them as hypotheses. The relevant literals are solutions of the abductive task $\langle B, Ab, G\rangle$ where:

$$B = N_B \cup T \cup C_H(N_T, S)$$
$$G = O \cup \neg(\bigwedge_{H_i \in C_H(N_T,S)} \neg H_i) \cup \neg(\bigwedge_{H_i \in C_H(N_T,S)} H_i)$$

and Ab is the set of ground instances of (possible) outcomes. The set E of relevant literals is a subset of Ab such that $B \cup E \models G$.

2.2 Ranking Relevant Literals

Ideally we want to be able to dismiss as many suggested revisions as possible, based on the truth value of relevant literals. We use the number of minimum hypotheses (or revisions) that a relevant literal may reject, in order to compare it against other relevant literals (we give a fractional score when a conjunction of literals is required to dismiss a hypothesis). Thus, the score of a relevant literal l is $s = minimum(n, m)$, where n is the number of suggested revisions that l rejects when true, and m is the number of suggested revisions that l rejects when false. The literals with maximum score are the most relevant literal. Literals with equal score could be further ranked according to the maximum number of hypotheses each one falsifies.

3. CASE STUDY

We have applied our approach to the file sharing agents example [3], where only VIP agents have permission to download a block of data without having previously shared its own block. We have used the same use-case for which the learned has produced many alternative changes. We have considered the following four of these changes, which were similar to each other:

1. $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(download(X, Y, B), I), holdsat(hasblock(Y, B), I).$
2. $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(download(X, Y, B), I), holdsat(hasblock(Y, B), I).$
   $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(viol(myDownload(Y, B2)), I),$
   $holdsat(hasblock(Y, B), I).$
3. $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(download(Y, Y, B), I).$
4. $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(download(X, Y, B), I), holdsat(hasblock(Y, B), I).$
   $occurred(myDownload(X, B), I) \leftarrow$
   $occurred(viol(myDownload(X, B2)), I),$
   $holdsat(hasblock(Y, B), I).$

Using our method and forming an abductive problem with background knowledge given by the above revisions and the original specification, we were able to identify the relevant literals and score them according to how many suggestions they could discard. Table 1 contains literals that could potentially dismiss suggestions (2) or (4). Using these values, we were able to identify the following most relevant literals:

$occurred(viol(myDownload(bob, x1)), i06)$
$occurred(viol(myDownload(bob, x2)), i06)$
$occurred(viol(myDownload(bob, x3)), i06)$
$occurred(viol(myDownload(bob, x4)), i06)$
$occurred(viol(myDownload(bob, x5)), i06)$

Any of the above literals would dismiss suggestions (2) and (4) when false, or only one of them when true, provided that $occurred(misuse(bob), i06)$ was false.

Relevant literal	Truth value	
	True	False
$occurred(viol(myDownload(alice, x1)), i06)$	0.0	2.0
$occurred(viol(myDownload(alice, x2)), i06)$	0.0	2.0
$occurred(viol(myDownload(alice, x3)), i06)$	0.0	2.0
$occurred(viol(myDownload(alice, x5)), i06)$	0.0	2.0
$occurred(viol(myDownload(bob, x1)), i06)$	0.5	2.0
$occurred(viol(myDownload(bob, x2)), i06)$	0.5	2.0
$occurred(viol(myDownload(bob, x3)), i06)$	0.5	2.0
$occurred(viol(myDownload(bob, x4)), i06)$	0.5	2.0
$occurred(viol(myDownload(bob, x5)), i06)$	0.5	2.0
$occurred(misuse(bob), i06)$	0.0	0.5

Table 1: Scoring of relevant literals

4. DISCUSSION

In this paper, we have tackled the problem of distinguishing between revisions over normative specifications through the use of test generation. By identifying comparable consequences of the suggested revisions, we are able to use them as a rationale for rejecting possible changes. Our case study provides an example of what outputs can be acquired by the proposed approach. However, as well as showing the outputs that could be used for selection criteria, it also demonstrates that the approach is unable to discriminate between all suggested revisions (no relevant literals that could dismiss the suggestion (1) or (3) were found).

For the future, the problem of the inability to distinguish some hypotheses (possibly by extending the use-case's trace) needs be adressed. An implementation of an automated system for normative revision for further evaluation of the technique is required. We also plan a modification to the scoring method that was used to take into account additional factors such as the length of the revision suggestion.

5. REFERENCES

[1] Alexander Artikis, Marek Sergot, and Jeremy Pitt. An executable specification of an argumentation protocol. In *Proceedings of Conference on Artificial Intelligence and Law (ICAIL)*, pages 1–11. ACM Press, 2003.

[2] Owen Cliffe, Marina De Vos, and Julian Padget. Answer set programming for representing and reasoning about virtual institutions. In Katsumi Inoue, Ken Satoh, and Francesca Toni, editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *Lecture Notes in Computer Science*, pages 60–79. Springer Berlin / Heidelberg, 2007.

[3] Domenico Corapi, Alessandra Russo, Marina De Vos, Julian A. Padget, and Ken Satoh. Normative design using inductive learning. *TPLP*, 11(4-5):783–799, 2011.

[4] D. Grossi, H. M. Aldewereld, and F. Dignum. Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, M.V. Dignum, N. Fornara, and E. Matson, editors, *COIN II*, pages 101–114. Springer, 2007.

[5] Sheila Mcilraith. Generating tests using abduction. In *Proc. Inter. Conf. on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 449–460, 1994.