Norm Refinement and Design through Inductive Learning

Domenico Corapi\textsuperscript{1}, Marina De Vos\textsuperscript{2}, Julian Padget\textsuperscript{2},
Alessandra Russo\textsuperscript{1}, and Ken Satoh\textsuperscript{3}

\textsuperscript{1} Department of Computing, Imperial College London
\{d.corapi,a.russo\}@imperial.ac.uk
\textsuperscript{2} Department of Computer Science, University of Bath
\{mdv,jap\}@cs.bath.ac.uk
\textsuperscript{3} National Institute of Informatics
ksatoh@nii.ac.jp

Abstract. In the physical world, the rules governing behaviour are debugged by
observing an outcome that was not intended and the addition of new constraints to
prevent the attainment of that outcome. We propose a similar approach to support
the incremental development of normative frameworks (also called institutions)
and demonstrate how this works through the validation and synthesis of norma-
tive rules using model generation and inductive learning. This is achieved by the
designer providing a set of \textit{use cases}, comprising collections of event traces that
describe how the system is used along with the desired outcome with respect
to the normative framework. The model generator encodes the description of the
current behaviour of the system. The current specification and the traces for which
current behaviour and expected behaviour do not match are given to the learning
framework to propose new rules that revise the existing norm set in order to in-
hbit the unwanted behaviour. The elaboration of a normative system can then be
viewed as a semi-automatic, iterative process for the detection of incompleteness
or incorrectness of the existing normative rules, with respect to desired properties,
and the construction of potential additional rules for the normative system.

1 Introduction

Norms and regulations play an important role in the governance of human society. So-
cial rules such as laws, conventions and contracts prescribe and regulate our behaviour.
However it is possible for us to break these rules at our discretion and face the conse-
quences. By providing the means to describe and reason about norms in a computational
context, normative frameworks (also called institutions or virtual organisations) may be
applied to software systems allowing for automated reasoning about the consequences
of socially acceptable and unacceptable behaviour. This can be achieved by monitor-
ing the permissions, empowerment and obligations of the participants and generating
violations when norms are not followed.

\* This work is partially supported through the EU Framework 7 project \textit{ALIVE} (FP7-IST-
215890), and the EPSRC PRiMMA project (EP/F023294/1).
The formal model put forward in [11] and its corresponding operationalisation through Answer Set Programming (ASP) [3, 26] aims to support the top-down design of normative frameworks. \textit{AnsProlog} is a knowledge representation language that allows the programmer to describe a problem and required properties on the solutions in an intuitive way. Programs consist of rules interpreted under the answer set semantics. Answer set solvers, like CLASP [25] or SMODELS [35], can be used to reason about the given \textit{AnsProlog} specification, by returning acceptable solutions in the form of traces, as answer sets. In a similar way, the correctness of the specification with respect to given properties can be verified.

Currently, the elaboration of behavioural rules and norms is an error-prone process that relies on the manual efforts of the designer and would, therefore, benefit from automated support. In this paper, we present an inductive logic programming (ILP) [33] approach for the extraction of norms and behavioural rules from a set of use cases. The approach is intended as a design support tool for normative frameworks. Complex systems are hard to model and even if testing of properties is possible, sometimes it is hard to identify missing or incorrect rules. In some cases, e.g. legal reasoning, the abstract specification of the system can be in part given in terms of specific instances and use cases that ultimately drive the design process and are used to assess it. We propose a design support tool that employs use-cases, i.e. traces together with their expected normative behaviour, to assist in the revision of a normative framework. The system is correct when none of the traces are considered dysfunctional, i.e. they match the expected normative behaviour. When a dysfunctional trace is encountered the normative specification needs to be adjusted: the task is to refine the given description by learning missing norms and/or behavioural rules that, added to the description, entail the expected behaviour over the traces. We show how this task can be naturally represented as a non-monotonic ILP problem in which the partial description of the normative system provides the background knowledge and the expected behaviour comprises the examples. In particular, we show how a given \textit{AnsProlog} program and traces can be reformulated into an ILP representation that makes essential use of negation in inducing missing parts of the specification. As the resulting learning problem is inherently non-monotonic, we use a non-monotonic ILP system, called TAL [14], to compute the missing specification from the traces and the initial description.

Given the declarative nature of ASP, the computational paradigm used for our normative frameworks, we needed to adopt a declarative learning approach as we aim to learn declarative specifications. This differs from other approaches, such as reinforcement learning whereby norms or policies are learned as outcomes of estimation and optimisation processes. Such types of policies are not directly representable in a declarative format and are quite different in nature from the work reported here.

The paper is organised as follows. Section 2 presents some background material on the normative framework, while Section 3 introduces the non-monotonic ILP system used in our proposed approach. Section 4 describes the \textit{AnsProlog} modelling of normative frameworks. Section 5 illustrates how the revision task can be formulated into an ILP problem, and how the generated ILP hypothesis can be reformulated as norms and behaviour rules within the \textit{AnsProlog} representation. In Section 6 we illustrate