Modelling Normative Frameworks Using Answer Set Programing

Owen Cliffe, Marina De Vos, and Julian Padget

Department of Computer Science
University of Bath, BATH BA2 7AY, UK
{occ,mdv,jap}@cs.bath.ac.uk

Abstract. Norms and regulations play an important role in the governance of human society. Social 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 consequences. By providing the means to describe and reason about norms in a computational context, normative frameworks may be applied to software systems allowing for automated reasoning about the consequences of socially acceptable and unacceptable behaviour. In this paper, we outline our mathematical formulation for normative frameworks and describe how its semantics can be represented in ASP, thus enabling the construction of models of normative systems that can be subjected to formal verification and that can act as functional repositories of normative knowledge for the software components that participate in them.

1 Introduction

Normative frameworks are provide a mechanism to capture and reason about "correct" and "incorrect" behaviour within a certain context. The participants of a normative framework are governed by social norms and regulations. The framework monitors the permissions, empowerment and obligations of their participants and generate violations when norms are not adhered to. Information of the norms and the effects of participants actions is stored in the state of the framework. The constant change of the state over time as a result of these actions provides participants information about each others behaviour. The information can also be used by the designer to verify the normative structures of the framework. In this paper we will only look mainly at the latter.

We do not include an extensive and detailed case for the purpose and value of normative frameworks here—this can be found in [2] and [1]— as this paper focuses on applications of answer set programming and because of the need for brevity. What we will present is: (i) an outline of a formal event-based model of the specification of normative frameworks that captures all the essential properties, namely empowerment, permission, obligation and violation (ii) a summary of its formal translation to ASP that is sound and complete (essentially a long mechanical case analysis which not included here), resulting in a decidable and executable model of an normative frameworks that can be used for querying and verification.

2 The Normative Model

The components of a normative framework are events, which change the states and fluents, that characterize the state. The framework defines the interplay between these concepts over time and we here give the syntax and semantics.

2.1 Syntax

We define a normative framework as a 5-tuple $I := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ consisting of a set of events $\mathcal{E}$, a set of fluents $\mathcal{F}$, a set of causal rules $\mathcal{C}$, a set of generation rules $\mathcal{G}$ and an initial state Δ.

Events $\mathcal{E}$. Each normative framework defines a set of event symbols $e \in \mathcal{E}$, each of which denotes a type of event that may occur. We identify two disjoint subsets, $\mathcal{E}_{ex}$ consisting of *exogenous events* and $\mathcal{E}_{inst}$ consisting of *normative events*.

- **Exogenous events $\mathcal{E}_{ex}$:** consists of events that fall outside the scope of the normative framework. These may include agent communication events such as $tell(a, b, yes)$ (a tells b "yes") and other externally-defined events, such as time-outs. The exogenous events are in effect the interface to the normative framework. It is the responsibility of the designer of the specification to define this set carefully.
- **Normative events $\mathcal{E}_{inst}$:** that in contrast to exogenous events are those events that are generated *within* the normative framework as a, possibly indirect, consequence of an exogenous event.

  We identify two disjoint subsets of normative events: *normative actions*, $\mathcal{E}_{act}$ that capture significant changes in normative state, and *violation events*, $\mathcal{E}_{viol}$ denoting points at which violations have occurred. The set of violation events is defined such that it contains at least one violation event corresponding to each normative action and each exogenous event: $\forall e \in \mathcal{E}_{act} \cup \mathcal{E}_{ex} : viol(e) \in \mathcal{E}_{viol}$. Specifications may also define additional violation events not directly associated with particular exogenous or normative actions. Violation and sanction play an important role in the specification of normative frameworks. Violations may arise either from explicit generation, from the occurrence of a non-permitted event, or from the failure to fulfil an obligation. In these cases sanctions that may include obligations on violating agents or other agents and/or changes in agents' permission to do certain actions, may then be expressed as consequences of the occurrence of the associated violation event in the subsequent normative state.

Normative Fluents $\mathcal{F}$. We now turn to the definition of the normative state which we model through the definition of a set of fluent properties. We make a distinction between *normative fluents*, which express normative properties of the state such as permissions, powers and obligations, and *domain fluents* which correspond to properties which are specific to the normative framework itself. In both cases we model fluents as propositions which may be true or false in a given normative state. We define the set $\mathcal{D}$ to include all normative framework-specific fluents which may be true in the normative state. The set of normative fluents is broken down into sets of fluents for powers, permissions and obligations as follows:

– $\mathcal{W}$: A set of normative power fluents of the form $\mathrm{pow}(e) : e \in \mathcal{E}_{act}$ where each power proposition denotes the capability of some action e to be brought about in the normative framework.
– $\mathcal{P}$: A set of event permission fluents: $\mathrm{perm}(e) : e \in \mathcal{E}_{act} \cup \mathcal{E}_{ex}$, where each permission proposition denotes that it is permitted for action e to be brought about. We do not define a proposition for "forbidden" but just treat it as the absence of permission for that event to be brought about.
– $\mathcal{O}$: A set of obligations, of the form $\mathrm{obl}(e, d, v)$ with $e \in \mathcal{E}, d \in \mathcal{E}, v \in \mathcal{E}_{viol}$. The presence of an obligation fluent in the normative state denotes that event e should occur before the (deadline) event d or violation v will be generated.

The set of states, Σ, of an normative framework is defined as: $\Sigma = 2^{\mathcal{F}^*}$ but not all of these states may be reachable. Rules in the normative framework may have an effect in multiple normative states, and this effect may be conditional on the normative state. In order to qualify the effect of normative rules over a specific set of states, we define a language $\mathcal{X}$ of *state formulae* that allows us to specify to which normative states the rule may apply. The set of state formulae is defined as: $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$

Causal Rules $\mathcal{C}$. Each normative framework has a relation $\mathcal{C}$ that describes which fluents are initiated and terminated by the performance of a given action in a state matching some expression. The function is expressed as $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$, where the first set in the range of the function describes which fluents are initiated by the given event and the second those fluents terminated by the event. We use the notation $\mathcal{C}^{\uparrow}(\phi, e)$ to denote the fluents which are initiated by the event e in a state matching ϕ and $\mathcal{C}^{\downarrow}(\phi, e)$ to denote those fluents which are terminated by event e in a state matching ϕ.

Generation Rules $\mathcal{G}$. Each normative framework defines an event generation function $\mathcal{G}$ which describes when the performance of one event counts-as, or generates, another: $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$. The generation of events may be conditional on some properties in the normative state, and one event may generate multiple (consequent) events. It should be noted that this relation describes explicitly specified relationships between events in the normative framework. There are cases when events may be generated which are not in this relation, for instance in the case of unsatisfied obligations. Additionally this function represents cases where event generation *may* occur, however as normative events require empowerment, they will only be generated when this property holds.

Initial State Δ. Each normative framework defines the set $\Delta \subseteq \mathcal{F}$ that denotes the set proposition that shall hold when the normative framework is created.

2.2 Semantics

Due to space restrictions, we are only able to provide an informal overview of our semantics for normative frameworks. A full discussion with definitions can be found in [2]. The semantics of this framework is defined over a sequence, called traces, of exogenous events. Starting from the initial state, each exogenous event will be responsible for a state change, i.e. a set fluent of additions and deletions. Using the event generation function, the transitive closure of all events that count as this exogenous event are being obtained. Add to this set of events, all violations of events that were not permitted and obligations that were not fulfilled and we have the set of all events whose consequences will result in the new state. Using the consequence relation on this set of events, we obtain all fluents that need to be initiated and deleted from the current state in order to obtain the new state. So with each trace, we can associate a sequence of states. The combination of the two is referred to as a model or history. When verifying normative frameworks we are interested in finding out whether these models and traces have certain properties. To provide designer support for normative frameworks, we need a computation tools that can compute all traces/models or those that have specific features. In the next section, we demonstrate that ASP can be such a tool.

3 ASP Implementation

We summarize the representation of a normative framework in ASP through reference to the framework components identified earlier and a case-by-case translation of the mathematical model into an ASP representation. A crucial aspect of the model is how we have chosen to represent time and its relationship with the predicate `holdsat/2` to represent the truth of a particular fluent at a particular time. Time in this representation is linked to state transitions. Time zero is part of the initial state.

If the causal relation initiates a fluent, we want it to *hold at* the next time instant. A consequence of this representation of time is that unless explicit action is taken, a fluent does *not* hold at the *next* time instant, that is, the maintenance of inertia requires action. These two issues are addressed in this way:

$$\begin{aligned} f \in \mathcal{F}^* \ \texttt{holdsat(f,T1)} \leftarrow\ & \texttt{initiated(f,T)}, \texttt{instant(T)}, \\ & \texttt{instant(T1)}, \texttt{next(T,T1)}. \\ f \in \mathcal{F}^* \ \texttt{holdsat(f,T1)} \leftarrow\ & \texttt{holdsat(f,T)}, \\ & \textbf{not}\ \texttt{terminated(f,T)}, \texttt{instant(T)}, \\ & \texttt{instant(T1)}, \texttt{next(T,T1)}. \end{aligned}$$

Events cause the evolution of the normative state through the generation relation, thus an exogenous event is observed and its occurrence is added to the normative state:

$$\begin{aligned} e_{ex} \in \mathcal{E}_{ex} \ \texttt{occurred}(\texttt{e}_{\texttt{ex}},\texttt{T}) \leftarrow\ & \texttt{observed}(\texttt{e}_{\texttt{ex}},\texttt{T}), \texttt{instant(T)}, \\ & \texttt{instant(T1)}, \texttt{next(T,T1)}. \end{aligned}$$

To model conditions on the state we use the auxiliary function EX:

$$\begin{aligned} EX(x_1 \wedge x_2 \wedge \dots x_n, t_i) &\stackrel{\text{def}}{\equiv} EX(x_1,t_i), EX(x_2,t_i), \dots EX(x_n,t_i) \\ EX(\neg f, t_i) &\stackrel{\text{def}}{\equiv} \textbf{not}\ EX(f,t_i) \\ EX(f,t_i) &\stackrel{\text{def}}{\equiv} \texttt{holdsat}(\texttt{f},\texttt{t}_\texttt{i}) \end{aligned}$$

Thus, in the first of the following rules, the empowered event e_2 is added to the normative state through conventional generation as a result of the occurrence of e_1 and in the second, the event e_1 is added as a result of the occurrence of the violation event e_2, in both cases subject to a condition ϕ[1]:

$$\left.\begin{array}{l} e_1 \in \mathcal{E}, \phi \in \mathcal{X}, \\ e_2 \in \mathcal{G}(\phi, e_1), \\ e_2 \in \mathcal{E}_{act} \end{array}\right\} \begin{array}{ll} \mathtt{occurred(e_2,T)} \leftarrow & \mathtt{occurred(e_1,T), EX(\phi,T),} \\ & \mathtt{holdsat(pow(e_2),T), instant(T).} \end{array}$$

$$\left.\begin{array}{l} e_1 \in \mathcal{E}, \phi \in \mathcal{X}, \\ e_2 \in \mathcal{G}(\phi, e_1), \\ e_2 \in \mathcal{E}_{viol} \end{array}\right\} \begin{array}{ll} \mathtt{occurred(e_1,T)} \leftarrow & \mathtt{occurred(e_2,T), EX(\phi,T),} \\ & \mathtt{instant(T).} \end{array}$$

Violations arise when there is either an event that is not permitted (the first rule below) or the deadline event of an obligation (second rule):

$$\begin{array}{lll} e \in \mathcal{E}_{act} \cup \mathcal{E}_{ex} & \mathtt{occurred(viol(e),T)} \leftarrow & \mathtt{occurred(e,T),} \\ & & \textbf{not}\ \mathtt{holdsat(perm(e),T), instant(T).} \\ \mathrm{obl}(e,d,v) \in \mathcal{O} & \mathtt{occurred(v,T)} \leftarrow & \mathtt{holdsat(obl(e,d,v),T),} \\ & & \mathtt{occurred(d,T), instant(T).} \end{array}$$

The causal relation, covering the initiation and termination of fluents arising from some normative state, is captured in the following two rules:

$$\left.\begin{array}{l} e \in \mathcal{E}, \phi \in \mathcal{X}, \\ f \in \mathcal{C}^{\uparrow}(\phi, e) \end{array}\right\} \mathtt{initiated(f,T)} \leftarrow \mathtt{occurred(e,T), EX(\phi,T).}$$

$$\left.\begin{array}{l} e \in \mathcal{E}, \phi \in \mathcal{X}, \\ f \in \mathcal{C}^{\downarrow}(\phi, e) \end{array}\right\} \begin{array}{ll} \mathtt{terminated(f,T)} \leftarrow & \mathtt{occurred(e,T), EX(\phi,T),} \\ & \mathtt{instant(T).} \end{array}$$

Obligations are removed (terminated) subject to the occurrence of either the event necessary to satisfy the obligation or the deadline event associated with the obligation[2]:

$$\begin{array}{ll} \mathrm{obl}(e,d,v) \in \mathcal{O} & \mathtt{terminated(obl(e,d,v),T)} \leftarrow \mathtt{occurred(e,T), instant(T).} \\ \mathrm{obl}(e,d,v) \in \mathcal{O} & \mathtt{terminated(obl(e,d,v),T)} \leftarrow \mathtt{occurred(d,T), instant(T).} \end{array}$$

3.1 Traces and Queries

The purpose of constructing a (event-driven) model of a normative framework is to be able to test it in various ways through querying and this in turn depends on the construction of all the traces of events that could arise between one state and another. Thus query formulation and trace construction are intimately tied up. However, before traces can be generated, the program must be grounded, which means being explicit about the meaning of time instants and precisely how many there are, which in turn determines the length of the trace. For time instants $t_i : 0 \leq i \leq n$, we define the following three rules: $\mathtt{instant(t_i).}$, $\mathtt{next(t_i, t_{i+1}).}$ and $\mathtt{final(t_n).}$ denoting each ground instant of time, relative order and final state, respectively.

[1] Note that for violations, we do not need to verify that they are empowered.

[2] See violations above for how the deadline event is then translated into a violation.

The general trace program generates answer sets containing all possible combinations of n exogenous events, but by the addition of further constraints, the answer sets can be limited to those containing changes, being either the initiation or termination of fluents. We define an effective transition as:

$$\begin{aligned}
\texttt{changed(T)} &\leftarrow \texttt{initiated(F,T)}, \textbf{not } \texttt{holdsat(F,T)},\\
&\quad\ \texttt{ifluent(F)}, \texttt{instant(T)}.\\
\texttt{changed(T)} &\leftarrow \texttt{terminated(F,T)}, \texttt{holdsat(F,T)},\\
&\quad\ \texttt{ifluent(F)}, \texttt{instant(T)}.
\end{aligned}$$

so that something changes at time instant T (indicated by changed(T)) because there is either a fluent F which does (not) hold at time instant T which is initiated/terminated at time instant T. Of these, the *valid* states are those in which in which something has changed and the rest are discarded. This can further be enhanced to select traces of *up to* length n, but for the sake of space, those details are also not included here.

4 Extensions and Conclusions

So far we have written of a single normative framework and its model. However, there are two motivations to address the concept of several interacting normative frameworks: (i) one is that for design modularity, it is attractive to focus on small self-contained frameworks as well as in so doing enabling re-use, while (ii) another is that in the real world actors are typically simultaneously governed by many such frameworks. The technical details of this formalization and the notion of power and permission of one framework over another appear in [3]. The important point from the perspective of this paper is the ease with which such models can be combined by virtue of the ASP representation, so that given the necessary linkages between frameworks, their combination can be processed in just the same way as a single framework definition and subsequently a model for the combination is constructed that can answer queries about events and states in their combination.

The normative framework we have presented is designed to capture the essence of regulated environments and through the use of ASP enables its expression in a form that allows the construction of a computable model that can then be subjected to queries through the traces it admits. Subsequently, we have developed a high level action language to specify normative frameworks and a complementary query language.

References

1. Cliffe, O.: Specifying and Analysing Institutions in Multi-Agent Systems using Answer Set Programming. PhD thesis, University of Bath (2007)
2. Cliffe, O., De Vos, M., Padget, J.A.: Answer set programming for representing and reasoning about virtual institutions. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS (LNAI), vol. 4371, pp. 60–79. Springer, Heidelberg (2007)
3. Cliffe, O., De Vos, M., Padget, J.A.: Specifying and reasoning about multiple institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS (LNAI), vol. 4386, pp. 67–85. Springer, Heidelberg (2007)