Answer Set Programming for Representing and Reasoning about Virtual Institutions

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. It is recognised that institutions are potentially powerful means for making agent interactions effective and efficient, but institutions will only really be useful when, as in other safety-critical scenarios, it is possible to prove that particular properties do or do not hold for all possible encounters. In contrast to symbolic model-checking, answer set programming permits the statement of problems and queries in domain-specific terms as executable logic programs, thus eliminating the gap between specification and verification language. Furthermore, results are presented in the same terms. In this paper we describe the use of answer set programs as an institutional modelling technique. We demonstrate that our institutional model can be intuitively be mapped into an answer set program such that the ordered event traces of the former can be obtained as the answer sets of the latter, allowing for an easy way to query properties of models.

1 Introduction

The case for institutions as mechanisms to structure and enable agent interactions has been made at length in numerous places over the last 10 years. Probably the most relevant fact for this paper is the recognition that it is the *institutional norms* [18] that hold the key, where norms are statements that serve to guide or regulate agent behaviour, ranging from the abstract ("treat others as you would wish to be treated yourself") through rules ("if this boat's catch of cod exceeds its annual quota then a fine is payable") to protocols defining sequences of (typically) speech acts.

This is not the place for a repetition of the arguments for institutions, but for the sake of making this paper self-contained, we give a brief introduction to the literature for the interested reader. The earliest presentation in the computer science literature is perhaps Noriega's thesis [35], followed by Rodriguez [37] and Vazquez-Salceda [38]. Alongside, there have been several attempts at finding tractable representations of institutional norms, starting from the original FishMarket paper [36] using automata [22], process algebra [29], symbolic model-checking with temporal logics [11], commitments [39], social institutions [41] action languages [2] and answer set programming [10]. Initial approaches were bottom-up, starting from protocols, but to date creating a verifiable relationship between protocols and higher level representations of norms has not proven fruitful. Thus more recent approaches such as [1, 16, 42] and including our own, have

sought to address this problem by specifying normative behaviour at a level which is both easily expressed by designers and computationally executable and verifiable.

In this paper a top-down approach to virtual institutions is described, in which external normative concepts are represented in terms that at the same time designers may analyse (off-line) and about which agents may reason (on-line) using the answer set programming paradigm. In formalising the ideas set out in [10], this paper makes two further contributions: (i) a formal event-based model of the specification of institutions that captures all the essential properties, namely empowerment, permission, violation and obligation (ii) a verifiable translation to answer set programming, resulting in a decidable and executable model for institutions.

2 Virtual Institutions

To provide some context for the theory that follows, this section begins with a brief overview of institutions and the terms that we use. As outlined in the introduction the essential characteristics of an institution are captured in its norms with varying degrees of specificity. What agents do or say is constrained by a given institutional context, so that irrelevant actions or communications are filtered out, and relevant ones advance the interaction, cause an agent to acquire an obligation, or through a violation, invite a sanction. But while that serves to capture the agent's point of view, what about the (institutional) environment? How are actions to be observed, how are obligations to be recorded and their satisfaction enforced, and how are violations to be detected and the corresponding sanctions to be applied?

The model we propose is based on the concept of *Observable Events* that capture notions of the physical world — "shoot somebody" — and *Institutional Events* that are those generated by society — "murder" — but which only have meaning within a given social context. While observable events are clearly observable, institutional ones are not, so how do they come into being? Searle [28] describes the creation of an institutional state of affairs through *Conventional Generation*, whereby an event in one context *Counts As* the occurrence of another event in a second context. Taking the physical world as the first context and by defining conditions in terms of states, institutional events may be created that count as the presence of states or the occurrence of events in the institutional world.

Thus, we model an institution as a set of *institutional states* that evolve over time subject to the occurrence of *events*, where an institutional state is a set of *institutional fluents* that may be held to be true at some instant. Furthermore, we may separate such fluents into *domain* fluents, that depend on the institution being modelled, such as "A owns something", and *normative fluents* that are common to all specifications and may be classified as follows:

- **Institutional Power:** This represents the institutional capability for an event to be brought about meaningfully, and hence change some fluents in the institutional state. Without institutional power, the event may not be brought about and has no effect; for example, a marriage ceremony will only bring about the married state, if the person performing the ceremony is empowered so to do.

– **Permission:** Each permission fluent captures the property that some event may occur without violation. If an event occurs, and that event is not permitted, then a *violation event* is generated.
– **Obligation:** Obligation fluents are modelled as the dual of permission . An obligation fluent states that a particular event is obliged to occur before a given deadline event (such as a timeout) and is associated with a specified violation. If an obligation fluent holds and the obliged event occurs then the obligation is said to be satisfied. If the corresponding deadline event occurs then the obligation is said to be violated and the specified violation event is generated.

Events can be classified into: (i) a set of observable events, being those events external to the institution which may be brought about independently from the institution and (ii) a set of *institutional events* which may be broken down into *violation events* and *institutional actions*; these events may only be brought about if they are generated by the institutional semantics. Finally we have a set of institutional rules which associate the occurrence of events with some effects in the subsequent state. These can be divided into: (i) *generation rules* which account for the conventional generation of events. Each generation rule associates the satisfaction of some conditions in the current institutional state and the occurrence of an (observed or institutional) event with a generated institutional event. For example: "A wedding ceremony counts as civil marriage only if the couple have a licence". The generating and generated events are taken by the institution to have occurred simultaneously. (ii) *consequence rules*, each of which associates the satisfaction of some conditions in the current institutional state and the occurrence of an event in the institution or the world to the change in state of one or more fluents in the next institution state. For example: "Submitting a paper to a conference grants permission for the paper to be redistributed by the conference organisers".

Violation and sanction play an important role in the specification of institutions. 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 simply be expressed as consequences of the occurrence of the associated violation event in the subsequent institutional state.

2.1 The Institutional Model

From the introduction above, it can be seen that a definition of a institution is a quintuple $\mathcal{I} := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \mathcal{S}_0 \rangle$ consisting of institutional events ($\mathcal{E}$), fluents ($\mathcal{F}$), a consequence relation ($\mathcal{C}$), an event generation relation ($\mathcal{G}$) and an initial state ($\mathcal{S}_0$). We now describe each of these in more detail.

Institutional Events Each institution defines a set of event signatures $\mathcal{E}$, to denote the types of events that may occur. $\mathcal{E}$ comprises two disjoint subsets, $\mathcal{E}_{obs}$ denoting *observable events* and $\mathcal{E}_{inst}$ denoting *institutional events*. We break institutional events down further into the disjoint subsets: *institutional actions*, $\mathcal{E}_{instact}$ and *violation events* $\mathcal{E}_{viol}$. We define $\mathcal{E}_{viol}$ such that $\forall e \in \mathcal{E}_{instact} \cup \mathcal{E}_{obs} \cdot \mathrm{viol}(e) \in \mathcal{E}_{viol}$ (i.e. each institutional action has a corresponding violation event $\mathrm{viol}(e)$ in $\mathcal{E}_{viol}$ which may arise from the performance of e when it is not permitted).

Institutional Fluents Each institution comprises the union of four distinguished sets of fluents: one defines a set of *Domain Fluents* denoted $\mathcal{D}$ that account for the description of the domain the institution operates, while the remainder are sets of boolean fluents indicating different types of *normative fluents*:

$\mathcal{W}$ A set of institutional powers of the form $\mathrm{pow}(e) : e \in \mathcal{E}_{instact}$ where each power fluent denotes the capability of some event e to be brought about (generated) in the institution.

$\mathcal{M}$ A set of event permissions: $\mathrm{perm}(e) : e \in \mathcal{E}_{instact} \cup \mathcal{E}_{obs}$ where each permission fluent denotes that it is permitted for an event e to be brought about. An event is not explicitly forbidden, instead this is implicitly represented by the absence of permission for that event to be brought about.

$\mathcal{O}$ A set of obligations, of the form $\mathrm{obl}(e, d, v) : e \in \mathcal{E}, d \in \mathcal{E}, v \in \mathcal{E}_{inst}$ where each obligation fluent denotes that event e should be brought about before the occurrence of event d or be subject to the violation v.

Together, these disjoint sets of domain fluents and normative fluents form the *Institutional Fluents* $\mathcal{F}$ ($\mathcal{F} = \mathcal{W} \cup \mathcal{M} \cup \mathcal{O} \cup \mathcal{D}$).

The state of an institution at a certain time is determined by those institutional fluents that are valid at that time. The set of all possible *institutional states* is denoted as Σ with $\Sigma = 2^{\mathcal{F}}$. It is important to appreciate that not all those states will actually be used in an institution.

Events can have the same effect on multiple of states, not just one. Borrowing a book from a library will result in the obligation to bring it back regardless of how many books have been borrowed in the past. To facilitate this, we introduce the concept of *State Formula* as a collection of states that satisfy certain properties in that they either contain certain fluents or they do not. The set of all state formulae is denoted as $\mathcal{X}$ with $\mathcal{X} = 2^{\mathcal{F} \cup \neg\mathcal{F}}$, where $\neg\mathcal{F}$ is the negation of each fluent in $\mathcal{F}$.

Consequences Each institution defines a function $\mathcal{C}$ that describes which fluents are initiated and terminated by the occurrence of a certain event in a state matching some criteria. The function is expressed as $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$. Given $X \in \mathcal{X}$ and $e \in events$, $C(X, e) = (\mathcal{C}^{\uparrow}(X, e), \mathcal{C}^{\downarrow}(X, e))$ with $\mathcal{C}^{\uparrow}(X, e)$ containing those fluents which are *initiated* by the event e in any state matching X and $\mathcal{C}^{\downarrow}(X, e)$ collecting those fluents which are *terminated* by event e in any state matching X.

Event Generation Each institution defines an event generation function $\mathcal{G}$ which describes when the occurrence of one event *counts as* the occurrence of other events inside the institution: $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$.

As a consequence there could be a cascading of events. As we will see later, we require the transitive closure to obtain all generated events from one initial observable event.

Initial State Each institution defines the set $\mathcal{S}_0 \subseteq \mathcal{F}$ that denotes the set of fluents that hold when the institution is created.

2.2 Semantics

During the lifetime of an institution, its state changes due to events taking place. Each observable event possibly generates more events which in turn could create further events. Each of these events could affect the current state, while their confluence determines the next state.

States We define the semantics of an institution over a set of states Σ. Each state comprises a set of fluents in $\mathcal{F}$ which are held to be true at a given time (see for example Figure 7). We say that a state $S \in \Sigma$ satisfies fluent $f \in \mathcal{F}$, denoted $S \models f$, when $f \in S$. It satisfies its negation $\neg f$, when $f \notin S$. This notation can be extended to sets $X \subseteq \mathcal{X}$ in the following way: $S \models X$ iff $\forall x \in X \cdot S \models x$.

Event Generation In order to account for event generation we define a function that describes which events to generated in a given state. $\mathrm{GR} : \Sigma \times 2^{\mathcal{E}} \to \mathcal{E}$ ($\mathcal{E}$ is the set of all institutional events). In some state S subject to a set of events E, $\mathrm{GR}(S, E)$ includes all of the events which must be generated by the occurrence of events E in state S and is defined as follows:

$$
\begin{array}{lll}
\mathrm{GR}(S,E) = \{e \in \mathcal{E} \mid & e \in E & \textbf{or} \\
& \exists e' \in E, \phi \in \mathcal{X}, e \in G(\phi, e') \cdot S \models \mathrm{pow}(e) \wedge S \models \phi & \textbf{or} \\
& \exists e' \in E, \phi \in \mathcal{X}, e \in G(\phi, e') \cdot e \in \mathcal{E}_{viol} \wedge S \models \phi & \textbf{or} \\
& \exists e' \in E \cdot e = viol(e'), S \models \neg\,\mathrm{perm}(e') & \textbf{or} \\
& \exists e' \in \mathcal{E}, d \in E \cdot S \models \mathrm{obl}(e', d, e)\} &
\end{array}
$$

1. The first condition ensures that events remain generated (inertia).
2. The second condition defines event generation to be explicitly specified by the institutional relation G. One event generates another event in a given state, when (i) the generation was specified by the institution, (ii) the current state satisfies the conditions for the generation and (iii) the generated event is empowered.
3. The third condition deals with violations generated as specified by the institution rather that violations resulting from events that were not permitted. Violations do not require empowerment.
4. The fourth condition considers the generation of violation events as the result of the occurrence of non-permitted events.
5. The last condition deals with the generation of violation events as the result of the failure to bring about an obliged event. For all asserted obligation fluents, the occurrence of the deadline event d generates the corresponding violation event e.

The parallel generation of events, means it is possible for an event which fulfils an obligation to be generated simultaneously with the obligation's deadline (that is, the deadline counts as the fulfilment of the obligation or the obligation counts as the fulfilment of the deadline or another action counts as both the fulfilment of the deadline and the fulfil lent of the obligation). While we consider this situation undesirable we do not prohibit its specification, but say that when it does occur the obligation is considered as not to have been fulfilled.

It is easy to see that $\mathrm{GR}(S, E)$ is a monotonic function. This implies that for any given state and a set of events, we can obtain a fixpoint $\mathrm{GR}^{\omega}(S, E)$. In our institutional model, generated events come about from the performance of one *observable event* $e_{obs} \in \mathcal{E}_{obs}$ in a given state S. So, to obtain all events that originate from this one event in this state, we simply need $\mathrm{GR}^{\omega}(S, \{e_{obs}\})$.

Event Effects Each fluent in $\mathcal{F}$ may either be asserted or not in each state in S. The status of these fluents changes over time according to which generated events have occurred in the previous transition.

Events can have two sorts of effects: fluents can be initiated (they become true in the next state) or they can be terminated (they cease to be true in the next state). The combination of all effects generated in a state defines the state transition. The state transition function captures inertia, so all fluents that are not affected in the current state remain valid in the next state.

As mentioned above, given an observable event e_{obs} all events that could have an effect on the state S, are obtained by $\mathrm{GR}^{\omega}(S, \{e_{obs}\})$.

The set of all *initiated fluents* $\mathrm{INIT}(S, e_{obs}) \subseteq \mathcal{F}$ for some state $S \in \Sigma$ and an observable event $e_{obs} \in \mathcal{E}_{obs}$ is defined as:

$$\mathrm{INIT}(S, e_{obs}) = \{p \in \mathcal{F} \mid \exists e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), X \in \mathcal{X} \cdot p \in \mathcal{C}^{\uparrow}(X, e) \wedge S \models X\}$$

A fluent will be initiated if an event is generated in the current state for which $\mathcal{C}$ specifies, that in current state, this event has the consequence that the fluent is initiated.

We go on to define which fluents are terminated in a given state by the occurrence of a given event:

$$\begin{aligned}\mathrm{TERM}(S, e_{obs}) = \{p \in \mathcal{F} \mid\ & \exists e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), X \in \mathcal{X} \cdot p \in \mathcal{C}^{\downarrow}(X, e), S \models X \textbf{ or} \\ & p = \mathrm{obl}(e, d, v) \wedge p \in S \wedge e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}) \qquad \textbf{or} \\ & p = \mathrm{obl}(e, d, v) \wedge p \in S \wedge d \in \mathrm{GR}^{\omega}(S, \{e_{obs}\})\}\end{aligned}$$

A fluent is terminated if an event is generated in the current state for which $\mathcal{C}$ specifies that it needs terminating. Furthermore, an obligation fluent is terminated if either its deadline or the obliged event are in the set of generated events.

Now that we know which fluents need adding or deleting we can define the transition function $\mathrm{TR} : \Sigma \times \mathcal{E}_{obs} \to \Sigma$ as:

$$\begin{aligned}\mathrm{TR}(S, e_{obs}) = \{p \in \mathcal{F} \mid\ & p \in S, p \notin \mathrm{TERM}(S, e_{obs}) \textbf{ or} \\ & p \in \mathrm{INIT}(S, e_{obs})\}\end{aligned}$$

The first condition models inertia: all fluents which are asserted in the current state persist into the next state, unless they are terminated. The second condition includes fluents which are initiated in the current state.

Ordered Traces Now that we have defined how states may be generated from a previous state and a single observable event, we are able to define traces and their state evaluations:

$$\mathcal{E}_{obs} = \{\texttt{shoot}, \texttt{startwar}, \texttt{declaretruce}, \texttt{callup}, \texttt{provoke}\} \quad (1)$$

$$\mathcal{E}_{instact} = \{\texttt{conscript}, \texttt{murder}\} \quad (2)$$

$$\mathcal{E}_{viol} = \{\mathrm{viol}(\texttt{shoot}), \mathrm{viol}(\texttt{startwar}), \mathrm{viol}(\texttt{declaretruce}), \mathrm{viol}(\texttt{callup}), \mathrm{viol}(\texttt{provoke}), \mathrm{viol}(\texttt{conscript}), \mathrm{viol}(\texttt{murder})\} \quad (3)$$

$$\mathcal{D} = \{\texttt{atwar}\} \quad (4)$$

$$\mathcal{W} = \{\mathrm{pow}(\texttt{conscript}), \mathrm{pow}(\texttt{murder})\} \quad (5)$$

$$\mathcal{M} = \{\mathrm{perm}(\texttt{shoot}), \mathrm{perm}(\texttt{startwar}), \mathrm{perm}(\texttt{declaretruce}), \mathrm{perm}(\texttt{callup}), \mathrm{perm}(\texttt{provoke}), \mathrm{perm}(\texttt{conscript}), \mathrm{perm}(\texttt{murder})\} \quad (6)$$

$$\mathcal{O} = \{\mathrm{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} \quad (7)$$

$$\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E}): \langle\{\neg atwar\}, \texttt{startwar}\rangle \rightarrow \{atwar\} \quad (8)$$

$$\langle\{\neg atwar\}, \texttt{provoke}\rangle \rightarrow \{\mathrm{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} \quad (9)$$

$$\langle\emptyset, \texttt{conscript}\rangle \rightarrow \{\mathrm{perm}(shoot)\} \quad (10)$$

$$\langle\emptyset, \texttt{startwar}\rangle \rightarrow \{\mathrm{pow}(conscript)\} \quad (11)$$

$$\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E}): \langle\{atwar\}, \texttt{declaretruce}\rangle \rightarrow \{atwar\} \quad (12)$$

$$\langle\emptyset, \texttt{declaretruce}\rangle \rightarrow \{\mathrm{perm}(shoot)\} \quad (13)$$

$$\langle\emptyset, \texttt{declaretruce}\rangle \rightarrow \{\mathrm{pow}(conscript)\} \quad (14)$$

$$\mathcal{G}(\mathcal{X}, \mathcal{E}): \langle\emptyset, \texttt{callup}\rangle \rightarrow \{conscript\} \quad (15)$$

$$\langle\emptyset, \texttt{viol(shoot)}\rangle \rightarrow \{murder\} \quad (16)$$

$$S_0 = \{\mathrm{perm}(callup), \mathrm{perm}(startwar), \mathrm{perm}(conscript), \mathrm{perm}(provoke), \mathrm{pow}(murder), \mathrm{perm}(murder)\} \quad (17)$$

Fig. 1. The War Institution

- An *ordered trace* is defined as a sequence of observable events

$$\langle e_0, e_1, \ldots, e_n\rangle \;\; e_i \in \mathcal{E}_{obs}, 0 \leq i \leq n$$

- The *evaluation of an ordered trace* for a given starting state S_0 is a sequence $\langle S_0, S_1, \ldots S_{n+1}\rangle$ such that $S_{i+1} = \mathrm{TR}(S_i, e_i)$
- Ordered traces and their evaluations allow us to monitor or investigate the evolution of an institution over time. They also provide us with the data necessary to answer most queries one might have about the dynamic evolution of institutional state.

2.3 An example: War

A country is constantly swinging between war and peace with its neighbour. The countries have agreed that when they are at peace, the act of a citizen of the first shooting a citizen of the second counts as murder. But, when they are at war and a citizen has been conscripted into the army it is permitted to shoot. When one country is provoked, it is obliged to start war first before it is allowed to shoot.

The institutional model is depicted in Figure 1. (1) shows that a country can observe a shooting, that either party has started the war or declared a truce, that the citizenry have

been called up and that a country has been provoked, while the institution as a whole can acknowledge that conscription has taken place and somebody has been murdered, as stated by (2). (3) indicates all the violations that could occur. (4) contains one domain fluent stating that the country is at war, while (5–7) indicate the empowerments, permissions and obligation the countries can hold.

The decision to start a war in time of peace results in the institutional state changing to war, as shown in (8). (9) generates the obligation to start a war first before shooting to avoid committing a murder whenever being provoked during a period of peace. (10) provides the permission to shoot whenever conscription has taken place, which is empowered when a war is started, as indicated by (11). Declaring a truce will end the state of war (12) when at war and revoke the permission to shoot (13) and the power to conscript (14). When a country issues the *callup* command, the institution will generate conscription when empowered (15). When a shooting violation occurs, the institution will raise the murder event (16). Initially (17), the institution declares a number of permissions and empowerments.

3 Modelling Institutions using Answer Set Programming

By encoding institutions as declarative specifications it becomes possible to reason computationally about the consequences of "real world" actions such as message exchanges, on social states. This allows agents participating in an institution to take account of events up to given point in time and to execute the specification in order to determine the social state at that time. Similarly agents may reason about the social effects of future actions and act accordingly.

In this section we discuss the use of answer set programming (ASP) [4] to model and reason about institutions, the agents that participate in them and the norms that govern them. ASP is a logic programming language that has the advantage that specification and implementation are identical, the language is easy to understand yet very powerful and expressive, it comes with efficient algorithms, called solvers, to provide the solution to the encoded problem and the availability of different types of negation: classical negation and negation-as-failure[1], the latter giving rise to non-deterministic outcomes.

3.1 Answer Set Programming

In *answer set programming* ([4, 24]) a logic program is used to describe the requirements that must be fulfilled by the solutions of a certain problem. The answer sets of the program, usually defined through (a variant/extension of) the stable model semantics [24], then correspond to the solutions of the problem. This technique has been successfully applied in domains such as planning [20, 33], configuration and verification [40], super-optimisation [6], diagnosis [19], game theory [14] and multi-agent systems[5, 8, 15, 7, 10] where [7, 10] use answer set programming to reason about the behaviour of a group of agents, while [5, 8, 15] use the formalism to model the reasoning capabilities, knowledge and beliefs of a single agent within a multi-agent system.

[1] For classical negation one expects a proof that something is indeed false, while for negation-as-failure it is sufficient that no proof exists that something is true.

$$
\begin{aligned}
\texttt{guilty} &\leftarrow \texttt{evidence}.\\
\texttt{evidence} &\leftarrow \texttt{trusted_witness}.\\
\texttt{trusted_witness} &\leftarrow \mathbf{not}\ \texttt{lying}, \texttt{witness}.\\
\texttt{witness}.&\\
\texttt{believe} &\leftarrow \mathbf{not}\ \texttt{disbelieve}.\\
\texttt{disbelieve} &\leftarrow \mathbf{not}\ \texttt{believe}.\\
\texttt{lying} &\leftarrow \texttt{disbelieve}.
\end{aligned}
$$

Fig. 2. Program for jury example

The smallest building block of an answer set program is an atom or predicate, e.g $\texttt{owns}(\texttt{X}, \texttt{Y})$ stating that X owns Y. X and Y are variables which can be grounded with constants, e.g. $\texttt{owns}(\texttt{me}, \texttt{book})$. Each ground atom can be assigned the truth value *true* or *false*. Answer set programs uses two types of negation: $\neg$ and $\mathbf{not}$. The former is classical negation, indicating that something is know to be false because a proof exists. The latter denotes negation as failure, stating that something should be assumed false due to the failure of proving it to be true. A literal is an atom a or its negation $\neg a$. An *extended* literal is either a literal l or its negation $\mathbf{not}\ l$.

An answer set programs consist of a set of statements, called rules. Each rule $l \leftarrow B$ is made of two parts namely the body B, which is a set of extended literals, and a head literal l. It should be read as: "if all the elements of B are true, so is the head l" or "l" is *supported* if all elements of B are considered to be true. An assignment of truth values to all literals in the program, without causing contradiction, is called an interpretation. Often only those literals that are considered true are mentioned, as all the others are false by default (negation as failure).

Obviously, we only assume those literals to be true that are actually supported. This form of reasoning is referred to as the minimal model semantics. Unfortunately, in the presence of negation-as-failure this approach is insufficient. Negation-as-failure gives us no guarantee that something is indeed false and that information derived from it is actually correct. To obtain intuitive solutions, we need to verify this. This is done by reducing the program to a simpler program containing no instances of negation-as-failure. Given an interpretation, all rules that contain $\mathbf{not}\ l$ that are considered false are removed while the remaining rules only retain their literals. This reduction is often referred to as the Gelfond-Lifschitz transformation. When this program gives the same supported literals as the ones with which we began, we have found an answer set.

Definition 1. *Let P be a ground program.*
The Gelfond-Lifschitz transformation of P w.r.t S, a set of ground literals, is the program P^S containing the rules $l \leftarrow B$ such that $l \leftarrow B, \mathbf{not}\ C \in P$ with $C \cap S = \emptyset$, with B and C sets of literals.
A set of ground literals S is an answer set of P iff S is the minimal model of P^S.

The uncertain nature of negation-as-failure gives rise to several answer sets, which are all acceptable solutions to the problem that has been modelled. It is in this non-determinism that the strength of answer set programming lies.

Example 1. Consider the following situation. A jury member has to decide if the accused is guilty or not based on evidence provided by a witness. The only problem for the jury member is to decide whether they trust this witness or not. This situation can be represented by the following program shown in Figure 2, which has two answer sets:
- $\{\texttt{guilty}, \texttt{evidence}, \texttt{trusted_witness}, \texttt{witness}, \texttt{believe}\}$
- $\{\texttt{witness}, \texttt{lying}, \texttt{disbelieve}\}$

These two answer sets indicate clearly that the jury member has to decide on the credibility of the witness and her decision is vital for her judgement of the accused.

Algorithms and implementations for obtaining answer sets of logic programs are referred to as *answer set solvers*. The most popular and widely used solvers are DLV[21] and SMODELS[34]. An alternative is CMODELS[26], a solver based on translating the program to a SAT problem.

Each solver has two phases. First the program is grounded, that is the variables are substituted for constants. Within this phase, rules which are obviously leading to nothing are eliminated. Take for example the program:

$\texttt{ifluent}(\texttt{atwar}).$
$\texttt{event}(\texttt{shoot}).$
$\texttt{holdsat}(\texttt{P}, 2) \leftarrow \texttt{holdsat}(\texttt{P}, 1), \textbf{not}\ \texttt{terminated}(\texttt{P}, 1), \texttt{ifluent}(\texttt{P}).$
this last rule has two grounded instances:
$\texttt{holdsat}(\texttt{atwar}, 2) \leftarrow \texttt{holdsat}(\texttt{atwar}, 1), \textbf{not}\ \texttt{terminated}(\texttt{atwar}, 1), \texttt{ifluent}(\texttt{atwar})$
$\texttt{holdsat}(\texttt{shoot}, 2) \leftarrow \texttt{holdsat}(\texttt{shoot}, 1), \textbf{not}\ \texttt{terminated}(\texttt{shoot}, 1), \texttt{ifluent}(\texttt{shoot}).$

The parser will eliminate the second ground instance as no rules are provided to derive `ifluent(shoot)`. The second phase is the actual solver where a grounded program is taken and the set of its answer sets is produced.

For this paper we have opted to use SMODELS as our solver and hence we use the SMODELS syntax in the examples that follow. This will also allow us to use the distributed PLATYUS solver[27], which uses SMODELS as a back-end, for larger implementations of institutions.

3.2 Translation into Answer Set Programs

In order to reason about traces over a given institution, we define the following translation from the institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \mathcal{S}_0 \rangle$ into an answer set program. We use instances of time to indicate the state transitions of an institution.

The mapping uses the following atoms: `ifluent(P)` to identify fluents, `evtype(E,T)` to describe the type of an event, `event(E)` to denote the events, `instant(I)` for time instances, `final(I)` for the last time instance in a trace, `next(I1,I2)` to establish time ordering, `occurred(E,I)` to indicate that the event happened at time I, `observed(E,I)` that the event was observed at that time, `holdsat(P,I)` to state that the institutional fluent holds at I, `initiated(P,I)` and `terminated(P,I)` for fluents that are initiated and terminated at I.

Since we are using SMODELS, we can take advantage of some of its syntactic constructs. In our mapping we use their choices syntax, symbolic functions and the built-in compute statement:

$$
\begin{aligned}
\texttt{occurred(E,I)} &\leftarrow \texttt{observed(E,I)}. && (18)\\
\texttt{holdsat(P,I2)} &\leftarrow \texttt{holdsat(P,I1)}, \textbf{not}\ \texttt{terminated(P,I1)}, &&\\
&\quad \texttt{next(I1,I2)}, \texttt{instant(I1;I2)}, &&\\
&\quad \texttt{ifluent(P)}. && (19)\\
\texttt{holdsat(P,I2)} &\leftarrow \texttt{initiated(P,I1)}, \texttt{ifluent(P)}, &&\\
&\quad \texttt{next(I1,I2)}, \texttt{instant(I1;I2)}, && (20)\\
\texttt{occurred(viol(E),I)} &\leftarrow \texttt{occurred(E,I)}, &&\\
&\quad \textbf{not}\ \texttt{holdsat(perm(E),I)}, &&\\
&\quad \texttt{event(E)}, \texttt{event(viol(E))}, \texttt{instant(I)}. && (21)\\
\texttt{occurred(V,I)} &\leftarrow \texttt{holdsat(obl(E,D,V),I)}, \texttt{occurred(D,I)}, &&\\
&\quad \texttt{event(E;D;V)}, \texttt{instant(I)}. && (22)\\
\texttt{terminated(obl(E,D,V),I)} &\leftarrow \texttt{occurred(E,I)}, &&\\
&\quad \texttt{holdsat(obl(E,D,V),I)}, &&\\
&\quad \texttt{event(E;D;V)}, \texttt{instant(I)}. && (23)\\
\texttt{terminated(obl(E,D,V),I)} &\leftarrow \texttt{occurred(D,I)}, &&\\
&\quad \texttt{holdsat(obl(E,D,V),I)}, &&\\
&\quad \texttt{event(E;D;V)}, \texttt{instant(I)}. && (24)
\end{aligned}
$$

Fig. 3. The institution base program

- Choices written $L\{l_1, \dots l_n\}M$ are a convenient construct to express that any number of literals between L and M from the set $\{l_1\ l_n\}$ need to be true in order to satisfy the construct. When omitted L is considered 0 and M to be n.
- A symbolic function $\texttt{f(X,Y)}$ defines a new constant that is the value of the function. It is used as a shorthand to group sets of variables together in a meaningful way. We use this represent obligations $\texttt{obl(E,D,V)}$ and violations $\texttt{viol(R)}$.
- The compute statement is used to generate only those answer sets that satisfy certain properties. The statement $\texttt{computenumber}\{\texttt{l}_1, \dots \texttt{l}_n\}$. makes sure that only answer sets that satisfy every extended literal l_i for $1 \leq i \leq n$ are computed. The number of generated answers is controlled by `number`.
- We also use facility for passing multiple argument lists to literals: when used in the body of a rule $\texttt{a}(\texttt{args}_1; \dots; \texttt{args}_n)$ is replaced by $\{\texttt{a}(\texttt{args}_1), \dots, \texttt{a}(\texttt{args}_n)\}$.

Each mapping of each institution $\mathcal{I}$ consists of two parts: P_{base} which is identical for each interpretation and $P^*_{\mathcal{I}}$ specific for the institution being modelled. Together they form the program $P_{\mathcal{I}}$.

The base program P_{base} (Figure 3) consists of rules responsible for the occurrence of observed events and dealing with obligations and inertia. The first rule (18) assures that each observed event ($\texttt{observed(E,I)}$) will be marked as occurred, as all observable events are valid events. Rules (19) encode standard inertia, using negation as failure: any fluent which is currently valid ($\texttt{holdsat}(\texttt{I}_1)$) and will not be terminated in this state ($\textbf{not}\ \texttt{terminated}(\texttt{P},\texttt{I}_1)$) needs still to be valid in the next state ($\texttt{holdsat}(\texttt{P},\texttt{I}_2)$). The

$$\{\mathtt{observed(E,I)}\} \leftarrow \mathtt{evtype(E,obs)}, \mathtt{event(E)}, \mathtt{instant(I)}, \mathbf{not}\ \mathtt{final(I)}. \quad (25)$$

$$\mathtt{ev(I)} \leftarrow \mathtt{observed(E,I)}, \mathtt{event(E)}, \mathtt{instant(I)}. \quad (26)$$

$$\leftarrow \mathbf{not}\ \mathtt{ev(I)}, \mathtt{instant(I)}, \mathbf{not}\ \mathtt{final(I)}. \quad (27)$$

$$\leftarrow \mathtt{observed(E1,I)}, \mathtt{observed(E2,I)}, \mathtt{E1!} = \mathtt{E2}, \mathtt{instant(I)}, \mathtt{event(E1)}, \mathtt{event(E2)}. \quad (28)$$

Fig. 4. Rules for ensuring observable traces

atoms $\mathtt{next(I_1, I_2)}$ and $\mathtt{instant(I_1; I_2)}$ are responsible for obtaining the next time instance and for restricting the grounding domain. The rule (20) ensures that fluents that are initiated ($\mathtt{initiated(P, I_1)}$) become valid ($\mathtt{holdsat(P, I_2)}$) in the next state. Rule (21) is responsible for the generation of violations that are caused by non-permitted events. Whenever an event occurs ($\mathtt{occurred(E, I)}$) for which no permission exists in that state (**not** $\mathtt{holdsat(perm(E), I)}$) a violation is raised ($\mathtt{occurred(viol(E), I)}$). The last three rules deal with obligations. (22) is responsible for raising a violation ($\mathtt{occurred(V, I)}$) whenever the deadline expires ($\mathtt{occurred(D, I)}$). The other atoms in the body of this rule guarantee appropriate grounding of this rule. The rules (23) and (24) regulate the end of obligations ($\mathtt{terminated(obl(E, D, V), I)}$) when either the obligation is fulfilled ($\mathtt{occurred(E, I)}$) or the deadline expires ($\mathtt{occurred(D, I)}$).

To constrain the answer set to those containing observable traces we add the rules in Figure 4 to P_{base}. Rule (25) is responsible for the generation of $\mathtt{observed(E, I)}$ atoms. For each combination of an event ($\mathtt{event(E)}$) which is observable ($\mathtt{evtype(E, obs)}$) and non-final (**not** $\mathtt{final(I)}$) at time instance ($\mathtt{instance(i)}$) an $\{\mathtt{observed(E, I)}\}$-choice is created, indicating that you can either use this $\mathtt{observed(E, I)}$ atom or not. (26) creates for each choice of $\mathtt{observed(E, I)}$ atom an $\mathtt{ev(I)}$ atom, which will be used by (27) to restrict the answer sets to observable traces, that is an observable event occurs at each time instance. The last constraint (28) assures that each answer set has only one observable event at every time instance.

To make the program $P^*_{\mathcal{I}}$ more readable we introduce the shorthand $EX(_, I)$ to denote the translation of expression $X \in \mathcal{X}$ into the body of an ASP rule referring to time I. $EX(x_1 \wedge x_2 \wedge \ldots x_n, I)$, with $x_i \in X$, is translated into an ASP conjunction $EX(x_1, I), EX(x_2, I), \ldots, EX(x_n, I)$. $EX(\neg p, I)$ is translated using negation as failure into **not** $EX(p, I)$. $EX(p, I)$ is translated into $\mathtt{holdsat(p, I)}$.

With these syntactic rules $P^*_{\mathcal{I}}$ becomes the program shown in Figure 5. By (29), all the fluents are encoded as facts $\mathtt{ifluent(p)}$ in the program. The main purpose of these facts is to facilitate grounding. Each event e in the institution is responsible for the creation of two facts: (30) generates $\mathtt{event(e)}$. facts while (31–33) record the types of events with facts of the form $\mathtt{evtype(e, X)}$ with $\mathtt{X}$ equal to $\mathtt{obs}, \mathtt{act}, \mathtt{viol}$ to indicate observable, institutional actions and violations. (34) and (35) produce the rules for consequence generation. Whenever a fluent needs to be initiated/terminated a rule will be created with the occurrence of the responsible event ($\mathtt{occurred(e, I)}$) and the conditions on the state ($\mathtt{EX(X, I)}$ in the body and the initiation/termination atom in the head ($\mathtt{initiated(p, I)}$/$\mathtt{terminated(p, I)}$). Event generation is dealt with by (36).

$$p \in \mathcal{F} \Leftrightarrow \texttt{ifluent(p)}. \tag{29}$$

$$e \in \mathcal{E} \Leftrightarrow \texttt{event(e)}. \tag{30}$$

$$e \in \mathcal{E}_{obs} \Leftrightarrow \texttt{evtype(e, obs)}. \tag{31}$$

$$e \in \mathcal{E}_{instact} \Leftrightarrow \texttt{evtype(e, act)}. \tag{32}$$

$$e \in \mathcal{E}_{viol} \Leftrightarrow \texttt{evtype(e, viol)}. \tag{33}$$

$$\mathcal{C}^{\uparrow}(X, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{initiated(p, I)} \leftarrow \texttt{occurred(e, I)}, EX(X, I). \tag{34}$$

$$\mathcal{C}^{\downarrow}(X, e) = P \Leftrightarrow \forall p \in P \cdot \texttt{terminated(p, I)} \leftarrow \texttt{occurred(e, I)}, EX(X, I). \tag{35}$$

$$\mathcal{G}(X, e) = E \Leftrightarrow g \in E, \texttt{occurred(g, I)} \leftarrow \texttt{occurred(e, I)}, \texttt{holdsat(pow(e), I)}, EX(X, I). \tag{36}$$

$$p \in S_0 \Leftrightarrow \texttt{holdsat}(\texttt{p}, \texttt{i}_0). \tag{37}$$

Fig. 5. Rules for translation into SMODELS

For each event that could be generated a rule is produced containing the occurrence of the triggering event ($\texttt{occurred(e, I)}$), the permission to execute this triggering event ($\texttt{holdsat(pow(e), I)}$) and the conditions for the generation in the body and the occurrence of the generated event in the head ($\texttt{occurred(g, I)}$). Finally, the encoding of the initial state is taken care of by (37), each fluent p in the initial state is transformed into a fact $\texttt{holdsat}(\texttt{p}, \texttt{i}_0)$.

Note that P_i^* is only ungrounded with respect to the time instances. The constants for these are provided by a third program P^n. It is this program that determines the length of the traces. This modularisation into three programs allows for easy reuse.

$$0 < k < n : \texttt{instant}(\texttt{i}_\texttt{k}). \tag{38}$$

$$0 < k < n - 1 : \texttt{next}(\texttt{i}_\texttt{k}, \texttt{i}_{\texttt{k+1}}). \tag{39}$$

$$\texttt{final}(\texttt{i}_\texttt{n}). \tag{40}$$

The facts produced by (38) provide the program with all available time instances, while the facts from (39) give order time necessary to go from one state to the other. Since we cannot have an observable event occurring at the final time instance, we need a fact indicating the final state. This fact is produced by (40).

Together P_{base}, $P_{\mathcal{I}}^*$ and P^n generate $P_{\mathcal{I}}^n$, an answer set program capable of providing all ordered traces of length n for the institution $\mathcal{I}$.

Theorem 1. *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \mathcal{S}_0 \rangle$ be an institution with $P_{\mathcal{I}}^n$ its corresponding answer set program. Then, a one-to-one mapping exists between the ordered traces of length n and the answer sets of $P_{\mathcal{I}}^n$.*

Given such a mapping we can add the necessary rules that allow us to produce those traces that fulfil certain requirements. We will demonstrate this in the next section by means of our war institution.

```
ifluent(atwar).

ifluent(obl(startwar, shoot, murder)).

event(shoot).
event(startwar).
event(declaretruce).
event(callup).
event(conscript).
event(murder).
event(provoke).
event(viol(shoot)).
event(viol(startwar)).
event(viol(declaretruce)).
event(viol(callup)).
event(viol(conscript)).
event(viol(provoke)).

evtype(shoot, obs).
evtype(startwar, obs).
evtype(declaretruce, obs).
evtype(callup, obs).
evtype(conscript, inst).
evtype(murder, inst).
evtype(provoke, obs).
evtype(viol(shoot), viol).
evtype(viol(startwar), viol).
evtype(viol(declaretruce), viol).
evtype(viol(callup), viol).
evtype(viol(conscript), viol).
evtype(viol(murder), viol).
evtype(viol(provoke), viol).

initiated(obl(startwar, shoot, murder), I) ← occurred(provoke, I), instant(I),
                                             not holdsat(atwar, I).
                      initiated(atwar, I) ← occurred(startwar, I), instant(I),
                                             not holdsat(atwar, I).
                initiated(perm(shoot), I) ← occurred(conscript, I), instant(I).
            initiated(pow(conscript), I) ← occurred(startwar, I), instant(I).

        terminated(atwar, I) ← occurred(declaretruce, I), instant(I)
                               , holdsat(atwar, I).
  terminated(perm(shoot), I) ← occurred(declaretruce, I), instant(I).
terminated(pow(conscript), I) ← occurred(declaretruce, I), instant(I).

occurred(conscript, I) ← occurred(callup, I), instant(I),
                         holdsat(pow(conscript), I).
   occurred(murder, I) ← occurred(viol(shoot), I), instant(I).

instant(i0; i1; i2; i3).
next(i0, i1).
next(i1, i2).
next(i2, i3).
final(i3).

holdsat(perm(callup), i0).
holdsat(perm(startwar), i0).
holdsat(perm(conscript), i0).
holdsat(perm(declaretruce), i0).
holdsat(perm(murder), i0).
holdsat(perm(provoke), i0).
holdsat(pow(murder)), i0.
```

Fig. 6. War in ASP

3.3 An Example: War in ASP

When we translate the War institution $\mathcal{I}$ from §2.3 for traces of length 3, we obtain for $P^*_{\mathcal{I}} \cup P^3$ the program shown in Figure 6. From left to right and top to bottom, the first two boxes encode the two non auto-generated facts produced by (29). For clarity, we omit the encodings of permissions and power for each institutional event. The two following boxes show the encodings of the events and the event types, as prescribed by

(30–33). The initiating consequence generation rules of (34) are in box five, while box six has the terminating consequence rules of (35). The event generation rules (36) are in the next box. The program P^3 is in box eight and box nine has the initial state (37).

Once we have this basic program $P_{\mathcal{I}}$ we can start to query for specific results, like "Is it possible to have a wartime murder?", "Will provocation always lead to shooting?". In order to do this, two rules have to be added to the program: one to represent the query and one to indicate to the solver that we are only interested in those ordered traces that satisfy the condition. The following ASP rules encode the query "Will the country ever have the obligation to start the war before shooting?":

```
condition←holdsat(obl(startwar, shoot, murder), I), instant(I).
compute all {condition}.
```

The Figures 7 and 8 provide a graphical representation of two of the answer sets from running the program with this query. The former demonstrates that the obligation can be satisfied while the latter shows that there exists at least one trace in which the obligation is broken. The circles represent the time instances. The observable events are given in bold above the arrows linking the time instances together with the result of event generation. Below the circles, we list all the institutional fluents that hold in the current state with the new fluents in bold.

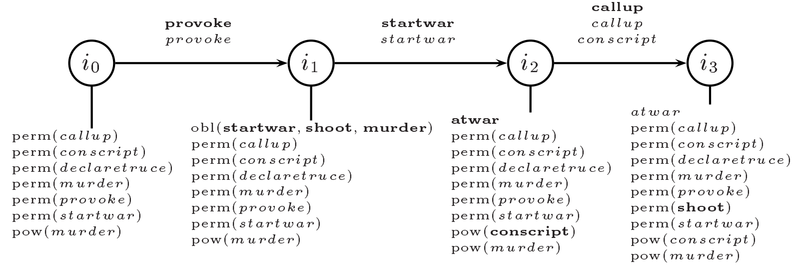
4 Related Work

Much recent and contemporary work on modelling norms and violations has chosen temporal logics as a starting point, as we now discuss.

Colombetti et al in [12] outline an abstract model for agent institutions based on social commitments, where institutions comprise a set of *registration rules* that capture agents' entry into and exit from institutions, a set of *interaction rules* that govern commitment creation and satisfaction, a set of *authorisations* that describe agents' capabilities and an *internal ontology* that describes a model for the interpretation of terms relevant to the institution. Their approach (outlined in [23, 13, 41]) builds on the CTL± extension of CTL[9], which includes past tense modalities for reasoning about actions which have already occurred. Dignum in [17] also uses an extension of CTL to describe her language for representing contracts in the building of agent organisations.

The Event Calculus (EC) [31, 32] is a declarative logic that reinterprets the Situation Calculus to capture when and how states change in response to external events. EC has been used to model both the behaviour of commitments [42] among agents in order to build interaction protocols, corresponding to the regulatory aspects of the work described above, as well as more general social models such as those described in [30]. From a technical point of view, our approach essentially has a kind of duality compared to EC, in that the basis for the model is events rather than states. In itself, this offers no technical advantage although we believe that being able to express violations in terms of events rather than states better captures their nature. More significant are the consequences of the grounding in ASP:

- For the most part the state and event models are equivalent with respect to properties such as induction and abduction, but non-monotonicity is inherent in ASP and so resort to the tricky process of circumscription is avoided.



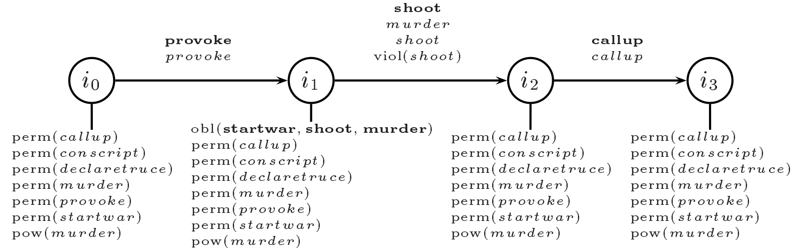
i_0**:** The initial state, wherein all the initial fluents are initiated. The institution observes that a country is provoked. From the event generation function (15 and 16) we know that no further events are generated. The consequence relation (9) is responsible for initiating the obligation $\texttt{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})$ in the next state.

i_1**:** As of this state the obligation $\texttt{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})$ holds. The institution observes startwar event. This event does not generate any further events. Since the obligation has been fulfilled it can be terminated in the next state. The consequence relation (8 and 11) indicate that atwar and $\texttt{pow}(\texttt{conscript})$ have to be initiated in the next state.

i_2**:** In this state the obligation no longer holds and atwar and $\texttt{pow}(\texttt{conscript})$ have been initiated. The institution now observes the callup event. The event generation function (15) thus generates the conscription event, since conscription is now empowered. This results in the consequence relation (10) to order $\texttt{perm}(\texttt{shoot})$ to be initiated in the next state.

i_3**:** This leads us to the final state in which the institution has the permission to shoot.

Fig. 7. Answer set illustrating the obligation satisfied



i_0**:** As for i_0 of Figure 7

i_1**:** As of this state the obligation $\texttt{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})$ holds. The institution observes the event shoot. The events indicates that the deadline of the obligation has passed, so event generation will produce the corresponding violation, in this case murder. Furthermore, since the event shoot was not permitted, the violation $\texttt{viol}(\texttt{shoot})$ is generated, which in turn is responible for the event murder by (17). Since the obligation is violated, it will be terminated in the next state. None of the events cause any state change.

i_2**:** After the violation of the obligation, the institution is returned to its original state. In this state the institution observes the callup event. Because the institution is not empowered to conscript, no other events are generated and no state changes are considered.

i_3**:** The institution has not changed.

Fig. 8. Answer set illustrating the obligation violated

- Likewise, reasoning about defaults requires no special treatment in ASP.
- The consequence rules of our specification have equivalents in EC, but the event generation rules do not.
- The state of a fluent is determined by its truth-value in the ASP interpretation, whereas EC (typically) has to encode this explicitly using two predicates.
- Inertia in EC is axiomatic, whereas in our approach it follows from the application of the TR operator—although there is a strong syntactic similarity (perhaps compounded by using the same terminology!) the philosophy is different.
- ASP allows a wider variety of queries than is typically provided in EC implementations but space constraints do not allow the full illustration of this aspect here.

Artikis et al. in [1, 2, 3, 30] describe a system for the specification of normative social systems in terms of power, empowerment and obligation. This is formalized using both the event calculus [31] and a subset of the action language $C+$ [25]. The notions of power and empowerment are equivalent in both systems, but additionally we introduces violation as events and our modelling of obligations differs in that (i) they are deadline-sensitive, and (ii) can raise a violation if they are not met in time. Violations greatly improve the capacity to model institutions, but it should be remembered that institutional modelling was (apparently) not Artikis's goal. Likewise, although the interpretation of $C+$ using the CCalc tool gives rise to similar reasoning capabilities (with similar complexity) to ASP, we believe our approach, including violations, provides a more intuitive and natural way of expressing social constraints involving temporal aspects. A further advantage is in the formulation of queries, where ASP makes it possible to encode queries similar to those found in (bounded) temporal logic model checking, whereas, as noted above, queries on action languages are constrained by the action language implementation. The other notable difference is once again, our focus on events rather than states, which we have discussed at some length above.

In [7], Buccafurri et al. address the problem of specifying normative properties through the use of *Social Logic Programs* which discriminate between states considered to be acceptable or unacceptable by particular agents. For a given society and situation these social logic programs can be combined and solved under the stable models semantics to give the set of states which are considered to be socially acceptable by group as a whole. In our work we intentionally view the internal models of agents' attitudes as unknown (and hence that all actions which *might* be chosen by are included in possible models of our programs). From the perspective of our work, in the case where it is known that for instance some actions will never be performed by some agents because those actions are considered unacceptable by the agent performing them, it would be desirable to remove these actions from the set of possible models for a given institution. Resolving this automatically represents an interesting area for future research.

5 Conclusions and Directions for Future Research

We have described a formal specification for institutions for the purpose of modelling obligations, permissions and violations, while interactions between agents create traces that record their actions. We demonstrate how the specification may be translated into ASP and subsequently executed producing an answer set. Through the careful specifica-

tion of the institutional state manipulation operations, this answer set has a one-to-one relationship with the institutional event traces of the formal model. In consequence, we arrive at an executable institutional specification that agents may dynamically compute and query to establish both how the current institutional state was reached and which actions will have what consequences in the future of the current state. Tools are currently being prototyped to automate these processes and aid in their visualization.

The ability to reason about and query time-related information is a strong point for using ASP. In our current model of time is discrete, yet we would also like to reason about durations, for examples when dealing with obligations. The DLV[21] system already provides a limited set of aggregates, which would appear to offer a solution and we will experiment with them in the near future.

The current approach does not deal with the effectiveness of sanctions since we do not encode the agent's utility. One solution to this problem would be to encode it as an atom $utility(Agent, X, T)$ and to use an extension of the ASP language we currently use that allows preference. In such a language one would be able to express that $utility(Agent, 10, T)$ is more preferred than $utility(Agent, 5, T)$ for any given Agent at any time.

References

[1] A. Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, Sept. 2003.

[2] A. Artikis, M. Sergot, and J. Pitt. An executable specification of an argumentation protocol. In *Proceedings of conference on artificial intelligence and law (icail)*, pages 1–11. ACM Press, 2003.

[3] A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the Causal Calculator. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Proceedings of Workshop on Agent-Oriented Software Engineering III (AOSE)*, LNCS 2585. Springer, 2003.

[4] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[5] C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[6] M. Brain, T. Crick, M. De Vos, and J. Fitch. Toast: Applying answer set programming to superoptimisation. In *International Conference on Logic Programming*, LNCS. Springer, Aug. 2006.

[7] F. Buccafurri and G. Caminiti. A social semantics for multi-agent systems. In C. Baral, G. Greco, N. Leone, and G. Terracina, editors, *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 317–329. Springer, 2005.

[8] F. Buccafurri and G. Gottlob. Multiagent compromises, joint fixpoints, and stable models. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *Lecture Notes in Computer Science*, pages 561–585. Springer, 2002.

[9] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1981.

[10] O. Cliffe, M. De Vos, and J. Padget. Specifying and analysing agent-based social institutions using answer set programming. In O. Boissier, J. Padget, V. Dignum, G. Lindemann,

E. Matson, S. Ossowski, J. Sichman, and J. Vazquez-Salceda, editors, *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*, pages 99–113. Springer Verlag, 2006. ISBN: 3-540-35173-6.

[11] O. Cliffe and J. Padget. Towards a framework for checking agent interraction within institutions. In *Model Checking and Artificial Intelligence Workshop (MoChArt 02)*, Lyon, France, 2002.

[12] M. Colombetti, N. Fornara, and M. Verdicchio. The role of institutions in multiagent systems. In *Proceedings of the Workshop on Knowledge based and reasoning agents, VIII Convegno AI*IA 2002, Siena, Italy*, 2002.

[13] M. Colombetti and M. Verdicchio. An analysis of agent speech acts as institutional actions. In *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '02)*, pages 1157–1164, New York, NY, USA, 2002. ACM Press.

[14] M. De Vos and D. Vermeir. Choice Logic Programs and Nash Equilibria in Strategic Games. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 266–276, Madrid, Spain, 1999. Springer Verslag.

[15] M. De Vos and D. Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, Sept. 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[16] V. Dignum. *A Model for Organizational Interaction Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.

[17] V. Dignum, J.-J. Meyer, F. Dignum, and H. Weigand. Formal Specification of Interaction in Agent Societies. In *Formal Approaches to Agent-Based Systems (FAABS-02)*, volume 2699 of *Lecture Notes in Computer Science*, pages 37–52, Oct. 2003.

[18] Douglass C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.

[19] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[20] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. The DLV^k planning system. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *European Conference, JELIA 2002*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 541–544, Cosenza, Italy, September 2002. Springer Verlag.

[21] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.

[22] M. Esteva, J. Padget, and C. Sierra. Formalizing a language for institutions and norms. In M. Tambe and J.-J. Meyer, editors, *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2001.

[23] N. Fornara and M. Colombetti. Operational specification of a commitment-based agent communication language. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 536–542, New York, NY, USA, 2002. ACM Press.

[24] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Seattle, Washington, August 1988. The MIT Press.

[25] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *Artificial Intelligence, Vol. 153, pp. 49-104*, 2004.

[26] E. Giunchiglia, Y. Lierler, and M. Maratea. SAT-Based Answer Set Programming. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-04)*, pages 61–66, 2004.

[27] J. Gressmann, T. Janhunen, R. Mercer, T. Schaub, S. Thiele, and R. Tichy. Platypus: A Platform for Distributed Answer Set Solving. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, pages 227–239, 2005.

[28] John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.

[29] Julian Padget. Modelling simple market structures in process algebras with locations. *Artificial Intelligence and Simulation of Behaviour Journal*, 1(1):87–108, 2001. ISSN 1476-3036.

[30] L. Kamara, A. Artikis, B. Neville, and J. Pitt. Simulating computational societies. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of workshop on engineering societies in the agents world (esaw)*, LNCS 2577, pages 53–67. Springer, 2003.

[31] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[32] R. A. Kowalski and F. Sadri. Reconciling the event calculus with the situation calculus. *Journal of Logic Programming*, 31(1–3):39–58, Apr.–June 1997.

[33] V. Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.

[34] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.

[35] P. Noriega. *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona, 1997.

[36] J.-A. Rodríguez, P. Noriega, C. Sierra, and J. Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, Apr. 1997. ISBN 0-9525554-6-8.

[37] J. A. Rodriguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001.

[38] J. V. Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains*. PhD thesis, Technical University of Catalonia, 2003.

[39] M. P. Singh. A social semantics for agent communication languages. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.

[40] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL '99)*, LNCS, San Antonio, Texas, 1999. Springer.

[41] M. Verdicchio and M. Colombetti. A logical model of social commitment for agent communication. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 528–535, New York, NY, USA, 2003. ACM Press.

[42] P. Yolum and M. P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 527–534. ACM Press, 2002.