Department of
Computer Science

UNIVERSITY OF BATH

Technical Report

Specifying and Analysing Agent-based Social Institutions using Answer Set Programming

Owen Cliffe, Marina De Vos, Julian Padget

Technical Report 2005-04 June 2005

ISSN 1740-9497

Copyright ©June 2005 by the authors.

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Specifying and Analysing Agent-based Social Institutions using Answer Set Programming

Owen Cliffe*, Marina De Vos, and Julian Padget

Department of Computer Science
University of Bath
Bath, United Kingdom
{occ,mdv,jap}@cs.bath.ac.uk

Abstract. In this paper we discuss the use of the Answer Set Programming paradigm for representing and analysing specifications of agent-based institutions. We outline the features of institutions we model, and describe how they are translated into ASP programs which can then be used to verify properties of the specifications. We demonstrate the effectiveness of this approach through the institutions of property and exchange.

1 Introduction

Most human interactions are governed by conventions or rules of some sort, having their origin in society (emergent) or the laws (codification of emergent rules) that society has developed. Thus we find that all human societies, even the least developed ones, have some kind of social constraints upon their members in order to structure their relations and simplify their interactions. Some of these constraints are quite informal (taboos, customs, traditions) while some others are formally defined (written laws, constitutions).

The economist and Nobel laureate Douglas North has analysed the effect of this corpora of constraints, that he refers as to *institutions*, on the behaviour of human organisations (including human societies). North states in [10] that institutional constraints ease human interaction (reducing the cost of this interaction), shaping choices and making outcomes foreseeable. By the creation of these constraints, either the organisations and the interactions they require can grow in complexity while interaction costs can even be reduced. Having established these institutional constraints, every competent participant in the institution will be able to act—and expect others to act—according to a list of rights, duties, and protocols of interaction.

Within the field of multi-agent systems there is a view, which we share, that the social consequences of real-world (communicative) interactions among agents may be captured through an explicit *social semantics* [24, 17], these social semantics give an objective description of how an agents' actions in a society may necessarily lead to to the creation of *social states* which in turn may effect the consequences of agents' future interactions on other social states. We take the view that a particular social institution can be represented through the description of the types of social state which may created by agents participating in that institution, and the social rules which cause those states to be created.

Institutions can be applied to the description of a large class of social systems, operating at varying levels of abstraction, from highly abstract notions such as that of property to more concrete ones such as exchange scenarios and protocols. Additionally institutions may be related to one another in a variety of ways, with one making reference to, or depending on another. Our intention is to make it possible to specify a variety of these institution independently and in the case that two institutions are related, to make those relationships explicit.

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

As with any complex specification language the potential for errors in institution specifications is high, and as such it is highly desirable to have a reasoning framework in which instances of specifications can be animated, and the presence of various desirable properties verified.

In this paper we report on our initial experiments in capturing some of the concepts above using the Answer Set Programming (ASP) paradigm, we show how institution specifications may be written as answer set programs, and reasoned about using an answer set solver.

Answer set programming formalised as $AnsProlog^*$[4] is a modern logic programming system, designed for semantic clarity, efficient implementation and ease of use for knowledge representation and declarative

* This work was partially supported by the European Fifth Framework Programme under the grant IST-2001-37004 (WASP).

problem solving. It has been under development for the past 15 years and as well as an extensive body of theoretical work, a number of mature implementations [11, 23] exist.

ASP has a variety of powerful and useful features supporting non-monotonic reasoning, handling of multiple possible world views, both classical and epistemic negation and the ability to characterise and reason about partial and incomplete information, it is these capabilities we aim to exploit in modelling and reasoning about institutions.

2 Why Answer Set Programming?

ASP is a powerful and intuitive non-monotonic logic programming language for modelling reasoning and verification tasks. One common question asked of researchers working on non-monotonic logic programming systems such as ASP is 'Prolog has been around for many years and is a mature technology, why not just use that?'. The short answer is that Prolog has a number of limitations both in concept and design that make it unsuitable for many knowledge representation and 'real world' reasoning tasks. As with comparing any languages or language paradigms the key issues here are suitability and ease of expression in the problem domain in question.

Negation is problematic in logic programming languages and Prolog is no exception. A variety of different mechanisms for computing when the negation of a predicate is true and a variety of different intuitions of what this means have been proposed[8]. The most common approach is to compute negation as failure, i.e. $not(p)$ is true if p cannot be proved using the current program; and to characterise this as classical negation i.e. every proposition is either true or false and cannot be both. This combination creates a problem referred to as the closed world assumption when using Prolog to model real world reasoning. By equating negation as failure with classical negation anything that cannot be proven to be true is known to be false, essentially assuming that everything that is known about the world is contained in the program.

In contrast the semantics used in ASP naturally give rise to two different forms of negation, negation as failure and constraint-based negation. Negation as failure, (i.e. we cannot prove p to be true) is characterised as epistemic negation, (i.e. we do not know p to be true). Constraint-based negation introduces constraints that prevent certain combinations of atoms from being simultaneously true in any answer set. This is characterised as classical negation as it is possible to prevent a and $\neg a$ both being simultaneously true, a sufficient condition for modelling classical negation. This is a significant advantage in some reasoning tasks as it allows reasoning about incomplete information, and is supported by the intuition that "I do not know that P is true" (auto-epistemic negation) and "I know that P is not true" (classical negation) are fundamentally different. Critically the closed world assumption is not present in ASP, as negation as failure is not associated with classical negation.

One key difference with Prolog is that the semantics of ASP clearly give rise to multiple possible world views in which the program is consistent. The number and composition of these varies with the program. Attempting to model the same ideas in Prolog can lead to confusion as the multiple possible views may manifest themselves differently dependant on the query asked. In ASP terms Prolog would answer a query on a as true if there is at least one answer set in which a is true. However there is no notion of in which answer set this is true. Thus a subsequent query on b might also return true, but without another query it would not be possible to infer if a and b could be simultaneously true.

3 Answer Set Semantics

There is a large body of literature about ASP but it is largely unknown in the agents community, for in-depth coverage see [4]. For the sake of this paper we provide a brief overview.

$AnsProlog^*$ uses a language that has *terms* which are inductively closed. A *term* is a variable or a *constant*. An *atom* is denoted $a(t_1, \ldots, t_n)$, where a is a predicate of arity n and $t_1, \ldots, t_n$, its arguments, are terms. A term or an atom is called *ground* if it does not contain any variables. A literal is an atom $a(t_1, \ldots, t_n)$ or its negation $\neg a(t_1, \ldots, t_n)$, where $\neg$ should be read in the classical sense (i.e. $a(t_1, \ldots, t_n)$ is proven to be false). An extended literal is a literal L or $\mathbf{not}\ L$ with $\mathbf{not}$ being negation as failure (L cannot be proven to be true).

An $AnsDatalog^*$ program is made up of a series of *rules*. Each rule has the form:

$$L_0 \leftarrow L_1, \ldots, L_n, \mathbf{not}\ L_{n+1}, \ldots, \mathbf{not}\ L_m$$

.

Where L_0 is a *literal* or $\perp$ and L_i for $i \in [1, m]$ are literals. L_0 is the *head* of the rule, denoted $H(r)$ for rule r and $\{L_1, \ldots, L_m\}$ is the *body*, denoted $B(r)$. The intuition for this rule is that if all of $L_1, \ldots, L_n$ are known and none of $L_{n+1}, \ldots, L_m$ are known then L_0 is considered to be known (in the case that L_0 is $\perp$, this indicates a contradiction).

When speaking about the status of rules with respect to a given set of ground literals the terms *applicable* and *applied* are used. A rule is said to be applicable with respect to a set if all of $L_1, \ldots, L_n$ and none of $L_{n+1}, \ldots, L_m$ are in the set. It is applied if it is applicable and L_0 is also in the set.

In order for a program to obtain its full semantics, all variables that appear in the program need to be replaced by values. This process is called *grounding*. The values that a variable can take are defined by the ground terms in your program. Having these, a *ground instance* of a rule may be obtained by replacing each variable symbol by one of these values. The *ground version* of a program is the set of all ground instances of all the rules in the program.

In this paper we shall use the characterisation of answer set semantics given by [16]. This is divided into two sections, the semantics of ground programs that do not contain negation and a semantic criterion and reduct for removing negation. Ground programs without negation as failure ($\mathbf{not}$)(also referred to as $AnsDatalog^{-\mathbf{not}}$) each have at most one answer set. It can be obtained from the logical closure of the rule set, i.e. starting with the facts (rules that have no body and are thus not dependent on anything), recursively build a set of anything that can be concluded using a rule who's body is in the set.

To remove negation the Gelfond-Lifschitz reduct (or transformation)[15] is used, working with respect to a set of ground literals S:

- Removing every rule that contains $\mathbf{not}\ p$ in the body if $p \in S$
- Removing all remaining negative literals (i.e. $\mathbf{not}\ q$) from the rules

The answer sets of the program are the sets of literals S such that S is the answer set of the reduced program.

In short the answer sets of a program can be thought of as all of the possible world views that can be supported by the rules. For example, program: P:

$$\{a \leftarrow b; c \leftarrow \mathbf{not}\ d, a;\ d \leftarrow \mathbf{not}\ c; b\ ; e \leftarrow d\}$$

has two answer sets $\{a, b, c\}$ and $\{a, b, d, e\}$. When reduced with respect to $\{a, b, c\}$, only one rule is removed resulting the program:

$$\{a \leftarrow b\ ;\ c \leftarrow a\ ;\ b\ ;\ e \leftarrow d\}$$

which has the answer set $\{a, b, c\}$ (thus making it an answer set of P). Note that e is not included in the answer set of the reduced program as there is no way of concluding d and so the rule giving e cannot be used. On the other hand if P is reduced by $\{a, b, e\}$ then the following program is obtained:

$$\{a \leftarrow b\ ;\ c \leftarrow a\ ;\ d \leftarrow a\ ;\ b\ ;\ e \leftarrow d\}$$

which has the answer set $\{a, b, c, d, e\}$, which is not the same as the set used to perform the reduct and thus not an answer set of P. Notice that each answer set is a set of literals in which every rule in a program is either applied or not applicable. The converse is not true: a set of literals that makes every rule in a program not applicable or applied is not necessarily an answer set consider the set $\{a, b, c, e\}$ and the program P.

Algorithms and implementations for obtaining answer sets of (possibly un-ground) logic programs are referred to as *answer set solvers*. The most popular and widely used solvers are Smodels[23] and DLV[11]. For the development of this paper we used Smodels.

4 Specification of Agent-based Social Institutions

The foundation of our approach revolves around the descriptions of *institutional states*, and how these evolve over time. We define an institutional state as a set of *institutional facts* (cf. the definition of institutional fact in philosophy [19]) which may be held to be true at given point in time. These facts may be broken down into *institutional domain facts*, which are dependant on the institution being modelled (such as "A owns something" in our example below), and *normative facts* which are common to all specifications, which may be classified as follows.

Institutional Power We incorporate the notion of explicit institutional power (based on the formalisation in [20]) this may be summarised as the capability of an agent to bring about a change in some facts in the institutional state. We do not represent the power to change institutional facts directly, instead allow institutional facts which describe agents' ability to perform empowered *institutional actions*, (see below). In this case power separates meaningful (empowered) actions, which may have an effect, from non-empowered (meaningless) actions.

Permission This describes an agent's ability to perform some institutional action without sanction. Each permission fact captures the property that an agent is allowed to perform a given empowered institutional action. If an agent performs an empowered institutional action, and that action is not permitted, then a violation on that agent occurs.

Obligation Obligation facts are modelled as the dual of permission and are targeted towards a particular agent. We draw on the the formalism for obligations with explicit deadlines described in [14] where each obligation is associated with a corresponding deadline (this is is similar to Singh's formalisation of conditional commitment in [24]). In this paper we limit the obligations to those of the form $O_A(done(\alpha) \preceq \delta)$ where α is some action and δ is some deadline, which may be read as "Agent A is obliged to have done action α before deadline δ".

Violation Violation facts model the consequence of an agent either performing an action for which they did not have permission or not performing an action they were obliged to do before the deadline state of that obligation was reached. At present we do not model the effect of sanctions or an agent's ability to recover from violation.

Within our specifications we define a number of abstract *institutional action* descriptions, each of these notionally associates the satisfaction of some conditions in the current institutional and/or world state (such as the issuing of an utterance) with some consequences on the institution state. Actions are said to be have been *performed* by an agent if it caused these conditions to be met , and *validly performed* if the action was performed and the agent was empowered to do perform the action. In order to capture relationships between institutions we also allow an action's consequences to explicitly cause performance of an action in another institution.

Finally, the operational semantics of a specification are given by a set of *social constraints* which describe how institutional facts may be determined and evolve over time. Constraints may describe static declarative dependencies between institutional facts (such as "If an agent is empowered to perform an action then they are also permitted to perform that action") and causal rules which describe the effects of validly performed actions, such as "An obligation to pay before some time is caused by the valid performance of a buy action".

5 Expressing Institution Specifications in ASP

We express institution specifications as a set of ASP rules which describe possible values for each institutional fact at a given instance of time. The general form of these rules for determining the value of a fact of type f with parameters $Fp_1, \ldots, Fp_n$ at time $I$is as follows:

$f(Fp_1, \ldots, Fp_n, I) \leftarrow cons_1(..., I), \ldots cons_n(..., I).$

Where $cons_1, \ldots, cons_n$ are atoms (denoting the of state of some institutional facts at time I) which must hold true at time I. We express change in the value of institutional facts using a *frame rule* of the form:

$f(Fp_1, \ldots, Fp_n, I+1) \leftarrow cons_1(..., I), \ldots cons_n(..., I).$

Where $cons_1 \ldots, cons_n$ are atoms which must hold in the previous state (such as the occurrence of an action or, the value of one or more institutional states). In some cases we wish the value of some institutional fact to have *inertia* that is it should stay the same in the next state, unless something causes it to stop holding, we express inertia using classical negation as follows:

$f(Fp_1 \ldots, Fp_n, I+1) \leftarrow f(Fp_1 \ldots, Fp_n, I), \text{not } \neg f(Fp_1 \ldots, Fp_n, I+1).$

Which states that f holds in the next state, if it held in the previous state and we cannot show that it does not hold in the next state. A corollary of this is that inertial facts must be terminated by causal rules of the form $\neg f(Fp_1 \ldots, Fp_n, I) \leftarrow \ldots$.

Institutional Actions: Each action description (denoting the possible performance of an institutional action) is represented in ASP by a set of atoms of the form[1]:

$iact(actType(Ap_1, \ldots, Ap_n))$

Where $actType(Ap_1, \ldots, Ap_n)$ denotes an action type and its parameters, which may refer to agents, or objects in the domain of the specification.

We represent the occurrence of an institutional action at a given time with a set of atoms of the form: $iact_happened(Agent, IAct, I)$ meaning $Agent$ has performed the necessary conditions for $IAct$ to have occurred at time I (Note that the effects of $IAct$ are not performed unless $Agent$ was also empowered to perform the action (see institutional power below)).

Institutional Power Power is modelled through a set of atoms of the form: $pow(Agent, IAct, I)$ which state that a given agent $Agent$ has the power to enact institutional action $IAct$ at time I. Power atoms are used to determine when performance institutional action is considered to be valid (i.e. has some institutional effect), this fact is recorded through atoms of the form

[1] Note that in this and subsequent examples we use the symbolic function extension to conventional ASP syntax; $iact(actType(X, Y, Z))$ is equivalent to there being a set of atoms $iact(\alpha)$ with α ranging over $X \times Y \times Z$ for all grounded values of X,Y and Z.

$valid_act_happened(Agent, IAct)$

these atoms are inferred for using the following rule

$valid_act_happened(Agent, IAct, I) \leftarrow$
$\quad iact_happened(Agent, IAct, I), pow(Agent, IAct, I).$

Which states that at a given time, if an agent causes the conditions for some institutional action $IAct$ to be met at time I, and that action was empowered for that agent, then a valid occurrence of $IAct$ action occurred.

Permission and Violation: Permission is represented in a similar way to power as a set of atoms of the form: $permi(Agent, IAct, I)$.

The presence of permission entails the possibility for violation and violations are modelled as a set of atoms of the form $viol(Agent, I)$ indicating that $Agent$ is in a state of violation at time I. Violation atoms (in the case of performing an action which is not permitted (see obligation with deadlines, below)) are determined using the following causal rule:

$viol(Agent, I + 1) \leftarrow$
$\quad valid_act_happened(Agent, IAct, I), \text{not } permi(Agent, IAct, I).$

which states that at time $I + 1$ agent is in violation if it validly performed action $IAct$ at time I and it was not permitted to do so at time I.

Obligation and Deadlines: Each deadline is declared with an atom of the form $deadline(Deadline)$. Deadlines expire when some deadline condition is satisfied, a fact which is modelled by atoms of the form $deadline_sat(Deadline, I)$ indicating that $Deadline$ is satisfied at time I. Additionally deadlines have implicit inertia, so once they become satisfied they remain satisfied, this is modelled using a frame rule as follows:

$deadline_sat(Deadline, I + 1) \leftarrow$
$\quad deadline_sat(Deadline, I).$

The presence of an obligation on $Agent$ to have performed some institutional action $IAct$ before $Deadline$ is represented with atoms of the form $obl_deadline(Agent, IAct, Deadline, I)$. An obligation is satisfied if there exists a previous valid occurrence of an institutional action which satisfies the obligation and the obligation deadline has not been satisfied:

$obl_deadline_sat(Agent, IAct, Deadline, I) \leftarrow$
$\quad obl_deadline(Agent, IAct, Deadline, I), \text{not } deadline_sat(Deadline, I),$
$\quad valid_act_happened(Agent, IAct, J),\ J < I.$

Once an obligation on $Agent$ is instantiated at time I, the state of the obligation persists until either the obligation is satisfied, or its deadline is satisfied:

$obl_deadline(Agent, Obl, Deadline, I + 1) \leftarrow$
$\quad obl_deadline(Agent, Obl, Deadline, I),$
$\quad \text{not } \neg obl_deadline(Agent, Obl, Deadline, I + 1),$
$\quad \text{not } obl_deadline_sat(Agent, Obl, Deadline, I + 1),$
$\quad \text{not } deadline_sat(Deadline, I + 1).$

Finally a violation against $Agent$ occurs at time $I + 1$ if the obligation deadline is satisfied at time I and the obligation condition has not been satisfied by this time.

$viol(Agent, IAct, I + 1) \leftarrow obl_deadline(Agent, IAct, Deadline, I),$
$\quad deadline_sat(Deadline, I),$
$\quad \text{not } obl_deadline_sat(Agent, Obl, Deadline, I).$

5.1 Making Specifications Executable

In order to make institution specifications in ASP executable, it is necessary to define a set of "real world" actions Act_{ag} which might be performed by participating agents, such that each action in Act_{ag} corresponds to the performance of exactly one institutional action in the modelled specification (this mapping may be partial in the case of derived institutional actions, or if we are only modelling a subset of the institution). By limiting the number of action rules from Act_{ag} which may be inferred at any given time instance with an ASP constraint, we allow the definition of a labelled transition system over the institutional states, this has the effect in ASP of limiting answer sets to those containing action traces of the form:

$ag_act_happened(act_a, 0), ag_act_happened(act_b, 1), \ldots, ag_act_happened(act_x, n)$

and all associated inferable institutional states.

In general we assume that actions in Act_{ag} model communicative actions, and as such may be performed (albeit invalidly) by any agent at any time, this condition is necessary in the case of prediction and postdiction

queries (see below) (where a chain of actions may have occurred, but due to one or more actions not being empowered no corresponding change in institutional state occurred). However in the case of planning queries where we wish to determine if a given institutional state can be obtained, we can omit meaningless actions (as they have no possible effect on the institution state) from the transition system.

5.2 Specification queries

We identify three classes of query (from [3]):

Prediction: Where we know that a given sequence of events has occurred and we wish to determine some information about the institution state at some point along this trace.

Postdiction: In which we have some information about a final state and partial information about the initial state and the sequence of events which led us to this state and we wish to determine some additional information about the initial state.

Planning: Where given an initial state we wish to determine one or more sequences of agent actions which lead us to a desired final state.

Queries are specified in ASP by encoding a description of the initial state and then computing answer sets which include the states specified by the query. In the case of prediction and planning the initial state description is known and is asserted as a set of facts in the program. In the case of postdiction the initial state description is expressed as a set of choice rules denoting all possible initial states. If the query is satisfied then the result is one ore more answer sets describing possible traces which satisfy the query. Verification questions will in general be expressed as planning queries describing desirable or undesirable states, for example with simple validation, "given initial state, is this outcome ever possible", or more complex query to determine conflicts between two institutions which regulate a common set of agents: "is it possible for an agent to be in a state of obligation but unable or forbidden to dispense that obligation".

6 An Example

In order to illustrate our approach we specify a simplified institution of property (ownership of goods) and a related institution for exchanging goods with payment.

In our institution of property we wish to describe one type of institutional domain fact F_1 which captures the state of ownership of some type of object by one of a set of agents, and a single institutional action description A_1 which accounts for the transfer of ownership from one agent to another. We also wish to include the following social constraints.

C_1**:** After a valid transfer of ownership of an object the recipient of the transfer becomes the owner of the object.

C_2**:** After a valid transfer of ownership of an object the original owner ceases to be the owner of the object.

C_3**:** Agents are permitted to transfer ownership of objects, if they own them.

C_4**:** Transfers are empowered if the initiator of the transfer is the owner of the object being transferred.

The state of F_1 over time is modelled with a set of atoms of the form:

$owns(Agent, Object, I).$

As this fact has inertia we also add the following frame rule:

$owns(Agent, Object, I+1) \leftarrow \text{not } \neg owns(Agent, Object, I+1),$
$owns(Agent, Object, I).$

The sorts of action described by A_1 are specified with a set of atoms of the form:

$iact(transfer_ownership(FromAgent, ToAgent, Object)).$

C_1 and C_2 are encoded with the following rules:

$owns(ToAgent, Object, I+1) \leftarrow valid_act_happened(FromAgent,$
$transfer_ownership(FromAgent, ToAgent, Object), I).$

$\neg owns(FromAgent, Object, I+1) \leftarrow valid_act_happened(FromAgent,$
$transfer_ownership(FromAgent, ToAgent, Object), I).$

C_3 is encoded as follows:

$permi(FromAgent, transfer_ownership(ToAgent, Object), I) \leftarrow$
$owns(FromAgent, Object, I).$

and C_4 as follows:

$pow(FromAgent, transfer_ownership(FromAgent, ToAgent, Object), I) \leftarrow$
$owns(FromAgent, Object, I), ToAgent \neq FromAgent.$

The exchange institution is described below, in this institution we describe a small family of protocols where goods are exchanged for some payment, the scenario allows encodes the following five actions:

A_2 **Request Goods:** A customer sends a request for some goods to a merchant.
A_3 **Refuse Request:** The merchant refuses a request from a customer.
A_4 **Send Goods:** The merchant sends goods to the customer.
A_5 **Send Payment:** The customer sends payment for the good.
A_6 **Send Receipt:** The merchant sends a receipt to the customer.

We impose also impose the following constraints:

C_5 : Sending a request for goods (A_2) creates an obligation on the merchant to have sent the goods before the interaction ends (C_5).
C_6 **:** Sending a refusal (A_3) cancels the merchants obligation to send goods.
C_7 **:** Sending goods (A_4) creates an obligation on the customer to have payed for the goods before the interaction ends.
C_8 **:** Sending payment creates an obligation on the merchant to have sent a receipt for the payment before the interaction ends.
C_9 **:** Customers are initially empowered to perform actions of type A_2, A_5.
C_{10} **:** Merchants are initially empowered to actions of type A_3, A_4
C_{11} **:** All actions are permitted if they have not already been performed (i.e. all agents are only permitted to perform each action once).
C_{12} **:** Sending a receipt (A_6) is empowered only if an agent has received a valid payment (A_5) in the past.

We additionally wish to express the following relationship between the exchange scenario actions and the property institution:

C_{13} **:** If both a valid Send Goods (A_4) action a valid Send Payment (A_5) action take place between two agents then a transfer of ownership occurs .

Atoms for actions $A_{2,\ldots 6}$ are declared as follows:

$iact(sendRequest(Cust, Merch, Object)).iact(sendRefuse(Merch, Cust, Object)).$
$iact(sendGoods(Merch, Cust, Object)).iact(sendPayment(Cust, Merch, Object)).$
$iact(sendReceipt(Merch, Cust, Object)).$

Where $Cust$, $Merch$ are agent atoms, $Object$ range over atoms matching the domain predicate $object(Object)$ which is shared with the ownership institution.

Constraints C_5 and C_6 are written as follows (C_7 and C_8 are omitted for space reasons):

$obl_deadline(Merch, sendGoods(Merch, Cust, Object), end_int, I + 1) \leftarrow$
$\quad valid_act_happened(Cust, sendRequest(Cust, Merch, Object), I).$
$obl_deadline_sat(Merch, sendGoods(Merch, Cust, Object), end_int, I + 1) \leftarrow$
$\quad obl_deadline(Merch, sendGoods(Merch, Cust, Object), Deadline, I),$
$\quad valid_act_happened(Cust, sendRefuse(Merch, Cust, Object), I).$

The translations of $C_9, \ldots, C_{12}$ are omitted from this description.
C_{13} is written using the following rules (indicating either of the orderings of A_4 and A_5.

$iact_happened(Merch, transfer_ownership(Merch, Cust, Object), I) \leftarrow$
$\quad valid_act_happened(Merch, sendGoods(Merch, Cust, Object), I),$
$\quad J < I, valid_act_happened(Cust, sendPayment(Cust, Merch, Object), J).$
$iact_happened(Merch, transfer_ownership(Merch, Cust, Object), I) \leftarrow$
$\quad valid_act_happened(Cust, sendPayment(Cust, Merch, Object), I),$
$\quad J < I, valid_act_happened(Merch, sendGoods(Merch, Cust, Object), J).$

A sample validation query is described as follows, given a merchant *alis* and a customer *bob* and one object *soft* and an initial state of *alis* owning *soft* we wish to determine if there is a valid sequence of actions after which *bob* owns *soft* during which time neither *alis* or *bob* are ever in violation. In ASP the domain,initial state and query are encoded as follows:

$agent(alis; bob).$
$merchant(alis).customer(bob).$
$time(0..3).$
$owns(alis, soft, 0).$
$compute\ all\{owns(alis, soft, 3), not\ viol(alis, 3), not\ viol(bob, 3)\}.$

Two answer sets are produced indicating traces of actions corresponding to $< A_4, A_5, A_6 >$ and $< A_5, A_4, A_6 >$. Figure 1 shows how the social states (in bold) evolve in relation to the first trace (facts relating to permission and power have been omitted).

```
=========== ANSWER SET 1 ===========
owns(alis,soft,0)
ag_act_happened(bob,sendPayment(bob,alis,soft),0)
iact_happened(bob,sendPayment(bob,alis,soft),0)
valid_act_happened(bob,sendPayment(bob,alis,soft),0)
owns(alis,soft,1)
obl_deadline(alis,sendReceipt(alis,bob,soft),end_i,1)
ag_act_happened(alis,sendGoods(alis,bob,soft),1)
iact_happened(alis,sendGoods(alis,bob,soft),1)
iact_happened(alis,transfer_ownership(alis,bob,soft),1)
valid_act_happened(alis,sendGoods(alis,bob,soft),1)
valid_act_happened(alis,transfer_ownership(alis,bob,soft),1)
-owns(alis,soft,2)
owns(bob,soft,2)
obl_deadline(alis,sendReceipt(alis,bob,soft),end_i,2)
obl_deadline(bob,sendPayment(bob,alis,soft),end_i,2)
obl_deadline_sat(bob,sendPayment(bob,alis,soft),end_i,2)
ag_act_happened(alis,sendReceipt(alis,bob,soft),2)
iact_happened(alis,sendReceipt(alis,bob,soft),2)
valid_act_happened(alis,sendReceipt(alis,bob,soft),2)
deadline_sat(end_i,3)
owns(bob,soft,3)
obl_deadline_sat(alis,sendReceipt(alis,bob,soft),end_i,3)
obl_deadline_sat(bob,sendPayment(bob,alis,soft),end_i,3)
```

Fig. 1. First answer set for example query

7 Discussion and Related Work

Normative and institutional aspects of multi-agent systems have been studied extensively in recent years, while complete account of related work is beyond the scope of this paper, however some recent work deserves mention.

In [18] Vázquez-Salceda, Dignum et al outline the need for an *operational* system for expressing norms which allows for both their interpretation and also their efficient implementation and enforcement. In their work (including [14, 9]) they outline a language for expressing norms, their approach describes three types of deontic modality (OBLIGED, PERMITTED, FORBIDDEN) which may refer to either actions or states and which may be predicated on system states including temporal (BEFORE, AFTER) references to the occurrence of actions. As well as capturing a concise social semantics for norms they also extend their descriptions to include advisory properties which make explicit how the violation of norms should be detected by an agent responsible for the enforcement of a norm, and plans which describe how such agents should go about sanctioning violating agents. In their approach, unlike ours social states beyond those related to the deontic properties described above are considered as external to the description of norms. The mechanisms for determining these states are not specified as part of the description of norms.

Colombetti et al in [5] outline an abstract model for agent institutions based on social commitments. Their model describes institutions as being composed of a set of *registration rules* which deal with the entry and exit of agents from institutions, a set of *interaction rules* which govern how commitments are created and dispensed between agents, a set of *authorisations* which describe agents innate abilities to perform certain actions and an *internal ontology* which describes a model for the interpretation of terms relevant to the institution. A number of aspects of this approach correspond with our model for describing institutions as outlined in Section 4, in particular interaction rules correspond to our social rules, authorisations with our treatment of institutionalised power and the internal ontology with our domain specific rules. One particularly appealing aspect of their approach (further expanded in [13, 6, 25]) is the notion that institutions of this type can be applied to the specification of agent communication languages in general, with social consequences such as commitments between agents being ascribed to speech acts of a particular type. The combination of "base-level" institutions such as these with institutions capturing more general social properties such as ours provides an interesting area for further study.

The types of specification we describe are closely related to the work of Artikis et al described in[1–3, 21] from which we derive much of our specification model. In their work specifications of social systems are formalised in both the event calculus [22] and using a subset of the action language $C+$ [12]. Intuitively our approach is capable of expressing similar constraints and social properties as specifications in the above languages. However as our approach lacks a formal basis beyond its syntax in ASP at present, we are unable to make a formal comparison. In comparison to $C+$, which has similar reasoning capabilities (with similar complexity) to ASP using the CCalc tool, we feel our approach yields a more intuitive way of expressing social constraints which include temporal aspects such as C_{13} in our example (in $C+$ the program must be modified to record action histories). This also extends to the formulation of queries, where ASP makes it possible to encode queries similar to those found in (bounded) temporal logic model checking. As with $C+$, the properties we can verify using our approach are limited to those which can be found in models of specifications of a limited depth

in time and with a somewhat limited number of grounded actions, states, and agents, this is partly a constraint on the grounding process used in Smodels which requires that all possible atoms be grounded and stored in memory before answer sets are computed and partly due to the implicit complexity of computing answer sets of large models. Despite this constraint, early results indicate that even for relatively complex models which ground to hundreds of thousands of rules interesting properties may still be shown in reasonable time.

In this paper we have not dealt directly with expressing sanctions on violating agents, or agents' ability to recover from violations. Intuitively these may be expressed in our framework as follows: a sanction on a violating agent may be expressed as a permission and/or obligation and/or empowerment on a third party agent or agents to perform some sanction action or actions. Recovery from sanction would then be expressed as effects of the successful application of the sanction action(s).

We have not discussed a general mechanism for representing institutional roles which give a convenient way of referring to groups of permissions, obligations and empowerments in a variety of institutions, intuitively the property of an agent assuming a particular role may be expressed as an institutional fact which evolves in the same way as other normative facts which may then be used as constraints on the application or social rules according to the roles they are relevant to, the modelling of this is left to future work.

Finally, in this paper we have focussed on using ASP to reason about institutions from a design perspective. In [7] we describe an extension to ASP for reasoning within communicating agents, it would be interesting to see if these two approaches can be combined to allow agents to reason about institution descriptions online.

References

1. Alexander Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, September 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. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
5. 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.
6. Marco Colombetti and Mario 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.
7. Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artifical Intelligence*, 42(1–3):103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.
8. Mark Denecker. What's in a Model? Epistemological Analysis of Logic Programming. Ceur-WS, September 2003. online CEUR-WS.org/Vol-78/.
9. Virginia Dignum, John-Jules Meyer, Frank Dignum, and Hans 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, October 2003.
10. Douglass C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.
11. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
12. Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence, Vol. 153, pp. 49-104*, 2004.
13. Nicoletta Fornara and Marco 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.
14. Frank Dignum, Jan Broersen, Virginia Dignum, and John-Jules Meyer. Meeting the Deadline: Why, When and How. In Michael G. Hinchey, James L. Rash, and Walter F. Truszkowski, editors, *Proceedins of the 3rd Conference on Formal Aspects of Agent-Based Systems (FAABS III), Greenbelt, Maryland, USA*, volume 3228 of *Lecture Notes in Computer Science*, pages 30–40. Springer-Verlag, 26 April 2004.
15. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.
16. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.
17. Frank Guerin and Jeremy Pitt. Denotational semantics for agent communication language. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 497–504. ACM Press, 2001.

18. Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Imlementing Norms in Multiagent Systems. In Gabriela Lindemann, Jörg Denzinger, Ingo J. Timm, and et al., editors, *Multiagent System Technologies: Second German Conference, MATES 2004, Erfurt, Germany*, volume 3187 of *Lecture Notes in Computer Science*, pages 313–327. Springer Verlag GmbH, September 2004.
19. John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.
20. Andrew J.I. Jones and Marek Sergot. A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4es):121, 1996. Read 28/11/2004.
21. 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.
22. R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.
23. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil 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.
24. Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.
25. Mario Verdicchio and Marco 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.

Appendix

The second answer set computed from the example above is as follows:

```
=========== ANSWER SET 2 ===========
owns(alis,soft,0)
ag_act_happened(alis,sendGoods(alis,bob,soft),0)
iact_happened(alis,sendGoods(alis,bob,soft),0)
valid_act_happened(alis,sendGoods(alis,bob,soft),0)
obl_deadline(bob,sendPayment(bob,alis,soft),end_i,1)
owns(alis,soft,1)
ag_act_happened(bob,sendPayment(bob,alis,soft),1)
iact_happened(alis,transfer_ownership(alis,bob,soft),1)
iact_happened(bob,sendPayment(bob,alis,soft),1)
valid_act_happened(alis,transfer_ownership(alis,bob,soft),1)
valid_act_happened(bob,sendPayment(bob,alis,soft),1)
-owns(alis,soft,2)
owns(bob,soft,2)
obl_deadline(alis,sendReceipt(alis,bob,soft),end_i,2)
obl_deadline_sat(bob,sendPayment(bob,alis,soft),end_i,2)
ag_act_happened(alis,sendReceipt(alis,bob,soft),2)
iact_happened(alis,sendReceipt(alis,bob,soft),2)
valid_act_happened(alis,sendReceipt(alis,bob,soft),2)
owns(bob,soft,3)
deadline_sat(end_i,3)
obl_deadline_sat(alis,sendReceipt(alis,bob,soft),end_i,3)
obl_deadline_sat(bob,sendPayment(bob,alis,soft),end_i,3)
```