Making Golog Norm Compliant

Alfredo Gabaldon

Center for Artificial Intelligence (CENTRIA)
Universidade Nova de Lisboa
ag@di.fct.unl.pt

Abstract. In this work we consider how to enforce norms in the Situation Calculus based programming language Golog and its relatives. We define a notion of *norm compliant sequence of actions* with respect to norms prescribing some actions to be forbidden or obliged (ought-to-do norms), norms prescribing that a state-condition is forbidden (ought-to-be norms) and norms that are a form of deadline. We then show a procedure that allows incorporating the norms into the underlying action theory so that after this is done, the agent's behavior is guaranteed to be norm compliant.

1 Introduction

The use of *social laws* or *norms* as a behavior and coordination mechanism has attracted considerable interest among researchers in the area of autonomous agents and multi-agent systems. The work in this area includes [1,2,3,4,5,6] among many others.

Much of the current work on norms in autonomous agents involves developing agent programming languages that include facilities for expressing norms and mechanisms for enforcing them. Along these lines, in this work we look at an agent programming language and consider adding expressions for describing norms and then consider what it means for an agent programmed in this language to comply with the norms. The particular language we consider is Golog [7]. This high-level action programming language was developed for providing artificial agents with complex behaviors defined in terms of a set of primitive operations or actions. Golog consists of a set of programming constructs typical of imperative programming languages, e.g. sequence, conditional, iteration, and also some non-deterministic constructs such as choice between two subprograms. A distinguishing feature of Golog is that the primitive constructs are *actions* formalized in an underlying logic—the Situation Calculus [8].

Golog has grown into a family of languages which extend it with various features such as concurrency (ConGolog [9]), decision theory (dtGolog [10]), and incremental execution and sensing (IndiGolog [11]), among others. The underlying action language has also undergone substantial development with extensions including adding explicit time, an epistemic modality for knowledge, and stochastic actions. This makes the Golog family of languages an attractive choice for programming autonomous agents.

2 The Golog Language

We briefly review the main components of a Basic Action Theory [12,13] and of the Golog language [7].

2.1 Basic Action Theories

A *basic action theory* is a classical logic formalization of the dynamic domain of an agent(s) in the *Situation Calculus* (SitCalc for short) [14]. The ontology of the SitCalc includes *actions*, *fluents*, which are the properties of the domain that change when actions occur, and *situations*, which are sequences of actions representing possible ways in which the domain may evolve.

Formally, the SitCalc is a dialect of First-Order logic with sorts *action, situation*, and *object*. Consequently, actions, situations and domain objects are treated as quantifiable, first-class citizens in the language. A special constant S_0 is used to denote the initial situation, and the function *do* of sort $(\mathit{action} \times \mathit{situation}) \mapsto \mathit{situation}$ is used to form sequences of actions. For instance, a sequence consisting of actions a_1, a_2, a_3 is represented by the term $do(a_3, do(a_2, do(a_1, S_0)))$.[1]

Fluents are represented by means of relations $F(\boldsymbol{x}, s)$ where $\boldsymbol{x}$ is a tuple of arguments of sorts *object* or *action* and the last argument s always of sort *situation*. For example, a fluent $owner(ag, file, s)$ could be used to represent that an agent ag is the owner of a $file$ in situation s.

Function symbols of sort $\mathit{object} \mapsto \mathit{action}$, $A(\boldsymbol{x})$, represent action types. For instance, a function $write(ag, file)$ could be used to represent the action of an agent ag writing to a $file$. We call them action *types* because a single function symbol can be used to create multiple *instances* of an action, e.g. the instances $write(Ag_1, File_1)$, $write(Ag_2, File_2)$, etc. We will use $a_1, a_2, \ldots$ to denote action variables and $\alpha_1, \alpha_2, \ldots$ to denote action terms. Similarly, we use $s_1, s_2, \ldots$ for situation variables and $\sigma_1, \sigma_2, \ldots$ for situation terms.

A Basic Action Theory $\mathcal{D}$ consists of the following sets of axioms (variables that appear free are implicitly universally quantified. $\boldsymbol{x}$ denotes a tuple of variables $x_1, \ldots, x_n$):

1. For each action type $A(\boldsymbol{x})$ there is exactly one **Action Precondition Axiom** (APA), of the form:

$$Poss(A(\boldsymbol{x}), s) \equiv \Pi_A(\boldsymbol{x}, s)$$

   where variable s is the only term of sort situation in formula $\Pi_A(\boldsymbol{x}, s)$. The latter formula represents the conditions under which an action $A(\boldsymbol{x})$ is executable. The restriction that the only situation mentioned in this formula is s intuitively means that these preconditions depend only on the situation where the action would be executed.

[1] $do([a_1, a_2, \ldots, a_n], s)$ is an abbreviation of $do(a_n, do(a_{n-1}, \ldots, do(a_1, s) \ldots))$.

2. For each fluent $F(\boldsymbol{x}, s)$, there is exactly one **Successor State Axiom** (SSA), of the form:

$$F(\boldsymbol{x}, do(a, s)) \equiv \Phi_F(\boldsymbol{x}, a, s)$$

where s is the only term of sort situation in formula $\Phi_F(\boldsymbol{x}, a, s)$. This formula represents all and the only conditions under which executing an action a in a situation s results in a situation $do(a, s)$ where the fluent holds. These axioms embody Reiter's solution to the frame problem [12,13].
3. A set of sentences $\mathcal{D}_{S_0}$ describing the initial state of the world. This is a finite set of sentences whose only situation term may be the constant S_0 and describe the initial state of the domain. Any sentence is allowed as long as the only situation variable that appears in it is S_0, so one can write sentences such as $(\exists ag)owner(ag, File_1, S_0)$, reflecting incomplete information about the initial state.
4. The **Foundational Axioms** Σ which define situations in terms of the constant S_0 and the function *do*. Intuitively, these axioms define a tree-like structure for situations with S_0 as the root of the tree. They also define relation $\sqsubset$ on situations. Intuitively, $s \sqsubset s'$ means that the sequence of actions s is a prefix of sequence s'.
5. A set of unique names axioms (UNA) for actions. For example, $write(ag, f) \neq delete(ag, f)$, $write(ag, f) = write(ag', f') \supset (ag = ag' \wedge f = f')$, etc.

Given a basic action theory $\mathcal{D}$ we can define a few basic reasoning tasks. For instance, checking if a sequence of actions is executable, i.e. physically possible for the agent according to the axiomatization of its dynamic environment. This check is formally defined as follows: let $\alpha_1, \ldots, \alpha_k$ be action terms

$$\mathcal{D} \models executable(do([\alpha_1, \ldots, \alpha_k], S_0))$$

where $executable(\cdot)$ is defined as follows:

$$executable(s) \equiv (\forall a, s').do(a, s') \sqsubseteq s \supset Poss(a, s').$$

Another reasoning problem is *projection*: checking if some condition, denoted by a formula $\phi(s)$ with a free variable s, holds after a sequence of actions is executed:

$$\mathcal{D} \models \phi(do([\alpha_1, \ldots, \alpha_k], S_0)).$$

2.2 Golog

The situation calculus based programming language Golog [7] and variants such as ConGolog [9] and IndiGolog [11], provide Algol-like programming constructs for defining complex behaviors in terms of the primitive actions formalized in a basic action theory of the form described above. Among various applications, these languages have been employed for programming autonomous agents. For the purpose of this work, which mainly deals with the logic underlying these languages, we need not go into the details. Here we will refer to Golog when generally referring to the family of Golog variants.

In addition to atomic actions, which are the primitive construct in Golog, the language includes constructs such as a test $\phi?$, test whether ϕ currently holds; sequence $\delta_1; \delta_2$, execute program δ_1 followed by δ_2; non-deterministic choice $\delta_1|\delta_2$, choose between executing δ_1 and executing δ_2; among others.

An important aspect of these languages is the fact that they allow one to write non-deterministic programs. Intuitively, the execution of a program $\delta_1|\delta_2$ can result in the execution of either one of δ_1 and δ_2, as long as their execution is successful. A program may fail to execute if one of the primitive actions in its execution trace turns out not to be executable. In the case of programs run concurrently, e.g. $\delta_1\|\delta_2$, the result of the execution is any of the possible interleavings of the primitive actions that result from δ_1 and those from δ_2. Again, some interleavings may fail because one of the actions is not executable at the given time.

The semantics for these languages is captured through a relation $Do(\delta, s, s')$, meaning that executing program δ in situation s results in situation s'. Given a background theory $\mathcal{D}$, including a definition of $Do(\delta, s, s')$, the execution of a program δ in the initial situation S_0 is defined in terms of logical entailment as the problem of finding a sequence of actions $\alpha_1, \ldots, \alpha_k$ such that

$$\mathcal{D} \models Do(\delta,\ S_0,\ do([\alpha_1, \ldots, \alpha_k], S_0)).$$

3 Norms

Social laws, norms or policies, are used as mechanisms for regulating the behavior of agents, as a mechanism for coordination, and for access control in systems security, among others. Many normative system frameworks use pairs of expressions (ϕ, a) to represent norms. The intuitive meaning of a pair (ϕ, a) would be that in states where ϕ holds, the action a is permitted/forbidden/obligatory.

We will represent norms in terms of formulae denoted by $\phi(\boldsymbol{x}, s)$ and $\psi(\boldsymbol{x}, s)$ with free-variables $\boldsymbol{x}, s$, with s the only situation term appearing in the formulae and $\boldsymbol{x}$ the remaining free variables. Norms are enforced in all situations so when writing them we will omit the universally quantified situation variable and write $\phi(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$. We will use the notation $\mathbf{F}\ a$ to denote that an action a is forbidden, $\mathbf{O}\ a$ to denote that a is (immediately) obligatory, and $\mathbf{F}\ \psi$ to denote that a (state) condition ψ is forbidden.

We will assume that actions are permitted and not obligatory by default, that is, if there is no norm that in a given situation says that an action is forbidden (resp. obligatory), then the action is assumed permitted (resp. not obligatory). For this reason, it is unnecessary to write norms using negation in front of $\mathbf{F}\ a$ and $\mathbf{O}\ a$. Modifying the formalization to make the opposite assumptions, for example that actions are assumed forbidden unless a norm says otherwise, is straight forward.

In the norm expressions below, $\boldsymbol{t}, \boldsymbol{v}$ are tuples of terms, $\boldsymbol{t}$ a subset of the terms $\boldsymbol{v}$, and any variables appearing free, including omitted situation variables, are implicitly universally quantified.

3.1 Ought-to-Do Norms

1. Forbidden actions: $\phi(\boldsymbol{t}) \rightarrow \mathbf{F}\ A(\boldsymbol{v})$.
   Example: regular users are not allowed to write to files owned by others:
   $$file(f) \wedge regUsr(r) \wedge \neg owner(r, f) \rightarrow \mathbf{F}\ write(r, f).$$
2. Obligatory actions: $\phi(\boldsymbol{t}) \rightarrow \mathbf{O}\ A(\boldsymbol{v})$.
   Example: if a licensed file has an expired license, the owner must delete it.
   $$file(f) \wedge owner(r, f) \wedge expLic(f) \rightarrow \mathbf{O}\ del(r, f)$$

Given a set of norms, we can define a notion of *compliance* by a program with the norms. To that end, it will be useful to take a set of norms in the above forms and put them in a compact normal form by applying the following steps.

1. Take each norm $\phi(\boldsymbol{t}) \rightarrow \mathbf{F}\ A(\boldsymbol{v})$ and rewrite it so as to replace the parameters with variables::
   $$\phi(\boldsymbol{x}') \wedge \boldsymbol{x}' = \boldsymbol{t} \wedge \boldsymbol{x}'' = \boldsymbol{u} \rightarrow \mathbf{F}\ A(\boldsymbol{x})$$
   where $\boldsymbol{u}$ are the terms in $\boldsymbol{v}$ not in $\boldsymbol{t}$ and $\boldsymbol{x}', \boldsymbol{x}''$ are among the variables $\boldsymbol{x}$. Let us denote the resulting formula by $\Phi(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x})$.
2. Next, for each action type $A(\boldsymbol{x})$ we take all the norms
   $$\begin{array}{l} \Phi_1(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x}) \\ \Phi_2(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x}) \\ \ldots \\ \Phi_k(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x}) \end{array}$$
   and rewrite them as the following single norm for $A(\boldsymbol{x})$:
   $$\left[\bigvee_{i=1\ldots k} \Phi_i(\boldsymbol{x})\right] \rightarrow \mathbf{F}\ A(\boldsymbol{x})$$
   Let us denote the result by $\Phi_A(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x})$. This norm now completely characterizes the conditions that make any instance, i.e. for any $\boldsymbol{x}$, of the action type $A(\boldsymbol{x})$ forbidden.
3. Next we take each norm $\Phi_{A_i}(\boldsymbol{x}_i) \rightarrow \mathbf{F}\ A_i(\boldsymbol{x}_i)$ and rewrite it as follows using a fresh action variable a:
   $$\Phi_{A_i}(\boldsymbol{x}_i) \wedge a = A_i(\boldsymbol{x}_i) \rightarrow \mathbf{F}\ a$$

4. Finally, we gather all the norms

$$\begin{array}{l} \Phi_{A_1}(\boldsymbol{x}_1) \wedge a = A_1(\boldsymbol{x}_1) \ \rightarrow \ \mathbf{F}\ a \\ \Phi_{A_2}(\boldsymbol{x}_2) \wedge a = A_2(\boldsymbol{x}_2) \ \rightarrow \ \mathbf{F}\ a \\ \dots \\ \Phi_{A_n}(\boldsymbol{x}_n) \wedge a = A_n(\boldsymbol{x}_n) \ \rightarrow \ \mathbf{F}\ a \end{array}$$

   and rewrite them as a single expression:

$$\left[\bigvee_{i=1\dots n} \Phi_{A_i}(\boldsymbol{x}_i) \wedge a = A_i(\boldsymbol{x}_i) \right] \ \rightarrow \ \mathbf{F}\ a$$

   Let us denote the result by $\Phi_F(a) \ \rightarrow \ \mathbf{F}\ a$.

Following the same steps with obligation norms, we can obtain the corresponding expression $\Phi_O(a) \ \rightarrow \ \mathbf{O}\ a$.

By restoring the situation argument on the left-hand-side formula to obtain $\Phi_F(a, s)$, resp. $\Phi_O(a, s)$, we now have a legit SitCalc formula that tells us, in any given situation, whether or not an action is forbidden, resp. obligatory, according to the system norms.

We can then define a notion of a sequence of actions s being compliant with a set of norms in the above forms, by means of the following equivalence:

$$\begin{array}{rl} \mathit{compliant}(s) \equiv & s = S_0 \ \vee \\ & (\exists a, s').s = \mathit{do}(a, s') \wedge \mathit{compliant}(s') \ \wedge \\ & \neg\Phi_F(a, s') \wedge (\forall a')[\Phi_O(a', s') \supset a' = a]. \end{array} \tag{1}$$

Intuitively, this says that a sequence of actions s is compliant with the given norms iff s is the empty sequence, S_0, or s is s' followed by a, where s' is compliant and a satisfies the norms in s'.

Given a set of norms and a corresponding definition of $\mathit{compliant}(s)$, we can define compliance with the norms by a Golog program δ. We will say that δ is compliant with the norms if all its execution traces comply with the norms.

Definition 1. *Let $\mathcal{D}$ be a basic action theory, δ a program, and N a set of norms with corresponding definition of $\mathit{compliant}_N(s)$. Program δ is* compliant *with norms N iff*

$$\mathcal{D} \models (\forall s).Do(\delta, S_0, s) \supset \mathit{compliant}_N(s).$$

The above definition assumes the program is executed in the initial situation S_0. A stronger version of compliance of a program can be defined by requiring the program to satisfy the norms when executed in any situation:

$$\mathcal{D} \models (\forall s', s).Do(\delta, s', s) \supset \mathit{compliant}_N(s).$$

On the other hand, a weak version of compliance can be simply defined by just requiring the program to have at least one compliant execution trace:

$$\mathcal{D} \models (\exists s).Do(\delta, S_0, s) \wedge compliant_N(s).$$

In terms of analyzing the norms themselves, a problem that is often of interest is that of checking whether two sets of norms are equivalent or if one set is subsumed by another. These problems can be defined in terms of classical entailment in our language. Consider two sets of norms N_1, N_2 with corresponding definitions of compliance denoted by $compliant_{N_1}(\cdot)$ and $compliant_{N_2}(\cdot)$ and defined by an equivalence of the form (1).

Definition 2. *We say that the sets of norms N_1, N_2 are equivalent (wrt $\mathcal{D}$) iff*

$$\mathcal{D} \models (\forall s).compliant_{N_1}(s) \equiv compliant_{N_2}(s).$$

Intuitively, the norms are said to be equivalent if the sequences that are compliant under one set of norms are exactly the same sequences that are compliant under the other set of norms.

Definition 3. *We say that the set of norms N_1 subsumes the set of norms N_2 (wrt $\mathcal{D}$) iff*

$$\mathcal{D} \models (\forall s).compliant_{N_1}(s) \supset compliant_{N_2}(s).$$

As expected, two sets of norms N_1, N_2 are equivalent if they coincide, in all situations, in designating the same actions as forbidden or obligatory. This is established formally as follows.

Proposition 1. *Let N_1 be the norms $\Phi^1_F(a, s) \rightarrow \mathbf{F}\ a$ and $\Phi^1_O(a, s) \rightarrow \mathbf{F}\ a$ and N_2 be the norms $\Phi^2_F(a, s) \rightarrow \mathbf{F}\ a$ and $\Phi^2_O(a, s) \rightarrow \mathbf{F}\ a$.*

Then N_1 and N_2 are equivalent (wrt $\mathcal{D}$) iff

$$\mathcal{D} \models (\forall a, s).[\Phi^1_F(a, s) \equiv \Phi^2_F(a, s)] \ \wedge \ [\Phi^1_O(a, s) \equiv \Phi^2_O(a, s)].$$

Another problem of interest is checking whether a set of norms is *consistent* in some sense. Perhaps the simplest notion of consistency would be to define it as existence of compliant sequences. That is, a set of norms N is *inconsistent* iff

$$\mathcal{D} \models (\forall s).S_0 \sqsubset s \supset \neg compliant_N(s). \tag{2}$$

But this is probably too strong to be very useful. Perhaps a more useful notion would be one requiring only those sequences that are actually physically possible for the agent, to be compliant. This is expressed as follows:

$$\mathcal{D} \models (\forall s).[S_0 \sqsubset s \wedge executable(s)] \supset \neg compliant_N(s)$$

where $executable(s)$ is defined in terms of $Poss$ as described in Section 2.

Another possibly useful notion is that of consistency with respect to an agent's goal, as denoted by a formula $Goal(s)$. In this case we might say that a set of norms N is *inconsistent with respect to goal* $Goal(s)$ iff

$$\mathcal{D} \models (\exists s)Goal(s) \ \wedge \ (\forall s)[Goal(s) \supset \neg compliant_N(s)].$$

Intuitively, this says that a set of norms is inconsistent with respect to a goal if the goal is achievable but it is not possible to achieve it and satisfy the norms at the same time.

Other properties can be defined in a similar fashion in terms of logical entailment from a background theory $\mathcal{D}$. For example, a notion of two sets of norms being *equivalent with respect to a goal*, etc.

3.2 Ought-to-Be Norms

Ought-to-be norms specify situations in which a state condition, instead of an action as in ought-to-do norms, is forbidden or obligatory. We write such laws in the following form: $\Phi(\boldsymbol{t}) \ \rightarrow \ \mathbf{F}\ \Psi(\boldsymbol{v})$.

For example: when a user is logged out, no processes owned by the user should be executing:

$$loggedOut(usr) \wedge owner(usr, proc) \ \rightarrow \ \mathbf{F}\ executing(proc)$$

Since negation can appear in front of $\Psi(\boldsymbol{t})$, we do not use the obligation symbol $\mathbf{O}$ in these norms. Also, we understand these laws as dynamic, not static laws. That is, the condition $\Phi(\boldsymbol{t})$ is intended to be evaluated in the "current" state and the condition $\Psi(\boldsymbol{v})$ evaluated in the next state.

These norms cannot be put together into a normal form as in the case of ought-to-do norms, so we simply put them together as a conjunction of implications relative to a situation and its predecessor situation.

Compliance of a sequence is then defined as follows:

$$\begin{aligned} compliant(s) \equiv\ & s = S_0 \ \vee \\ & (\exists a, s').s = do(a, s') \wedge compliant(s') \ \wedge \\ & \textstyle\bigwedge_{i=1,\ldots,n} (\forall)[\Phi_i(\boldsymbol{t}_i, s') \supset \neg\Psi_i(\boldsymbol{v}_i, s)]. \end{aligned}$$

Having defined $compliant(s)$ this way for ought-to-be norms, the formal definitions of program compliance and norm equivalence, subsumption and consistency are exactly as for ought-to-do norms.

3.3 Deadlines

Consider an abstract form of deadline specifying that some condition ψ is forbidden before condition φ, written as $\mathbf{F}\ \psi \prec \varphi$ (deadlines like this are discussed in [15]). Norms involving deadlines would then take the form (to simplify the presentation, we assume all formulae have the same terms $\boldsymbol{x}$ as arguments):

$$\phi(\boldsymbol{x}) \;\rightarrow\; \mathbf{F}\; \psi(\boldsymbol{x}) \prec \varphi(\boldsymbol{x}). \tag{3}$$

For example, first-year students are not allowed to registered before the session starts:

$$firstyear(st) \;\rightarrow\; \mathbf{F}\; registered(st) \prec sessionstart.$$

Not surprisingly, compliance with a deadline is slightly more involved since it imposes conditions over a full sequence of actions. For the sake of clarity we will define it only for a single deadline of the form (3). The definition basically says that a sequence complies with a deadline if after any state where ϕ holds, either ψ never holds afterwards or φ holds at some point and ψ does not hold before that.

$$\begin{array}{l} compliant(s) \equiv \\ \qquad\qquad\qquad (\forall)(\forall s_1).[s_1 \sqsubset s \wedge \phi(\boldsymbol{x}, s_1)] \supset \\ \left\{ \begin{array}{l} (\forall s_2)[s_1 \sqsubset s_2 \sqsubseteq s \supset \neg\psi(\boldsymbol{x}, s_2)] \vee \\ (\exists s_2).s_1 \sqsubset s_2 \sqsubseteq s \wedge \varphi(\boldsymbol{x}, s_2) \wedge (\forall s_3).[s_1 \sqsubset s_3 \sqsubseteq s_2] \supset \neg\psi(\boldsymbol{x}, s_3) \end{array} \right\} \end{array}$$

As before, the definitions of program compliance and norm equivalence, subsumption and consistency apply to deadlines using the above definition of *compliance*(s).

4 Internalizing Norms

The straight forward way of enforcing a set of norms on an agent is to check that the norms are satisfied every time the agent chooses an action to execute next. That is, when an agent considers whether or not to execute an action α in situation σ, in addition to checking that α is physically possible, it would also check whether $compliant(do(\alpha, \sigma))$ holds.

An alternative way of enforcing the norms is to incorporate the norms into the agent's dynamic world description $\mathcal{D}$. Once the agent has "internalized" the norms into $\mathcal{D}$, it would behave in a norm compliant way.

One advantage of internalizing the norms is that, especially in the case of deadlines, compliance becomes a local check involving only the current state and the action being considered. There is no need to check conditions in the resulting state after executing an action nor on past states as required by the definition of compliance for deadlines. In implementations that update the belief base after each action is performed, internalizing the norms has the advantage that computing the tentative new state for each considered action, in order to check norm compliance, becomes unnecessary. And of course there is no need to store past states either.

4.1 Ought-to-Do Norms

Since we take norms as constraints on the behavior of an agent, their practical effect is to render some actions non-executable. The natural way then to incorporate a set of norms into the agent's domain description is in the form a additional preconditions in the APAs.

Consider the APA of an action type $A(\boldsymbol{x})$:

$$Poss(A(\boldsymbol{x}), s) \equiv \Pi_A(x, s).$$

The simplest way to incorporate a set of ought-to-do norms is by adding conditions saying a) $A(\boldsymbol{x})$ is not one of the forbidden actions, and b) if there is any obligatory action at all, it is $A(\boldsymbol{x})$. Assuming the set of norms are in the forms $\Phi_F(a) \rightarrow \mathbf{F}\ a$ and $\Phi_O(a) \rightarrow \mathbf{O}\ a$, we obtain the following modified APA for $A(\boldsymbol{x})$:

$$\begin{aligned} Poss(A(\boldsymbol{x}), s) \equiv\ & \Pi_A(x, s)\ \wedge \\ & (\forall a)[\Phi_F(a, s) \supset a \neq A(\boldsymbol{x})]\ \wedge \\ & (\forall a)[\Phi_O(a, s) \supset a = A(\boldsymbol{x})]. \end{aligned}$$

The additional preconditions added to each APA are the same except for the $A(\boldsymbol{x})$ term appearing in the consequent of the implications.

A more "efficient" way of incorporating the norms into the APAs, however, follows from the observation that each APA describes the preconditions of one specific action type $A(\boldsymbol{x})$. So any laws that forbid other actions are in fact irrelevant with respect to $A(\boldsymbol{x})$.

Taking then only those norms that forbid $A(\boldsymbol{x})$, if any, put in the form $\Phi_A(\boldsymbol{x}) \rightarrow \mathbf{F}\ A(\boldsymbol{x})$, as derived in Section 3, we modify the APA for $A(\boldsymbol{x})$ as follows:

$$\begin{aligned} Poss(A(\boldsymbol{x}), s) \equiv\ & \Pi_A(x, s)\ \wedge \\ & \neg\Phi_A(\boldsymbol{x}, s)\ \wedge \\ & (\forall a)[\Phi_O(a, s) \supset a = A(\boldsymbol{x})]. \end{aligned}$$

In this case the additional precondition $\neg\Phi_A(\boldsymbol{x}, s)$ is specific to the action type $A(\boldsymbol{x})$ so it varies with each APA.

Once the norms have been "compiled" into the underlying action theory $\mathcal{D}$, all programs will be compliant with the norms. Formally, let $\mathcal{D}$ be the background theory of an agent, N be a set of norms and $\mathcal{D}_N$ be the theory that results from applying the above transformation to $\mathcal{D}$ with respect to N.

Proposition 2. *For every program δ, $\mathcal{D}_N \models (\forall s).Do(\delta, S_0, s) \supset compliant_N(s)$.*

In other words, a sequence of actions is now considered to be a "legal" execution trace of the program δ only if it satisfies the norms.

Note that if N is inconsistent, e.g. as in (2), then δ has no legal execution traces, i.e. $\neg(\exists s)Do(\delta, S_0, s)$, and the implication in Prop. 2 is vacuously satisfied. If one does not want to count such a program as compliant, one can simply modify Def. 1 by adding the condition for weak compliance as a conjunct and define δ as compliant if it has at least one legal execution trace.

4.2 Ought-to-Be Norms

In the case of an ought-to-be norm of the form $\Phi(\boldsymbol{t}) \rightarrow \mathbf{F}\ \Psi(\boldsymbol{v})$, when the agent is considering an action α in a situation σ, it needs to check $\Phi(\boldsymbol{t})$ with respect to σ and $\Psi(\boldsymbol{v})$ with respect to $do(\alpha, \sigma)$. However, for the reasons mentioned earlier, we want to incorporate norms in terms of conditions on the current situation and the action under consideration. Moreover, according to the definition of APAs, the only term of sort situation allowed to appear in the formula on the right-hand-side of an APA is the variable s.

Fortunately, there is a mechanism that will allow us to compute a precondition relative to situation s from a condition relative to situation $do(A(\boldsymbol{x}), s)$. This mechanism is the *regression operator* $\mathcal{R}$ from [16,13]. Roughly, this operator takes a formula $\Gamma(do([\alpha_1, \ldots, \alpha_n], S_0))$ relative to a sequence of actions $do([\alpha_1, \ldots, \alpha_n], S_0))$ and computes a formula $\Gamma'(S_0)$ relative to S_0 that is equivalent to the original formula with respect to the background theory $\mathcal{D}$. The computation is purely syntactic and works by iteratively replacing each occurrence of a fluent $F(\boldsymbol{t}, do(\alpha, \sigma))$ with the formula $\Phi_F(\boldsymbol{t}, \alpha, \sigma)$ given a corresponding SSA $F(\boldsymbol{x}, do(a, s)) \equiv \Phi_F(\boldsymbol{x}, a, s)$. For details on operator $\mathcal{R}$, please refer to [16,13].

Consider any action type $A(\boldsymbol{x})$ and its corresponding APA

$$Poss(A(\boldsymbol{x}), s) \equiv \Pi_A(\boldsymbol{x}, s)$$

Let us now describe a procedure for incorporating a norm $\Phi(\boldsymbol{v}) \rightarrow \mathbf{F}\ \Psi(\boldsymbol{v})$ as additional preconditions.

1. Restore s as the situation argument in the premise: $\Phi(\boldsymbol{t}, s)$.
2. Restore $do(A(\boldsymbol{x}), s)$ as the situation argument in $\Psi(\boldsymbol{v})$ to obtain: $\Psi(\boldsymbol{v}, do(A(\boldsymbol{x}), s))$
3. Apply one regression step to the formula $\Psi(\boldsymbol{v}, do(A(\boldsymbol{x}), s))$ to obtain a formula $\Psi_A(\boldsymbol{v}, \boldsymbol{x}, s)$ relative to s, that is, let
$$\Psi_A(\boldsymbol{v}, \boldsymbol{x}, s) = \mathcal{R}^1[\Psi(\boldsymbol{v}, do(A(\boldsymbol{x}), s))].$$
4. Take the formulae from steps 1 and 3 and put them together in an implication as follows:
$$\Phi(\boldsymbol{t}, s) \supset \neg\Psi_A(\boldsymbol{v}, \boldsymbol{x}, s)$$
5. Finally, include the universal closure of the implication as an additional precondition in the APA for action $A(\boldsymbol{x})$:
$$Poss(A(\boldsymbol{x}), s) \equiv \Pi_A(\boldsymbol{x}, s) \wedge\ (\forall)[\Phi(\boldsymbol{t}, s) \supset \neg\Psi_A(\boldsymbol{v}, \boldsymbol{x}, s)].$$

The right-hand-side of the modified APA mentions only one situation term, s, as required. Note also that the subformula $\Psi_A(\boldsymbol{v}, \boldsymbol{x}, s)$ obtained by regression is specific to the action type $A(\boldsymbol{x})$. This is important because in many cases the result is that the subformula can be substantially simplified, as illustrated in the examples below. In fact, when a norm is completely irrelevant to a particular action type, the subformula frequently simplifies into a formula which is clearly valid and can be removed altogether.

As an example, consider a robot that lives in a university classroom building and has the norm

$$\mathit{lecture}(rm) \;\rightarrow\; \mathbf{F}\ at(rm).$$

saying that if there is a lecture occurring in a room, it should not be there. Suppose that it has actions $\mathit{enter}(rm)$ for entering a room and the action wait, as a representative of other actions that are not relevant to the location of the robot. The latter actions are interesting to consider since on the surface they appear irrelevant to the law. Let the corresponding APAs be:

$$\mathit{Poss}(\mathit{enter}(x), s) \equiv \neg at(x, s) \wedge \mathit{nextto}(\mathit{door}(x), s).$$

$$\mathit{Poss}(\mathit{wait}, s) \equiv \mathit{True}.$$

Let the SSA of fluent $at(x, s)$ be as follows:

$$at(x, do(a, s)) \equiv a = \mathit{enter}(x) \vee at(x, s) \wedge \neg(\exists y) a = \mathit{enter}(y).$$

Let us apply the above procedure to incorporate the law into the APAs. Starting with action $\mathit{enter}(x)$, we compute the regression of $at(rm, do(\mathit{enter}(x), s))$:

$$\begin{aligned}&\mathcal{R}[at(rm, do(\mathit{enter}(x), s))] \;=\\ &\qquad \mathit{enter}(x) = \mathit{enter}(rm) \vee at(rm, s) \wedge \neg(\exists y)\mathit{enter}(x) = \mathit{enter}(y).\end{aligned}$$

Since $\neg(\exists y)\mathit{enter}(x) = \mathit{enter}(y)$ is unsatisfiable, the resulting formula can be simplified to $\mathit{enter}(x) = \mathit{enter}(rm)$. By UNA on actions, this can be further simplified to $x = rm$. Thus the procedure yields the implication $(\forall rm)[\mathit{lecture}(rm, s) \supset x \neq rm]$ which can be further simplified to $\neg \mathit{lecture}(x, s)$. Adding this to the preconditions of $\mathit{enter}(x)$ we obtain the following APA:

$$\mathit{Poss}(\mathit{enter}(x), s) \equiv \neg at(x, s) \wedge \mathit{nextto}(\mathit{door}(x), s) \;\wedge \neg \mathit{lecture}(x, s).$$

Intuitively, from the general norm we have obtain the additional precondition specific to the enter action saying that there should not be a lecture in progress in the room to be entered.

Consider now the wait action. Following the procedure, we compute the regression of $at(rm, do(\mathit{wait}, s))$:

$$\begin{aligned}&\mathcal{R}[at(rm, do(\mathit{wait}, s))] \;=\\ &\qquad \mathit{wait} = \mathit{enter}(rm) \vee at(rm, s) \wedge \neg(\exists y)\mathit{wait} = \mathit{enter}(y).\end{aligned}$$

By UNA on actions, $\mathit{wait} \neq \mathit{enter}(y)$ for any y, so the resulting formula can be simplified to $at(rm, s)$. So the procedure yields the implication $\mathit{lecture}(rm, s) \supset \neg at(rm, s)$, and thus we obtain the following APA for the action wait:

$$\mathit{Poss}(\mathit{wait}, s) \equiv \mathit{lecture}(rm, s) \supset \neg at(rm, s)$$

which intuitively says that the agent can *wait* as long as it is not in a room where there is a lecture in progress. Exactly the same result would be obtained for similar actions that are not relevant to the location of the agent, such as $paint(obj)$, $pickup(book)$, etc. In other words, if the agent happens to be in a room where a lecture has started, the norm imposes the obligation on the agent to take immediate action to change location, by rendering all other actions impossible.

As in the case of ought-to-do norms, the above procedure yields an action theory $\mathcal{D}_N$ such that executing a program will now always result in a norm compliant sequence of actions, as formally stated in Proposition 2.

4.3 Deadlines

Let us finally look at how we might incorporate deadlines into an agent's background theory. As for the other types of norm, the aim is to extract additional preconditions from deadlines and add them to the APAs. The complication in the case of deadlines is that they are not local but may refer to situations arbitrarily far from the current one. In order to access those situations while satisfying the requirement of basic action theories that axioms only refer to the current situation, we employ a technique that consists in adding a small number of auxiliary fluents to keep track of whether certain conditions have been satisfied in a previous situation. This approach has been employed before in other contexts such as DB integrity constraints, automated planning and evolving knowledge bases [17,18,19,20,21,22].

For a deadline of the form $\phi(\boldsymbol{z}) \rightarrow \mathbf{F}\ \psi(\boldsymbol{z}) \prec \varphi(\boldsymbol{z})$ it suffices to add one auxiliary fluent $F_\phi(\boldsymbol{z}, s)$ with the following corresponding SSA:

$$F_\phi(\boldsymbol{z}, do(a,s)) \equiv [\phi(\boldsymbol{z}, do(a,s)) \vee F_\phi(\boldsymbol{z}, s)] \wedge \neg\varphi(\boldsymbol{z}, do(a,s)).$$

Intuitively, $F_\phi(\boldsymbol{z}, s)$ holds in s if the deadline is active in s.

The above axiom is actually not yet in the required form of an SSA because of the subformulae on the right-hand-side that have argument $do(a,s)$. This needs to be fixed by applying one regression step using operator $\mathcal{R}$ on those subformulae.

Having applying this procedure to a deadline, we add the corresponding auxiliary fluent F_ϕ and its SSA to the background theory. Then we modify the APA of each action type $A(\boldsymbol{x})$ by adding an additional precondition as follows:

$$\begin{aligned} Poss(A(\boldsymbol{x}), s) \equiv\ & \Pi_A(\boldsymbol{x}, s)\ \wedge \\ & (\forall \boldsymbol{z}) F_\phi(\boldsymbol{z}, s) \supset [\neg\psi(\boldsymbol{z}, do(A(\boldsymbol{x}), s)) \vee \varphi(\boldsymbol{z}, do(A(\boldsymbol{x}), s))] \end{aligned}$$

Note that this again requires applying regression in order to obtain a legit APA. Moreover, since regression is applied for the specific action type $A(\boldsymbol{x})$, it is possible that the resulting formulae can be substantially simplified as was the case with the ought-to-be norms. The resulting theory $\mathcal{D}_N$ yields a result similar to that in Proposition 2.

5 Related Work

In addition to work already mentioned, we discuss here some other related work.

Governatori and Rotolo [23,24] present a logical language called *Process Compliance Language* (PCL) which has deontic operators of several kinds: *punctual obligation*, *maintenance obligation* and *achievement obligation*. In [23], an algorithm for checking compliance of a business process is given. The algorithm takes an execution trace of the process, described by a Petri Net, and checks if norms are satisfied. While the norms are formalized in PCL, the process is described as a Petri Net and the algorithm is extra-logical. This differs from our approach where the norms, the process (program) and compliance is all expressed in the same logical language. Another difference is that PCL is a much richer norm language. In [24], they show that in addition to expressing norms, PCL can be used to describe the control flow of business processes using the deontic features of PCL. Since PCL specifications can be executed by a rule engine, expressing business processes in terms of PCL allows for the execution of business processes under PCL norms on the same rule engine. Similarly, our work is based on expressing both processes (Golog programs) and norms in the same language (the Situation Calculus), which is an advantage over approaches using different formalisms for each task. Also related is earlier work by this group on business processes and business contracts [25,26].

In [27], Meneguzzi and Luck present an approach to adjusting the behavior of an agent to newly accepted norms. Similar to ours, the approach is based on modifying the agent's behavior by changing the implementation of such behavior. The main difference with our approach is that they apply the modifications to the programs in the plan library, e.g. by removing plans that include a forbidden action. These plans are then restored when the corresponding norm expires and the action is no longer forbidden. In our case, the modification is done in the underlying primitive action theory, not at the plan library level. This has several advantages: 1) If a program is non-deterministic, a norm may make some execution traces illegal while others remain legal. In our case, after modifying the underlying action theory, executing the program would result only in legal execution traces but the program remains the same. In their approach, the only choice seems to be to remove the program altogether. 2) Whether an action has a forbidden effect or not may depend on the state where the action is executed, and in turn the same applies for an agent program. In our approach, the agent would still be able to execute a program in a situation that does not lead to a forbidden state, even if the plan would violate a norm if executed in another context. Again, in their case the only choice is to remove the program altogether. On the other hand, they consider the interesting case of accepting new norms at run-time, which requires the ability to abandon programs already in execution or in the intention base.

The recent work of van Riemsdijk et al. [28] also deals with norms in multi-agent systems. Their main concern is to formalize norms present in the MOISE$^+$

organizational modeling language [29]. The work is complementary to ours since we consider how to ensure an agent complies with a set of accepted norms, without considering the source of the norms, which could very well be an organization such as those modeled in MOISE$^+$. The approach in [28] is to formalize norms in LTL, which has a close correspondence to the language used here. This means those norms would not be too difficult to integrate with our approach.

There is also a large amount of related work on verifying that an agent conforms to an interaction protocol. We discuss some of that work next.

In [30], Endriss et al. consider the problem of checking that communicating agents conform to a public dialog protocol. The approach is based on abductive logic programming and protocols are formalized as integrity constraints which "filter out" illegal dialog moves. This allows a simple way to enforce a protocol: add the protocol to the agent's knowledge base. The analogous way to enforce prohibitions in our framework would be to modify the definition of the relation $Do(\delta, s', s)$ to include $compliant(s)$ as a condition on s. This would be obviously correct. But contrast that with our proposed procedure for internalizing norms: in the former approach, *all* the norms have to be checked for *every* action the agent intends to execute next. In the latter approach, only those norms which are actually relevant to a particular action need to be checked, and in a simplified form. In a sense, the process of internalizing the norms computes what norms are relevant and simplifies them for each action.

Baldoni et al. [31,32] consider a-priori verification of conformance of an agent with a role in an interaction protocol. The main concern there is to guarantee interoperability: that the system will function correctly provided agents conform to their roles in the protocol. The approach is base on finite state automata.

Singh et al. [33,34,35] also look at the problem of conformance with an interaction protocol, but their approach is based on *commitments*. In [33], compliance with commitments is reduced to model checking CTL formulae against a model of the agents' interactions. Thus it roughly corresponds in our framework to checking $compliant(s)$ on an execution trace s. In [35], using a notion of *run subsumption*, an agent is said to conform to a protocol if the execution traces of its program are subsumed by the execution traces of the protocol. While we do not consider it here, run subsumption conformance is similar to norm subsumption (see Def. 3) so it seems it would not be too difficult to define a similar notion.

Finally, Chesani et al. [36] formalize a form of run-time conformance checking based on commitments using a reactive version of the Event Calculus. The approach allows "full" and "partial" violations where the latter allow the agent to fulfil a commitment after the deadline by paying a penalty.

Approaches to protocol conformance are mainly concerned with guaranteeing global interoperability and hence take an external view of agents. This is complementary to our work here where the problem is to ensure compliance with a set of norms by incorporating them into the agent.

6 Conclusions

In this work we have considered how to express several types of norms, namely ought-to-do, ought-to-be and a form of deadline, and how to incorporate them into the framework of Golog. We define a notion of a sequence of agent actions complying with a set of norms and a formal definition of an agent's program complying with the norms. We also describe notions of equivalence between norm systems with respect to an agent's background theory in the Situation Calculus, as well as notions of norm system subsumption and consistency. We have also shown procedures for incorporating a set of norms into the formalization of the primitive actions of an agent so that after the norms have been thus internalized, the agent is guaranteed to behave in a norm compliant manner.

This is a first approach at the problem of regulating the behavior of a Golog agent using a set of norms, so we make many strong simplifying assumptions. For example, we assume that the agent has accepted the norms and will not violate them. This should be extended to allow the agent to violate some of the norms if desirable, and perhaps provide also a penalty mechanism. We would also like to look at more complex forms of deadline involving explicit time, especially since there is already a temporal extension of Golog [37]. Further work is also necessary on the multi-agent aspect of this work.

References

1. Shoham, Y., Tennenholtz, M.: On social laws for artificial agent societies: Off-line design. Artificial Intelligence 73(1-2), 231–252 (1995)
2. Dastani, M., Grossi, D., Meyer, J.J.C., Tinnemeier, N.A.M.: Normative multi-agent programs and their logics. In: Meyer, J.-J.C., Broersen, J. (eds.) KRAMAS 2008. LNCS, vol. 5605, pp. 16–31. Springer, Heidelberg (2009)
3. Boella, G., van der Torre, L.W.N.: Regulative and constitutive norms in normative multiagent systems. In: Dubois, D., Welty, C.A., Williams, M.A. (eds.) Ninth International Conference on Principles of Knowledge Representation and Reasoning, pp. 255–266 (2004)
4. Sergot, M.: Norms, action and agency in multi-agent systems. In: Governatori, G., Sartor, G. (eds.) DEON 2010. LNCS, vol. 6181, pp. 2–2. Springer, Heidelberg (2010)
5. Fitoussi, D., Tennenholtz, M.: Choosing social laws for multi-agent systems: Minimality and simplicity. Artificial Intelligence 119(1-2), 61–101 (2000)
6. Craven, R., Sergot, M.J.: Agent strands in the action language nC+. Journal of Applied Logic 6(2), 172–191 (2008)
7. Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: A logic programming language for dynamic domains. Journal of Logic Programming 31(1-3), 59–83 (1997)
8. McCarthy, J.: Situations, actions and causal laws. Technical report, Stanford University (1963); Reprinted in Semantic Information Processing (M. Minsky ed.), pp. 410–417. MIT Press, Cambridge (1968)
9. De Giacomo, G., Lesperance, Y., Levesque, H.: ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121, 109–169 (2000)

10. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, pp. 355–362 (2000)
11. De Giacomo, G., Levesque, H.J.: An incremental interpreter for high-level programs with sensing. In: Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter, pp. 86–102. Springer, Heidelberg (1999)
12. Reiter, R.: The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In: Lifschitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation, pp. 359–380. Academic Press, London (1991)
13. Reiter, R.: Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
14. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 4, pp. 463–502. Edinburgh University Press (1969); Also appears in Nilsson, N., Webber, B.(eds.) Readings in Artificial Intelligence. Morgan-Kaufmann, San Francisco
15. Dignum, F., Broersen, J., Dignum, V., Meyer, J.J.C.: Meeting the deadline: Why, when and how. In: Hinchey, M.G., Rash, J.L., Truszkowski, W.F., Rouff, C. (eds.) FAABS 2004. LNCS (LNAI), vol. 3228, pp. 30–40. Springer, Heidelberg (2004)
16. Pirri, F., Reiter, R.: Some contributions to the metatheory of the Situation Calculus. Journal of the ACM 46(3), 325–364 (1999)
17. Chomicki, J.: Efficient checking of temporal integrity constraints using bounded history encoding. ACM Transactions on Database Systems 20(2), 148–186 (1995)
18. Gabaldon, A.: Compiling control knowledge into preconditions for planning in the situation calculus. In: Gottlob, G., Walsh, T. (eds.) 18th International Joint Conference on Artificial Intelligence (IJCAI 2003), pp. 1061–1066 (2003)
19. Gabaldon, A.: Precondition control and the progression algorithm. In: Dubois, D., Welty, C., Williams, M.A. (eds.) 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), pp. 634–643 (2004)
20. Bienvenu, M., Fritz, C., McIlraith, S.A.: Planning with qualitative temporal preferences. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Tenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 134–144 (2006)
21. Alferes, J.J., Gabaldon, A., Leite, J.A.: Evolving logic programming based agents with temporal operators. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2008), pp. 238–244. IEEE, Los Alamitos (2008)
22. Alferes, J.J., Gabaldon, A., Leite, J.A.: Evolving logic programs with temporal operators. In: Balduccini, M., Son, T. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS (LNAI), vol. 6565, pp. 193–212. Springer, Heidelberg (2011)
23. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: Link, S., Ghose, A. (eds.) 7th Asia-Pacific Conference on Conceptual Modelling (APCCM 2010), vol. 110, pp. 3–12 (2010)
24. Governatori, G., Rotolo, A.: Norm compliance in business process modeling. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 194–209. Springer, Heidelberg (2010)

25. Padmanabhan, V., Governatori, G., Sadiq, S.W., Colomb, R., Rotolo, A.: Process modelling: the deontic way. In: Stumptner, M., Hartmann, S., Kiyoki, Y. (eds.) 3rd Asia-Pacific Conference on Conceptual Modelling (APCCM 2006), vol. 53, pp. 75–84 (2006)
26. Governatori, G., Milosevic, Z., Sadiq, S.W.: Compliance checking between business processes and business contracts. In: 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), pp. 221–232. IEEE Computer Society, Los Alamitos (2006)
27. Meneguzzi, F., Luck, M.: Norm-based behaviour modification in BDI agents. In: Sierra, C., Castelfranchi, C., Decker, K.S., Sichman, J.S. (eds.) 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 177–184 (2009)
28. van Riemsdijk, M.B., Hindriks, K.V., Jonker, C.M., Sierhuis, M.: Formalizing organizational constraints: a semantic approach. In: van der Hoek, W., Kaminka, G.A., Lespérance, Y., Luck, M., Sen, S. (eds.) 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), pp. 823–830 (2010)
29. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. International Journal of AOSE 1(3-4), 370–395 (2007)
30. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Protocol conformance for logic-based agents. In: Gottlob, G., Walsh, T. (eds.) 18th International Joint Conference on Artificial Intelligence (IJCAI 2003), pp. 679–684 (2003)
31. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: A priori conformance verification for guaranteeing interoperability in open environments. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 339–351. Springer, Heidelberg (2006)
32. Baldoni, M., Baroglio, C., Chopra, A.K., Desai, N., Patti, V., Singh, M.P.: Choice, interoperability, and conformance in interaction protocols and service choreographies. In: Sierra, C., Castelfranchi, C., Decker, K.S., Sichman, J.S. (eds.) 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 843–850 (2009)
33. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)
34. Desai, N., Chopra, A.K., Singh, M.P.: Representing and reasoning about commitments in business processes. In: Holte, R.C., Howe, A. (eds.) 22nd AAAI Conference on Artificial Intelligence (AAAI 2007), pp. 1328–1333. AAAI Press, Menlo Park (2007)
35. Chopra, A.K., Singh, M.P.: Producing compliant interactions: Conformance, coverage, and interoperability. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS (LNAI), vol. 4327, pp. 1–15. Springer, Heidelberg (2006)
36. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus. In: Boutilier, C. (ed.) 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 91–96 (2009)
37. Reiter, R.: Sequential, temporal GOLOG. In: Cohn, A., Schubert, L. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference (KR 1998), pp. 547–556. Morgan Kaufmann, San Francisco (1998)