A Framework for Model Checking Institutions *

Francesco Viganò

Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
francesco.vigano@lu.unisi.ch

Abstract. To increase positive expectations in the outcome of open multiagent systems, institutions have been put forward to regulate agents' behaviour. To *model* and to *verify* such institutions, we propose to adopt the notion of status function, which provides for a unified approach to ontological and deontic aspects regulated by an institution. Also, to enhance the development of *functional* and *rational* institutions, we propose a language amenable to model checking to describe them and their properties. Finally, we present our tool and an evaluation of our approach.

1 Introduction

To increase positive expectations in the outcome of systems composed by autonomous agents, *norms* have been put forward to regulate and to make more predictable agents' behaviour [20, 8, 9, 24, 5]. While in [20] norms (also named *social laws*) are assumed to be respected by agents because they are designed and encoded by a single organization, in [8, 9, 24, 5] norms are *defined* by *institutions* and *enforced* by *organizations* to regulate *open* multiagent systems, where it is unrealistic to expect that agents implemented by different parties with interests in conflict will always behave according to a system of norms.

Our research is motivated on one hand by the fact that deontic aspects and institutional facts need a unified treatment, while so far they have been regarded separately [8, 5, 9, 10, 22]. Following [23], in [27] we have proposed to model institutions in terms of a single concept, that is, *status function*, which provides for a unified approach to ontological and deontic aspects regulated by an institution. Roughly speaking, status functions are positions imposed on agents to perform certain actions (institutional actions), which cannot be executed without the recognition of their effects by a community of agents (see [27] for more details). In fact, the effects of such actions essentially consist in the imposition or revocation of other status functions, which may confer to agents new authorizations or obligations to perform other institutional actions. For instance, in an English auction, when the auctioneer declares a new current price, participants are authorized only to make higher bids. For this reason, we think that our approach better clarify the interdependence existing among agents' deontic positions (authorizations, obligations, etc.) and the ontology of the context in which agents interact, while in [8, 5, 9, 10, 22] such relation is ignored.

* Supported by Swiss National Science Foundation project 200020-109525, "Artificial Institutions: specification and verification of open distributed interaction frameworks."

On the other hand, our research is motivated by the lack of formal support for the verification of institutions. For instance, as observed in [25], certain aspects of the language proposed in [8] to model institutions lack of a formal semantics. For example, the meaning of labels used in [8] "is not clear from the notation only" [25]. This fact has negatively influenced the application of automated techniques to verify such formalism: in [4] and [13] the authors propose two frameworks to model check electronic institutions, but, due to the aforementioned limitations, only properties regarding their syntactic structure can be defined and verified. As a consequence, ISLANDER [7], a tool for modelling institutions according to the formalism introduced in [8], only checks syntactical aspects and does not integrate any support for the automatic verification of electronic institutions. Also, in [22], where agent societies are described with a formalism inspired by the Event-Calculus [15], the authors must rely on "systematic runs" to guarantee the correctness of their protocol. Given that institutions are introduced to increase the reliability of agents interactions by ensuring that agents have at their disposal all the needed powers to fulfill their objectives and that they will not be subject to contradictory or undesired norms, the aid of automated techniques it is necessary to foresee all possible evolutions and states in which an institution may evolve.

To allow designers to describe institutions in terms of status functions and to offer them the support of a tool that provides for their automatic verification, in [27] we have introduced FIEVeL (*F*unctions for *I*nstitutionalized *E*nviron-ments *Ve*rification *L*anguage), which is an institution definition language amenable to model checking [2]. In this paper we extend FIEVeL to allow a designer to express desirable properties that should be satisfied by his or her models in terms of the concepts defined in our metamodel [27], ignoring the actual translation of FIEVeL into the input language of the model checker.

The remainder of this paper is structured as follows. In Section 2 we briefly resume the main legal and philosophical concepts that we perceive as essential to specify institutions. In particular, since institutions have been applied to define protocols [9, 22], we will ground our discussion with examples taken by a widely used interaction protocol, the English auction. Section 3 introduces certain critical aspects of the translation of FIEVeL models into Promela, the input language of SPIN [12]. In Section 4 we present how we have extended FIEVeL to specify domain-dependent properties, while Section 5 discusses an initial library of domain-independent properties. A key aspect of domain-dependent properties is that they can be automatically translated, while an important aspect of domain-independent properties is that they can be automatically generated by our tool, which is briefly presented and evaluated in Section 6. Finally, in Section 7 we provide a comparison of our approach with related works.

2 Modelling Institutions

The key concept that characterizes our approach to the description of an institution is the notion of *agent status function*, that is, a status imposed on an agent and recognized as existing by a set of agents [23]. Typical examples of status functions are not only the concept of auctioneer or winner of an auction, but also being the owner of a good, being the husband or the wife of somebody. Although there exist several similarities among the concepts of status function and role as it has been discussed in the literature [17], we prefer to use the term status function because it better represents the fact that we are concerned with statuses assigned to agents to perform several functions and whose existence depends on those agents that recognize them as existing.

Status functions are defined as possibly empty aggregates of *deontic positions* that can be expressed in terms of two main concepts, *authorizations* (also named *institutionalized power* [14, 22]) and *obligations*. An obligation is characterized by certain expressions which are used to specify conditional obligations and when an obligation should be considered fulfilled or violated (more details can be found in [27]). An obligation is created because a status function is imposed, changes its state when its conditions are satisfied, and eventually reaches a final state (*inactive*) either because its expressions are evaluated to true or because it is associated to a revoked status function. In particular, when an obligation reaches the inactive state, we consider that it has been *fulfilled* if its fulfillment expression is satisfied, otherwise if the violation expression is satisfied we say that the agent has violated it. Obligations can be also used to express prohibitions by specifying suitable violation expressions, while we do not define a specific construct to explicitly represent the fact that an agent is permitted to perform an action as in [5, 10, 22]. Instead, as in [9] we consider that every action, if it is not prohibited, is also permitted.

Interdependent status functions are declared within *institutional entities* which enforce on a group of status functions a set of *constraints* (e.g. an auctioneer cannot be also a participant). Moreover, institutional entities define cardinality constraints, like "an auction is composed by an auctioneer and a set of participants".

FIEVeL allows to model two kinds of events, *base-level events* and *institutional events*. There exists an ontological difference among them: while the former exist because they correspond to certain physical changes or are relative to lower level institutions, like *time* events and *message-exchange* events, the latter exist because they are recognized as existing by a community of agents. Therefore, institutional events are not directly produced by the environment or by an agent thanks to its own capabilities, because their effects need to be recognized by a set of agents. Instead, institutional events happen because agents accept that when certain base-level events occur, if certain contextual conditions are satisfied, they *count as* institutional events. As it can be noticed in Figure 1, FIEVeL models institutional events by describing their preconditions in terms of the existence or absence of certain status functions, while their effects are expressed by *assigning* (or *revoking*) several status functions.

In [27] we extended the treatment of the count-as relation as it was presented in [9] by describing what kind of contextual conditions are relevant to model that a base-level event counts as an institutional event. In particular, we single out the following conditions: (i) there must exist a convention binding the base-level event to the institutional event; (ii) a precondition, expressed in terms of the existence or absence of certain status functions, must be satisfied; (iii) in the case of institutional actions, the agent must be authorized to perform the institutional action. If all these conditions are satisfied, a base-level event counts as an institutional event and its effects take place, which means that certain status functions will be assigned or revoked.

FIEVeL models status functions, base-level events, and institutional events as complex types defined in terms of certain *basic types*, which can be specified as subsets of built-in basic types or by enumerating their elements (see Figure 1). FIEVeL defines few built-in basic types, like integers (*int*), agent identifiers (*AID*), and objects identifiers (*OID*).

Figure 1 reports a few fragments of our specification of the English Auction institution, showing that an auctioneer that has opened an auction (represented by INSESSION status function) is not only authorized to close the auction if certain conditions hold, but thereafter it is also obliged to do so before the next time instant has elapsed. An auctioneer is also authorized to open a new round if there are at least two participants by performing the institutional action *newRound*, which is conventionally bound to the exchange of a message of type *openRound* and, if the previous round has been closed with at least two offers, a new round is open and participants can make their bids.

3 Institutional Models and Computational Models

Every institution described with FIEVeL corresponds to an *ideal* transition system characterized by a *many sorted first-order signature* [18] where every transition represents all institutional effects (institutional events, fulfillment of obligations, etc) associated with a base-level event. Intuitively, every transition is caused by a base-level event, which may also count-as an institutional event if certain conditions are met, and, as a consequence, institutional reality is modified by the imposition and revocation of certain status functions. Also, since expressions that regulate the evolution of agents' obligations refer to both events and institutional states, certain obligations may update their state. Ideally, all these changes correspond to a single transition of the base-level and institutional transition system.

Actually, the generation of such transition system is not only computationally expensive, but also its encoding into Promela, the input language of the SPIN model checker [12], would also lead to a huge number of code lines. In fact, due to several limitations of SPIN, we should generate a transition for each possible combination of base-level events, institutional-events, and obligation-state changes. For this reason, instead of calculating the ideal transition system, we prefer to derive a *computational* transition system, such that each transition partially represents the institutional effects of an event, so that summing the effects of a sequence of transitions we can reach the same institutional state.

To demonstrate that we can build a computational transition system which simulates the ideal system, that is, satisfies the same properties, let M be a Kripke structure over a set AP of atomic propositions such that V is a valuation function associating a value in $\{0, 1\}$ at each atomic proposition p in AP for each state, Π is the set of all paths and Π_0 represents the set of all initialized paths in M. In the sequel we write π_k for the k-th state of path $\pi = s_0, s_1, s_2, \ldots$, π^k for the suffix of π starting at state π_k, that is, the sequence $s_k, s_{k+1}, s_{k+2}, \ldots$, and finally $M, \Pi_0 \models \varphi$ to mean that for each $\pi \in \Pi_0$, $M, \pi \models \varphi$.

Given two Kripke structures M and $\widehat{M}$, we define a relation $Z_{inst} \subseteq \Pi \times \widehat{\Pi}$ such that $(\pi, \widehat{\pi}) \in Z_{inst}$ if and only if for each proposition $p \in AP$:

```
basic-types:
   priceD subtype-of int;
     ...
base-events :
  message openRound();
     ...
institution EnglishAuctionInstitution {
  institutional-entity englishAuction {
  ...
    [0,ALL] status-function INSESSION() {
      authorizations:
      ...
       newRound n <- exists[>=2] p in PARTICIPANT [true];
      deontic-specification:
       obligation(done(open,subject),
           (exists [<2] x in PARTICIPANT [true] or
           done(newRound,subject)),activation-time>1);
      ...
    }// INSESSION
      ...
    constraints:
      AUCTIONEER disjoint PARTICIPANT;
  }// entity
  conventions:
      exch-Msg(openRound ()) =c=> newRound();
      time() =c=> endRound();
      ...
  institutional-events:
      ...
      institutional-action newRound():
      pre: ((not exists x in INROUND [true] and
              not exists o in OFFERED[true])
              and not exists x in ENDAUCTION [true]) ;
      eff: p in PARTICIPANT -X->
              assign(p.subject, INROUND())[true],
             a in AUCTIONEER -X->
              assign(a.subject, INROUND()[true];
      ...
}// institution
     ...
model-definition:
  basic-domains:
    AID={aid_0,aid_1,aid_2};
      ...
  initial-state:
    assign(aid_0,AUCTIONEER());
      ...
```

Fig. 1. Fragments of the English Auction institution.

1. $V(\pi_0, p) = \widehat{V}(\widehat{\pi}_0, p)$;
2. $\widehat{V}(\widehat{\pi}_0, inst) = 1$;
3. there exists a $k > 1$ such that:
   (a) $V(\pi_1, p) = \widehat{V}(\widehat{\pi}_k, p)$;
   (b) for every $0 < r < k$
      i. $V(\pi_0, p) = \widehat{V}(\widehat{\pi}_r, p)$;
      ii. $\widehat{V}(\widehat{\pi}_r, inst) = 0$;
   (c) $(\pi^1, \widehat{\pi}^k) \in Z$.

where $inst$ is an atomic proposition belonging to $\widehat{AP}$ but which does not appear in AP.

Intuitively, we can imagine building a path $\widehat{\pi}$ by taking path π, adding a set of atomic propositions which do not belong to AP, by marking every state of π such that the valuation function of $inst$ is *true*, and finally by inserting between two consecutive states (π_i, π_{i+1}) a new set of intermediate states which are characterized by the same valuation function of state π_i with respect to propositions AP and which valuate the proposition $inst$ to *false*. Path $\widehat{\pi}$ is therefore emulating the behavior of path π by simulating each transition on path π with a finite sequence of transitions, such that it keeps unchanged the truth values of common atomic propositions until it changes them in a single step.

Z_{inst} has been introduced to guarantee that an institution described with FIEVeL can be verified with SPIN and that every property regarding the institutional state holding in the ideal institution also holds in the Promela model, and vice versa. While in [27] we presented a relation which preserves all temporal properties of the ideal system which do not contain the *next* temporal operator, Z_{inst} preserves all temporal formulas which are satisfied in the institutional system. To obtain such result, we first introduce a transformation τ defined as follows:

$$\tau[p] = p$$
$$\tau[\neg\chi] = \neg\tau[\chi]$$
$$\tau[\chi \wedge \psi] = \tau[\chi] \wedge \tau[\psi]$$
$$\tau[\mathbf{X}\chi] = inst\mathbf{U}(\neg inst \wedge (\neg inst)\mathbf{U}(inst \wedge \tau[\chi]))$$
$$\tau[\chi\mathbf{U}\psi] = (inst \rightarrow \tau[\chi])\mathbf{U}(inst \wedge \tau[\psi])$$

where p represents any atomic proposition in AP, and χ and ψ represent any formula. It can be demonstrated [1] the following Lemma:

Lemma 1. *If $(\pi, \widehat{\pi}) \in Z_{inst}$ and φ is a temporal formula composed by atomic propositions belonging to AP, then $M, \pi \models \varphi$ if and only if $\widehat{M}, \widehat{\pi} \models \tau[\varphi]$.*

From Lemma 1 it follows the subsequent theorem:

Theorem 1. *If for each $\pi \in \Pi_0$ there exists a $\widehat{\pi} \in \widehat{\Pi}_0$ such that $(\pi, \widehat{\pi}) \in Z_{inst}$, and for each $\widehat{\pi} \in \widehat{\Pi}_0$ there exists a $\pi \in \Pi_0$ such that $(\pi, \widehat{\pi}) \in Z_{inst}$, for each temporal formula φ composed by atomic propositions belonging to AP, $M, \Pi_0 \models \varphi$ if and only if $\widehat{M}, \widehat{\Pi}_0 \models \tau[\varphi]$.*

[1] For proofs of Lemma 1 and Theorem 1 refer to [26].

This result means that given an *ideal* transition system which corresponds to a FIEVeL model, we can build a *computational* transition system which preserves all temporal properties of the *ideal* system.

Before briefly introducing how we proceed to translate FIEVeL institutions into Promela models, it is worth discussing how we model *time*. FIEVeL regards time aspects in two distinct ways: (i) as in classical temporal logic, to define *qualitative* properties (e.g. it is always the case that an auctioneer cannot win an auction), and (ii) as in RTTL [21] to express *quantitative* properties (e.g. the auctioneer must open the auction before two minutes since now). Two consecutive time events t_i and t_{i+1} may be separated by a sequence (possible empty) of other base level events, which are assumed to occur at time t_i. Hence the institutional state may change due, for instance, to message-exchange events even if the time does not change. In principle, there may be an infinite number of time ticks, while in [27] we considered only finite time intervals. The main drawback of the approach discussed in [27] is that certain formulas may (not) hold in a system because they refer to time instants that have not been considered by the model checker [19].

To solve this problem, we have slightly changed FIEVeL syntax by allowing only time expressions which assume a truth value that will not longer change after a fixed number of clock ticks have elapsed from the occurrence of a certain event. As a consequence, we have also changed our encoding of time references defined by an institution, which are substituted by suitable counters. In doing so, we are simulating an infinite number of time instants with a finite structure that is verifiable by a model checker. Therefore, we can ensure that a property holds in a Promela model if and only if it holds in the institution which is characterized by an infinite number of time tick. As a consequence, the number of possible events that the model checker should consider, and how they can be interleaved, has been increased, which means that we have increased the memory and time required to verify an institution.

To model check FIEVeL models we have defined an automatic translation of FIEVeL into Promela. A Promela model is composed by a set of *processes* and *global variables* that can be described by defining new *process types* and *record structures*. Essentially a Promela process is constituted by a set of statements (also named *guarded commands* in [6]), which can be *simple statements*, like assignments, or *compound statements*, like selection (`if`) and repetition (`do`). Each statement is characterized by an *enabling condition* and a *postcondition*. Promela imposes severe restrictions on what can be specified in a precondition, therefore, to overcome such limitations and to increase the expressiveness of FIEVeL without producing an huge number of intermediate states, we chose to use embedded C code to evaluate preconditions of transitions and to compute reachable states. The SPIN model checker adopts an interleaving semantics, which means that when several processes have executable statements, it randomly chooses one of them and executes it. When all enabling conditions are evaluated to false, two special preconditions (also named guards) `timeout` and `else`, are evaluated to *true*. In particular, `else` is enabled only if all transitions at the process level cannot be executed, while `timeout` is evaluated to true only if no process has an enabled transition. In this brief overview we have just introduced a few concepts that are necessary for the sake of the present discussion, while further details can be found in [12].

When we translate FIEVeL, status functions, institutional entities, obligations, base events, and institutional events are encoded as a set of type definitions, which are then used to declare two set of variables, one representing the *current institutional state* and another which is exploited to generate the *next institutional state*. Each institution is then translated into a new process definition according to the pattern represented in Figure 2. For sake of simplicity, we can imagine that every process representing an institution consists of a main loop which is enabled whenever a new base event is generated and contains an inner loop where each guard represents an institutional event or an obligation-state transition. Actually, to further reduce the number of intermediate transitions and to generate a more compact Promela code, our current implementation introduces several improvements that have not been reported in Figure 2.

```
proctype institutionProc(int id) {
  do
  :: (nextEvent.analyzed[id]==FALSE)->
    do
    :: ((condition_inst_event_x1 || condition_inst_event_x2) &&
         !(effects_x)) ->
           apply_effects(next_event);
    ...
    :: ((next_obligation.state==inactive) &&
        (start_obligation && obligation.state==inactive)->
         next_obligation.state=active;
    ::else -> break;
    od
    nextEvent.analyzed[id]=TRUE;
  od
}
    ...
proctype eventGenerator(){
  do
  :: timeout ->
    curr_state=next_state; inst=true;
    inst=false; updateConditions();
    if
    :: true             -> next_event = time_event;
    :: condition_event1 -> next_event = event1;
    :: condition_event2 -> next_event = event2;
    ...
    fi
  od
}
```

Fig. 2. Translation pattern for processes representing institutions and the structure of an event-Generator process.

As discussed above, institutional states evolve because base-level events happen and they count-as institutional events. To model base-level events, we define a new process, named *eventGenerator*, which generates actions and events as if they were produced by agents or the environment. Agents therefore are not modelled as processes as in [28], while we reduce them as parameters of base-level events. This choice is motivated by the fact that in open multiagent systems agents' internal states are not accessible and therefore we must assume an external point of view. The process structures presented in Figure 2 ensures that the *eventGenerator* is activated only when all institution instances have generated the next institutional state by considering all possible institutional transitions (counts-as, institutional-event effects, and obligation transitions), such that current state variables can be updated with the new calculated values. Therefore, when an *eventGenerator* is activated, we consider reached the next institutional state and thereafter we start to compute the following institutional state by choosing the next base-level event. In doing so, events are modelled as if they were perceived and analyzed by a centralized *institution manager*, which manages the state of the system and updates it when an event occurs. Although such an assumption would be unrealistic in the implementation of a distributed system, we introduce it to reduce the complexity of the verified model. Also, it can be noticed that several prototypes of institutions, for example [10], consider a single centralized component which makes the institutional state evolve.

Finally we declare an *init* process to generate the initial state. Due to values assigned by default to variables that compose complex types, we cannot always build a computational path $\widehat{\pi}$ whose valuation function at the initial state simulates the valuation function of the institutional path π at its initial state such that $(\pi, \widehat{\pi}) \in Z_{inst}$. To overcome this problem, we consider a computational path such that: (i) $V(inst, \widehat{\pi}_i) = 0$ for each $i < 2$ and (ii) $(\pi, \widehat{\pi}^2) \in Z_{inst}$. Therefore we obtain that $\pi \models \varphi$ if and only if $\widehat{\pi}^2 \models \tau[\varphi]$, which corresponds to check if $\widehat{\pi} \models \neg inst\mathbf{U}(inst \wedge \tau[\varphi])$.

4 Domain-Dependent Properties

Domain-dependent properties stem from peculiar features of the specified institution and regard its functionality: for instance, we may want to check if it is possible that a participant is declared to be the winner of an auction. In specifying domain-dependent properties we must take into account two different aspects: (i) since we are considering open multiagent systems, agents may violate norms of institutions and (ii) agents may be permitted and not obliged to perform certain actions (see Section 2). For instance, in our specification of the English auction, participants are permitted to make bids and an agent can be declared to be the winner only it has offered the highest price. Therefore, certain executions end without a winner because none of the agents has made a bid, while if the auctioneer respects its obligations, it is always the case that if there is at least an offer, there will be a winner.

As a consequence, agents' violations and permissions led us to consider not only *validity* of LTL formulae, but also their *satisfiability*. In the sequel we will write $M \models_A \varphi$ to mean that φ is valid in M and $M \models_E \varphi$ to express that φ is satisfiable in M. The set of LTL formulae can be seen as a subset of CTL* formulae of the form $\mathbf{A}f$, where $\mathbf{A}$ is the universal quantifier over paths and f is a formula composed by atomic propositions, boolean operators, and temporal operators [2]. For the relation existing among the universal and the existential quantifiers, we can define satisfiability of formula φ as follows:

$$M \models_E \varphi \equiv M \not\models_A \neg\varphi$$

Therefore, since SPIN is able to check only validity of LTL formulae, to verify if $M \models_E \varphi$, we check whether $\neg\varphi$ is a valid formula, and if the model checker generates a counterexample we assume that φ is satisfiable. Despite our interest for the existence of certain paths, we still prefer LTL to CTL because it is simpler and it helps the designer to focus his or her attention on single runs of the system.

Domain-dependent properties can be specified in FIEVeL by combining temporal operators with expressions that refer to institutional states (expressed in terms of status functions) and events. Despite FIEVeL expressions correspond to plain LTL formulae, we think that it is worth specifying properties of institutions with FIEVeL for three main reasons: i) a FIEVeL formula represents an abbreviated form for a long and complex LTL formula, ii) FIEVeL guarantees syntactic type checking of formulae, and iii) designers can reason in terms of institutional concepts ignoring their translation into Promela models. Moreover, institutions describe rules that typically are independent of the number of agents, objects, etc. involved in the interaction, which can be naturally expressed in FIEVeL by using quantification over sorts. Temporal operators and FIEVeL expressions can be combined because we assume that domains are fixed at all states of our system. For the same reason, we admit that domain-dependent properties can be externally quantified with respect to basic domains (see Section 2). For instance, we may want to verify that only an auctioneer can open an auction or that a participant may eventually become the winner. These properties are formalized in FIEVeL as follows:

```
|=A forall X in AID G(done(open,X)-> exists A in AUCTIONEER
                                          [A.subject= X]);      (1)

|=E exists X in AID ( exists P in PARTICIPANT [P.subject=X]
            -> F exists W in CURWINNER [W.subject= X]);        (2)
```

While properties that hold in paths where agents may violate norms are useful to evaluate *worst cases* and to check whether an institution is *robust*, the *functionality* of an institution can be better evaluated by assuming that all agents are compliant (i.e., behave according to the norms of the system). For this reason, domain-dependent properties are specified in two different sections, one where agents are assumed to be compliant and another where all possible paths are considered.

To express agents compliance, we introduce a new proposition, *violation*, which is set to *true* whenever an obligation is violated. Therefore, *illegal states* where at least an obligation has been violated evaluates $violation$ to true: in doing so, our treatment of agents violations is somehow similar to the approach presented in [16] and [24]. Therefore, to check whether $\models_E \varphi$ under the hypothesis of agent compliance we verify whether:

$$\models_E \mathbf{G}\neg violation \wedge \varphi$$

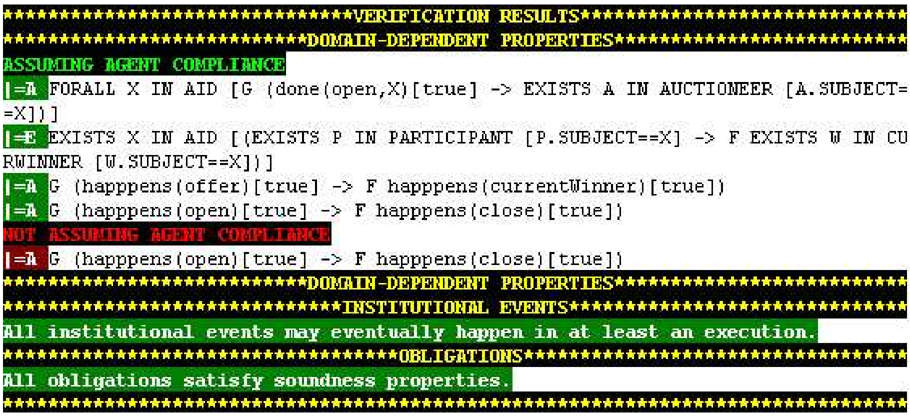
while under the hypothesis of agent compliance $\models_A \varphi$ is checked as follows:

$$\models_A \mathbf{G}\neg violation \rightarrow \varphi \textit{ and } \models_E \mathbf{G}\neg violation \wedge \varphi$$

For example, as mentioned above, assuming that the auctioneer is compliant with its obligations, we would expect that if a participant makes a bid, the auctioneer will declare the winner of that round, which can be expressed as follows:

$$\texttt{|=A G(happens(offer)->F happens(currentWinner));} \tag{3}$$

As we can see in Figure 3, property (3) holds only if we assume agents compliance. Verifying domain-dependent properties by assuming that agents will conform to the norms of the system not only allows us to check the functionality of an institution, but also considerably speeds up the verification process (see Table 1).



```
*******************************VERIFICATION RESULTS*****************************
***************************DOMAIN-DEPENDENT PROPERTIES**************************
ASSUMING AGENT COMPLIANCE
|=A FORALL X IN AID [G (done(open,X)[true] -> EXISTS A IN AUCTIONEER [A.SUBJECT=
=X])]
|=E EXISTS X IN AID [(EXISTS P IN PARTICIPANT [P.SUBJECT==X] -> F EXISTS W IN CU
RWINNER [W.SUBJECT==X])]
|=A G (happpens(offer)[true] -> F happpens(currentWinner)[true])
|=A G (happpens(open)[true] -> F happpens(close)[true])
NOT ASSUMING AGENT COMPLIANCE
|=A G (happpens(open)[true] -> F happpens(close)[true])
***************************DOMAIN-DEPENDENT PROPERTIES**************************
******************************INSTITUTIONAL EVENTS******************************
All institutional events may eventually happen in at least an execution.
**********************************OBLIGATIONS***********************************
All obligations satisfy soundness properties.
********************************************************************************
```

Fig. 3. The report generated by our tool during the verification of domain-dependent properties presented in Section 4.

We think that the availability of a tool for the automatic verification of institutions is especially useful when a designer wants to change an existing institution by introducing new institutional actions or norms. By defining new possibilities of actions or limiting existing ones, certain properties may cease to hold, compromising the functionality of an institution. In an earlier version of our specification of the English auction, we introduced the participant status function as an entry point for agents to participate to an auction. In that specification, among others, we checked, under the hypothesis of agents compliance, if every opened auction would be eventually closed, as stated by the following property:

$$\texttt{|=A G(happens(open) -> F happens(close));} \tag{4}$$

After that, we authorized participants to leave the auction during the time comprised among two consecutive rounds. In our first attempt, property (4) did not loger hold, since the auctioneer was authorized to declare a new round only if there were at least two agents (see Figure 1), while it was authorized to close the auction in such circumstances but was not obliged to do so. We solve this problem by introducing a new obligation for the auctioneer, and now property (4) is valid in our specification (see Figure 3).

5 Domain-Independent Properties

Domain-independent properties represent general desirable properties of institutions which stem from our metamodel (see [27]). For instance, a sound specification should be characterized by the fact that every institutional event must eventually happen in at least an execution where no violations occur. On the contrary, it would mean that either the preconditions of the institutional event are never met, that the designer has not defined the necessary authorizations, or that norms of the institution forbid agents from executing it whenever they are authorized. In any case, it is important that the designer becomes aware of this fact, and consider how to modify the institution, otherwise the definition of such institutional event would be useless.

A second important aspect that distinguishes domain-independent properties from domain dependent properties is that the former can be automatically generated by statically analyzing a specification of an institution, without considering the actual purposes of the designer when he or she has specified it. For instance, we can check whether an event of type *newRound* may occur by specifying the following property as a domain dependent property holding under the hypothesis of agent compliance:

$$\models_E \mathbf{G}\neg violation \wedge \mathbf{F}\mathtt{happens}(\mathtt{newRound})$$

The main point is that an event of type *newRound* is just an intermediate step, although a necessary one, to reach a state in which an agent wins the auction and the product is sold.

Finally, domain-independent properties are defined to guarantee the *rationality* of an institution with respect to the intended semantics of the concepts defined by our metamodel. For example, norms are introduced in open multiagent systems to constrain possible agents' behaviour, and therefore it must be the case that each norm can be eventually activated. If we assume that agents are autonomous, it should be possible for an agent both to violate and fulfill its obligations (and prohibitions), which means that norms regulate aspects of (social) reality which are contingent. It would be irrational to define an obligation characterized by an expression that makes it always fulfilled (or violated) independently of agent actions. Moreover, it should be the case that once a norm is activated, it ought to eventually reach a final state (fulfilled, violated, revoked), which guarantees that the whole life-cycle of a norm is limited and regulated only by the institution that defines it. Assuming that propositions $activated_i$, $violated_i$, and $fulfilled_i$ represent the fact that the i-th obligation associated to the j-th status function are respectively *activated*,*fulfilled*, or *violated*, and that $imposed\_sf_j$ and $revoked\_sf_j$ are true when the j-th status function is assigned or revoked, we can formalize the properties mentioned above as follows:

1. $\models_A \mathbf{G}((imposed\_sf_j \wedge (activated_i) \rightarrow \mathbf{F}(violated_i \vee fulfilled_i \vee revoked\_sf_j))$
2. $\models_E \mathbf{G}(imposed\_sf_j \rightarrow \mathbf{F}activated_i)$
3. $\models_E \mathbf{G}(imposed\_sf_j \rightarrow \mathbf{F}fulfilled_i)$
4. $\models_E \mathbf{G}(imposed\_sf_j \rightarrow \mathbf{F}violated_i)$

At the moment, these properties cannot be expressed in FIEVeL. Instead, they are generated by defining new constants in the Promela model which refer to the state of each obligation. We are considering how to extend FIEVeL such that a specification may refer to the current state of agents' obligations, which is a fundamental step to specify recovery policies and sanctions for the management of violations.

We think that the definition of a library of domain-independent properties, their automatic generation and verification is an important aspect of the development of sound institutions, since they regard the rationality of an institution and reflect certain aspects that may be critical in the development of an institution. Also, if we consider that a specification may be composed by many institutional events and norms, the support provided by a tool spares the designer the tedious task of hand coding them. Finally, their automatic definition allows the designer to focus his or her attention on the most relevant aspects of an institution to comply with his or her objectives.

6 The Verification Framework

To check with our tool whether an institution satisfies a set of domain-dependent properties, the designer must provide a *model definition* by describing which elements compose each basic domain (see Section 2) and the initial state of the system, defined in terms of which status functions are imposed on agents at the initial state. Moreover, the user can select which domain-independent properties must be generated to guarantee the soundness of the specification. At the moment, we have fully implemented only the verification of properties described in Section 5, although new domain-independent properties can be easily defined and incorporated into our tool thanks to its modular architecture.

Given an institution, a model description, a set of domain dependent properties specified by the designer, and a set of soundness aspects that should be checked, the tool automatically verifies the specification and creates a report of the verification activity as shown in Figure 3, which shows results obtained by checking most of the properties mentioned in this paper.

When a property does not hold, the model checker generates a counterexample [2, 12], which is used to extract the sequence of base-level events that characterize that execution. To better interpret why a trace violates a property, we have started to implement a translation of FIEVeL into a set of Java classes, which should help the designer to reason in terms of the abstractions provided by FIEVeL and to hide the details related with Promela and its syntax. Also, such classes represent a fist step to develop a simulator of institutions, which can help the designer to test his or her own specifications and to understand how they will evolve by interactively determining events that will be processed by the institution.

To provide the reader with a feeling of the computational costs of our approach, we report results obtained during the verification of property (1) presented in Section 4 on a desktop PC with installed Windows and equipped with a pentium 3.0 GHz and 0.5 GB of RAM. Table 1 reports our experiments, where "*" represents the fact that the verification process requires more than 500 seconds and therefore it has been interrupted. Analyzing results showed in Table 1, we can observe that by increasing the size of domains representing agents and prices we obtain a very fast growth of time and memory required to verify such property. This is essentially due to the fact that the number of agents and prices determines the number of possible events that can be interleaved with time events. Also, in our formalization of the English auction we keep trace of the order in which agents make their bids, so that the auctioneer is able to declare a *current winner* even if two agents have offered the same price during a round (the first agent that has bid the highest price is declared to be the current winner). Clearly, this fact contributes to increasing the number of possible paths that the model checker has to consider, and hence the amount of required memory and time.

Despite that, we consider our results interesting, considered the complexity of the specification and compared with our experience in the verification of systems specified in Promela. Instead, we are not able to offer to the reader any comparison with results obtained by related works, because most of them do not report any experimental result [13, 28, 24] or they mention results related with very simple properties [4]. In every case, Table 1 clearly indicates that our approach is able to cope only with small domains. In fact, performances rapidly decay with the growth of the size of the model due to the number of variables that concur to represent the state of the system and due to embedded code used to compute events preconditions and propositions of LTL formulas. Several techniques have been proposed in the literature to solve these problems, and we think that the most promising for the verification of institutions are *predicate abstraction* [11] and *symbolic representation* [3] of institutional transitions.

Agents	*Prices*	*Memory (MB)* agent compliance	*Time* (sec) agent compliance	*Memory (MB)*	*Time* (sec)
3	1	11.164	0.18	61.962	0.93
3	3	12.342	6.17	107.826	74.75
3	5	52.093	42.14	*	*
3	7	155.772	170.49	*	*
4	1	12.867	0.26	67.81	1.29
4	3	13.162	16.06	116.841	198.10
4	5	58.65	128.34	*	*
4	7	179.131	499.59	*	*

Table 1. Time and memory required to verify Property (1) of Section 4. Results are reported showing the size of the considered models and if they have been obtained assuming agent compliance.

7 Discussion and Conclusions

In this paper we have presented a framework for the definition and verification of institutions. The main advantage of our approach consists in the definition of a language that allows a designer to specify institutions and their desired properties in terms of a set of legal and philosophical concepts, and to automatically translate them into the input language of a model checker. Although in this paper we have exemplified our approach with a well known protocol, we expect that our framework is suitable for the specification and verification of systems where normative constraints and authorizations are generated by different sources and do not specify a precise sequence of messages and events. In presence of a collection of deontic positions that come from different sources, it is important to check that certain properties hold, to guarantee that agents and their designers can fulfill their objectives.

A few attempts have been previously carried out to apply model checking to verify institutions. [13] and [4] developed techniques to translate certain aspects of *electronic institutions* [8] into the input languages of SPIN or NuSMV [1]. Roughly speaking, the language presented in [8] describes institutions as composed by protocols, defined as finite automata, and a systems of norms, which may also regulate the execution of protocols by prohibiting agents from exchanging certain messages. [13] and [4] focus their attention only on properties of finite automata (e.g. "it is always possible to reach a final state"), while they do not take into account *normative rules*, that is, norms that model obligations agents get as a consequence of previous actions [8]. Instead, in our approach the attention is essentially focused on verifying properties of institutional states, described in terms of status functions, which are intrinsically related with the norms of the system. Moreover, in [13] each transition of the automata is interpreted an obligation, while according to [8] transitions represent agents' permissions. Therefore, it may occur that properties verified by the model checker actually do not hold in the original model and *vice versa*.

In [5, 22] the authors distinguish among *institutional domain facts* and *normative facts* (obligations, institutionalized power, and permissions), while in our approach we proposed a unified view of institutional domain and normative facts. Indeed, we claim that institutional facts are such only because they imply new normative facts for the interacting agents, which also represents a significant difference with respect to our previous attempts to model institutional reality [9].

In the literature only agent actions have been considered relevant to describe institutions [5, 9, 22, 24], and the attention has been focused on a single action type, namely the act of exchanging a message [8]. The importance of time events has been neglected [8, 24], while not only time events are important for the management of obligations, but also they can count as institutional events (see Figure 1). For instance, in most cultures the 18th birthday imposes new status functions on a person.

As we have discussed in Section 6, we are considering how to improve the performance of the verification process by applying *predicate abstractions* [11]. In particular, we think that the approach presented in [3], where SAT solvers-techniques have been proposed to increase the efficiency of the abstraction process, could be applied both to reduce the size of the state space and to obtain institutional paths starting from computational paths by symbolically representing institutional transitions.

ACKNOWLEDGMENTS The author would like to thank his Ph.D. advisor, Marco Colombetti, for fruitful discussions and criticisms about the contents presented in this paper.

References

1. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proceedings of the International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.
2. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambrige, MA, 1999.
3. E. M. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate Abstraction of ANSI-C Programs Using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.
4. O. Cliffe and J. Padget. A Framework For Checking Interactions Within Agent Institutions. In *Proceedings of the ECAI Workshop on Model Checking and Artificial Intelligence (MoChart I)*, 2002.
5. O. Cliffe, M. D. Vos, and J. Padget. Specifying and Analysing Agent-based Social Institutions using Answer Set Programming. In O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowski, J. S. Sichman, and J. Vázquez Salceda, editors, *Coordination, Organization, Institutions and Norms in Multi-Agent Systems*, volume 3913 of *LNAI*, pages 99–113. Springer, 2006.
6. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communication of the ACM*, 18(8):453–457, 1975.
7. M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In C. Castelfranchi and W. L. Johnson, editor, *Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems (AAMAS '02)*, pages 1045–1052, New York, NY, USA, 2002. ACM Press.
8. M. Esteva, J. A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the Formal Specification of Electronic Institutions. In F. Dignum and C. Sierra, editors, *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *LNAI*, pages 126–147. Springer, 2001.
9. N. Fornara, F. Viganò, and M. Colombetti. Agent Communication and Institutional Reality. In R. van Eijk, M. Huget, and F. Dignum, editors, *Agent Communication*, volume 3396 of *LNAI*, pages 1–17. Springer, 2005.
10. A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. W. Vasconcelos. A distributed architecture for norm-aware agent societies. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Declarative Agent Languages and Technologies III, Third InternationalWorkshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, volume 3904 of *LNCS*, pages 89–105. Springer, 2005.
11. S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In *Proceedings of the 9th International Conference on Computer Aided Verification*, pages 72–83. Springer-Verlag, 1997.
12. G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.
13. M.-P. Huget, M. Esteva, S. Phelps, C. Sierra, and M. Wooldridge. Model Checking Electronic Institutions. In *Proceedings of the ECAI Workshop on Model Checking and Artificial Intelligence (MoChArt I)*, 2002.

14. A. Jones and M. J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
15. R. A. Kowalski and M. J. Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 4:67–95, 1986.
16. A. Lomuscio and M. Sergot. A formulation of violation, error recovery, and enforcement in the bit transmission problem. *Journal of Applied Logic (Selected articles from DEON02 - London)*, 1(2):93–116, 2002.
17. C. Masolo, L. Vieu, E. Bottazzi, C. Catenacci, R. Ferrario, A. Gangemi, and N. Guarino. Social Roles and their Descriptions. In D. Dubois, C. Welty, and M. Williams, editors, *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, pages 267–277, 2004.
18. K. Meinke and J. V. Tucker, editors. *Many-sorted logic and its applications*. John Wiley & Sons, Inc., New York, NY, USA, 1993.
19. A. Morzenti, D. Mandrioli, and C. Ghezzi. A Model Parametric Real-Time Logic. *ACM Transactions on Programming Languages and Systems*, 14(4):521–573, 1992.
20. Y. Moses and M. Tennenholtz. Artificial Social Systems. *Computers and AI*, 14(6):533–562, 1995.
21. J. S. Ostroff. Deciding properties of timed transition models. *IEEE Transactions on Parallel Distributed Systems*, 1(2):170–183, 1990.
22. J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Formalization of a voting protocol for virtual organizations. In *Proceedings of the 4th International Joint Conference on Autonomous agents and Multi-Agent Systems (AAMAS 2005)*, pages 373–380, New York, NY, USA, 2005. ACM Press.
23. J. R. Searle. *The construction of social reality*. Free Press, New York, USA, 1995.
24. M. J. Sergot. Modelling unreliable and untrustworthy agent behaviour. In B. Dunin-Keplicz and A. Jankowski and A. Skowron and M. Szczuka, editor, *Monitoring, Security, and Rescue Techniques in Multiagent Systems)*, Advances in Soft Computing, pages 161–178. Springer, 2005.
25. W. Vasconcelos. Logic-based electronic institutions. In J. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Workshop on Declarative Agent Languages and Technologies (DALT 2003)*, volume 2990 of *LNCS*, pages 221–242. Springer, 2003.
26. F. Viganò. FIEVeL, a Language for the Specification and Verification of Institutions. Technical Report 3, Institute for Communication Technologies, Università della Svizzera Italiana, 2006.
27. F. Viganò and M. Colombetti. Specification and Verification of Institutions through Status Functions. In *Proceedings of the AAMAS Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN06)*, 2006.
28. C. D. Walton. Model Checking Agent Dialogues. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, New York, Revised Selected Papers*, volume 3476 of *LNCS*, pages 132–147. Springer, 2004.