

# Specifying and Monitoring Market Mechanisms using Rights and Obligations

Loizos Michael, David C. Parkes, and Avi Pfeffer

Division of Engineering and Applied Sciences  
Harvard University, Cambridge, MA 02138, U.S.A.  
{loizos, parkes, avi}@eecs.harvard.edu

**Abstract.** We provide a formal scripting language to capture the semantics of market mechanisms. The language is based on a set of well-defined principles, and is designed to capture an agent's rights, as derived from property, and an agent's obligations, as derived from restrictions placed on its actions, either voluntarily or as a consequence of other actions. Rights and obligations are viewed as first-class goods, from which we define fundamental axioms about well-functioning market-oriented worlds. Coupled with the scripting language is a run-time system that is able to monitor and enforce rights and obligations. Our treatment extends to represent a variety of market mechanisms, ranging from simple two-agent single-good exchanges, to complicated combinatorial auctions.

## 1 Introduction

Many authors have written about a future of agent-mediated electronic commerce, in which agents engage in commerce on behalf of individuals and businesses. We take this idea seriously, and provide a formal scripting language for describing economic markets that is: (i) natural and easy to understand, for humans to be able to participate, (ii) formal and unambiguous, for artificial agents to be able to participate, and (iii) amenable to automatic monitoring.

The need for a solution to such a problem naturally arises in a variety of contexts, with most prevalent that of online transactions between agents, human or artificial ones. As economic markets increasingly enjoy the benefits of modern technology and the internet, the need for an appropriate way of describing these markets in a computer-compliant, yet human-friendly way becomes more of an issue, justifying all three of the said goals.

An equally important context, which is also of great interest to the audience of this workshop, is the need for a platform for testing new agent designs, simulating new mechanism designs, and evaluating their properties. Our framework provides such a platform, by implementing a set of well-defined design principles, and respecting the aforementioned goals. We claim that these principles and goals are central, and propose that they should serve as a starting point for the specifications of platforms for describing and monitoring market mechanisms.

The scripting language we propose captures the essential semantics, namely *rights* and *obligations*, of market mechanisms. Rights enable agents to obtain

utility by taking actions on goods that they own, while obligations allow them to engage in safe transactions and to make credible commitments to the rules of market mechanisms.

We adopt rights and obligations as first class goods, from which fundamental market axioms can be derived. These axioms are enforced within a monitoring environment, that we couple with our formal scripting language. Given a description of a market mechanism, the monitoring environment implements the market in a prescribed way, thus giving precise semantics to our scripting language.

Agents can interact with the monitoring environment and affect, through their actions, the state of the virtual market. During such an interaction, agents themselves can initiate new market mechanisms, by specifying obligations on their behavior and granting rights to participants (such as the right to place a bid). We take a black-box approach to the agent specification, and impose no restrictions to their design and internal workings. This frees the monitoring environment from complex activities such as planning for agents, or winner-determination in auctions. The monitoring environment, however, can verify whether certain goals are established, by having agents state obligations and then provide sufficient information to enable the easy verification of their satisfaction. For instance, an auctioneer can provide market-clearing prices to allow the monitoring environment to check that the outcome satisfies a competitive equilibrium, without the need for the system to compute that equilibrium itself. Thus our approach provides a middle road between a completely formal but hard to program system, and a completely open-ended but informal system.

The framework is introduced through a discussion of its main characteristics and capabilities. We demonstrate its flexibility through a number of examples, including an English auction, a second-price auction, and a combinatorial auction. Given the context of this workshop, we keep our discussion at a non-technical level throughout the paper. We close with a discussion of the broad scope and applicability of our framework for formal approaches.

## 1.1 Related Work

Our approach is consistent with economic theory on property rights and organization theory. Quoting Tirole [13], “*a decision right or authority granted to a party is the right for the party to pick a decision in an allowed set of decisions. A property right on an asset, i.e. its ownership, is a bundle of decision rights.*” It is standard to model a firm as a collection of assets and consider the ability of a firm to retain a specific subset of its bundle of rights while selling all other residual rights [5]. The role of obligations and commitment is recognized to be important for efficient contracts [5], and for auction and mechanism design [6].

Prior work in multi-agent systems has considered the role of rights and obligations for the specification and semantics of open systems [3, 1, 4, 15, 12], with approaches differing in whether the monitoring environment actively enforces sanctions (as in our work) [1, 4], or only passively maintains the global state and informs agents of their obligations [3]. Approaches also differ as to whether obligations are state-based (as in our work) [3] or specified in terms of actions

that an agent must perform in a particular state [1]. Some work [12, 3] observes that agents might contract other agents to satisfy the formers’ obligations, but none of this work adopts rights and obligations as first class goods that agents can explicitly trade and exchange. Similarly, we are unaware of any work that explicitly sets out to model the rights that derive from goods in economic worlds or the semantics of ownership and possession.

Our notions of conditional, limited, and disjunctive rights are shared with previous work on formal specification languages for financial contracts [7], although that work focuses on the formal description and analysis of new forms of financial contracts and not on providing frameworks for open agent societies. The  $\pi$ -calculus has also been used for the specification of a complex model of a Spanish fish market [10], although again the goal in that work was to assist with the development of complex institutions rather than provide semantics for participants or monitor and enforce properties of dynamic state.

The formal theory of deontic logic [8, 9], the logic of rights and obligations, is concerned with performing valid inference in high-level deontic logics, seeking to establish the validity of statements like “is every obligatory action permitted?” (\*) A duality between rights and obligations provides a cornerstone of deontic logic, with an obligation defined as an action that must be performed when no other action can be, due to lack of rights. Our work differs in this aspect, by defining rights on actions, but obligations in terms of properties on states. We are also agnostic to questions like (\*) because we focus on soft obligations with sanctions rather than hard obligations, an approach termed “contrary-to-duty” in the deontic logic literature [9]. In particular, our agents can make mistakes and take actions that lead to dead-ends in which obligations cannot be met.

## 2 Framework Overview

In this work we propose a framework comprised of a scripting language and a monitoring environment, with the former providing the necessary syntax for describing economic markets, and the latter providing the language semantics. This is directly analogous to the case of programming languages, where a programmer uses the language to write a program, with the semantics defined through the program’s execution in a prescribed manner. The programmer in our framework is the *domain designer*, and the program is the *domain description*, a collection of laws governing the particular economic market being modeled. As an ordinary program can import libraries that provide specific functionality, so is the case with a domain description. The domain designer can import libraries describing laws of economic markets that are commonplace in a variety of settings. We have written such libraries, like: (i) a library on “exchanges of goods” with laws on how goods can be traded, given, or sold between agents, and (ii) a library on “handling rights and obligations” with laws on how rights can be given up, issued, or revoked, and how obligations can be taken on, imposed, or cleared.

A domain description is fed into the monitoring environment, which then runs a virtual market world governed by the laws specified in the domain de-

scription. In particular, the laws define the initial state of affairs of the market, the objects that populate it, and the relevant properties of these objects, whose values essentially determine the state of affairs as the market evolves over time. The laws also dictate how agents might join or leave the market, and the available actions through which the agents might affect and observe the market's status. The agents are not simulated as part of the virtual world, but are acting independently of and simply communicate with the monitoring environment.

It is clear that every implementation of a given market may lead to a different sequence of states describing the evolution of the virtual world. Each such sequence is called a *scenario* and corresponds to a specific instantiation of an economic market. All scenarios share the same initial state, and the same potential/capacity for evolving in certain ways, as this is determined by the domain description. The actual scenario, though, is ultimately defined by agents' actions.

## 2.1 Design Principles

Our proposed framework implements a set of well-defined design principles, which we discuss below:

**Black-Box Principle:** Agents are entities that exist outside our framework, implemented in some fashion that is (possibly) independent of the proposed scripting language. They can reason based on their own beliefs and freely choose to take actions or not, within the market world they participate in.

**Free-Will Principle:** We cannot force agents to take specific actions, and in particular, to take actions that satisfy their obligations. Instead, we impose punitive sanctions to agents that fail to meet their obligations.

**Restriction Principle:** The monitoring environment is able to restrict the execution of actions for which appropriate rights are not held by the agents.

**Soundness Principle:** When an action is actually executed (i.e., when the appropriate right was held and the invoked action was physically executable in the current state of the virtual world), its effects are produced in accordance with the laws of the economic market being modeled.

The first two principles exemplify the generality of the framework we propose. Agents are treated as black boxes, without imposing any requirements on their internal workings, other than their ability to interact with the provided interface. As such, we cannot force agents to act in a prescribed way, which justifies the approach of using punitive sanctions, as we do in this work.

Our Restriction Principle is justified in the context of the virtual worlds we consider here; agents can only request that actions be taken, while the final decision lies with the monitoring environment, which screens the action execution requests based on agents' rights. This principle is further supported, when viewed in conjunction with the Free-Will Principle: An agent's options can be limited by the monitoring environment, but the agent still retains the choice of which (if any) option to exercise. This is the situation faced by any agent trying to devise a plan to achieve some goal; the actions available to the agent are restricted (albeit

not by lack of rights, but rather by lack of physical ability), but the agent itself is responsible for choosing appropriate actions that will fulfil its goal.

Finally, the Soundness Principle provides an appropriate soundness condition for our framework, by stating that the monitoring environment will always respect the laws of the market, as these are described by the domain designer.

## 2.2 eBay Example

In this section we demonstrate the parallels between eBay and our framework, illustrating how our stated principles are justified by existing virtual markets.

eBay participants freely choose to join or leave eBay’s virtual world. While there, they interact with the market by invoking actions (e.g., define upper bounds on bids) through the provided interface, and depending on whether certain conditions are met (e.g., if the bid was actually a valid numerical value), the actions are executed and produce their effects (e.g., the input value becomes the agent’s new upper bound for the proxy bidding). The agents then observe the new state of affairs, and continue to invoke their next action (if any).

Notice how the Black-Box Principle applies here, with the agents being independent of the engine (i.e., eBay’s servers) running the virtual world, and that the only requirement they need to meet is that they can interact with eBay’s market through the provided interface (e.g., web page links and forms). The Free-Will Principle applies in particular, with eBay not forcing a participant to honor a transaction, but punishing violators by means of negative feedback.<sup>1</sup>

The Restriction Principle relates to how eBay participants can auction items, or bid on items, only if they have appropriate rights. Thus, a participant does not have the right to place a bid lower than the current highest bid; invoking such an action will result in the action being rejected and not executed.

Lastly, the Soundness Principle applies, for instance, in that a paying agent is guaranteed that if the paying action is executed, then the payment will be made, no matter what other events (e.g., the concurrent execution of some other payment, or the fact that some auction closed one hour ago) take place.

## 3 Rights and Obligations

Agents pursuing goals, either by choice, or as an imposed requirement, often have restrictions on their available options of how to meet these goals. These options and goals correspond precisely to the notions of rights and obligations, which play a prevalent role in our framework. We view rights and obligations as tradable goods that can be given, taken, exchanged, sold, or auctioned. As such, rights and obligations are treated as any normal object, with a set of pre-determined properties, whose values are part of the market’s state. We contend

---

<sup>1</sup> From <http://pages.ebay.com/help/confidence/programs-investigations.html>: “eBay cannot force a seller to honor their transactions. You should leave appropriate feedback for the reluctant seller...”. The action of providing negative feedback is taken by participants, but it is ultimately provided by eBay as a sort of a punitive sanction.

that viewing rights and obligations as goods is necessary for defining natural economic market protocols, and we illustrate the strength of such a treatment via a number of example domain descriptions later on. To give a taste of why this idea is in fact very powerful, consider an audio compact disk sold in an eBay auction, with the winner being awarded the item as per eBay’s rules. What is implicit in this transaction is that the winner is also awarded the rights and obligations accompanying the item, and in particular, the right to listen to the audio compact disk, and the obligation not to infringe the copyright of the producers of the music. That is, in reality the auctioneer was not simply selling an audio compact disk, but rather a bundle of goods that include the item itself and certain rights and obligations. We make this transfer of rights and obligations explicit in our framework, giving the domain designer great flexibility as to the economic markets that can be modeled.

We view rights as the options of an agent participating in a market mechanism, coming from property or possession of items, or otherwise given to the agent. Rights determine the actions that an agent can take as a means of fulfilling its own private goals, by qualifying the executability of actions. In their full generality rights are conditional, with their provisions being applicable only under certain conditions.

**Definition 1 (Rights).** *We let  $\text{right}(\#action, \#condition)$  denote the right to execute action  $\#action$  whenever condition  $\#condition$  is true.*

As an example, an agent renting a car from 10:00am to 6:00pm might be given a right of the form  $\text{right}(\text{drive\_car}, 10:00am \leq \text{Time} \leq 6:00pm)$ . If the condition is not met (and given that the agent does not have any other rights on driving the car), then the agent cannot drive the car, virtue of the Restriction Principle. The syntax of conditional rights is sufficiently expressive to account for perpetual and expiring rights, and for more involved rights, such as the perpetual right to buy bonds, but only once every year and within a limited time span.

On the other side we have obligations, which we view as constraints on an agent’s behavior, or goals an agent should fulfill as a participant in some market mechanism. An agent freely chooses when and how to satisfy its obligations by appropriately exercising its rights, in the spirit of the Free-Will Principle. Rather than enforcing, via planning, that agents meet their obligations, we let the monitoring environment detect violations and appropriately impose punitive sanctions, as these are defined by the domain designer or the participating agents. Sanctions might include the revocation of an agent’s rights, the loss of money or possessions, the enforcement of additional obligations, or the banning of an agent from participating in the market mechanism altogether.

**Definition 2 (Obligations).** *We let  $\text{obligation}(\#satisfy, \#violate, \#punishment)$  denote the obligation of ensuring that condition  $\#satisfy$  is satisfied no later than condition  $\#violate$ , under penalty of executing action  $\#punishment$ .*

The obligation is flagged as satisfied or violated according to which condition is met first, and in the case of a violation the appropriate punitive sanction is

imposed through the execution of action *#punishment*. This general form allows us to represent obligations of the following forms

*obligation(false, balance(alice)<1000, close\_account(alice))*  
*obligation(balance(alice)>1000, Time>12/31/2004, deduct\_account(alice, 100))*

Assuming that Alice holds such obligations, then in the first case she must ensure that her bank balance does not drop below 1000 dollars at any time, under penalty of her bank account being closed, while in the second case she must ensure that her bank balance goes above 1000 dollars (but not necessarily stays there) at some time before the end of the year, under penalty of 100 dollars being deducted from her account.

## 4 Objects, States, Actions

The dynamic model of our monitoring environment is fairly standard. The world goes through a sequence of states, with each state specifying values for the properties of certain objects that populate the state. Each object is associated with a class (like in object oriented programming), of which the object is an instance of, and which defines the set of properties of the object. Every object has a unique name used by agents to reference that object. A set of basic classes are defined by our framework, but the domain designer can extend this set.

The use of objects provides a uniform treatment for both physical goods, like apples, and abstract goods, like rights and obligations. Money is supported through the use of *account* objects, with an object property corresponding to the account balance. Transferring money through payments is equivalent to changing this balance in an appropriate way. The notions of ownership and possession are also readily supported as properties of objects. Transferring an item from some agent to another reduces to simply changing the values of these properties.

The properties and existence of these objects are only affected by means of actions taken by the agents, through their interaction with the monitoring environment. In their *primitive* form, actions have preconditions and effects. When the monitoring environment attempts to execute an action, following its *invocation* by an agent, it first checks whether the agent holds an appropriate right, and whether the preconditions of the action are satisfied, and subsequently updates the state according to the action's effects. The set of effects is as follows.

**Definition 3 (Effects).** *We let  $create(\#object, \#class)$ ,  $destroy(\#object)$ , and  $set(\#object, \#property, \#value)$  denote respectively the action effects that create object  $\#object$  as an instance of class  $\#class$ , destroy object  $\#object$ , and set the property  $\#property$  of object  $\#object$  to the value  $\#value$ .*

Our framework also supports more expressive conditional and quantified effects, special instances of which are the non-deterministic, or probabilistic effects.

In their *transactional* form, actions are ordered sequences of actions (which may be primitive or transactions themselves). As in databases, when executing a

transaction, either all or none of the actions are successful. The execution model is to execute each of the actions in turn, updating the world state after each action. If any of the actions fails to meet its preconditions, or the agent fails to have an appropriate right at the time of each primitive action’s execution, the entire execution is rolled back to its original state. Conceivably the first action can grant or revoke an agent’s right to execute a subsequent action in the transaction, allowing for a really expressive set of transactions to be modeled.

The notion of transactions is very powerful and useful, with a number of applications, like that of implementing safe exchanges of goods. The transaction

$$\textit{sell}(\textit{apple}, 1, \textit{alice}) \equiv \textit{transaction}([\textit{give}(\textit{apple}, \textit{alice}), \textit{take\_money}(1, \textit{alice})])$$

for instance, specifies a fail-safe way for Bob to sell his apple to Alice for one dollar, without either party being vulnerable to the other’s renegeing. Transactions can also be used in a number of other contexts, including that of implementing disjunctive or expiring rights, where the agent with the disjunctive/expiring right essentially has the right of executing a transaction comprised of the action intended to be executed, and followed by the agent giving up the right.

A certain set of basic primitive actions and transactions are implemented by our framework, including actions for transferring ownership or possession of goods, transferring money, issuing or giving up rights, taking on obligations, etc. In addition to the built-in actions, the domain designer may define actions specific to the domain being modeled, like, for instance, the *fill\_gas*(#car) action whose effect is that of setting the *gas\_level* property of the #car object to *full*.

We capture exogenous events that are outside the agents’ control by allowing the monitoring entity to execute certain actions and attributing their execution to an all-powerful “god” agent. Thus, for instance, the initial state of the system is populated by means of the god agent executing the *initialize* action once the domain description is loaded. Although the god agent can execute only a certain fixed set of actions, on a well-defined set of occasions, these actions are in general transactions, with their constituent actions being defined as part of the domain description. This allows the domain designer to essentially specify the effects of god’s intervention, in situations like the arrival or departure of an agent, or the violation of some obligation, in which case the god agent executes the punitive action associated with the violated obligation.

An important special action implemented by our framework is that of querying, which serves as a way to implement private information. We treat the values of object properties as been hidden from an agent, unless the agent has an appropriate right to query an object property for its value.

**Definition 4 (Query).** *We let  $\textit{query}(\#\textit{object}, \#\textit{property})$  denote the action of querying the value of property #property of object #object. When the action is executed, the agent that invoked the action learns the queried value.*

For example, the property representing the collected bids in a sealed-bid auction is only viewable by the auctioneer, thus preserving secrecy.

## 5 Ownership and Possession

Property rights are a basic building block of markets and our framework takes a stand on what the rules governing these property rights should look like. To start with, we make an important distinction between ownership and possession. Ownership of an object implies a bundle of rights, including the right to use the object and the right to sell it. It also includes the right to sell various rights to the object. For possession, we use the word *holding*, which we take to mean rightful possession. When one holds something, one has the ability to use it through possession, and one also has the right to use it. However, one does not have the right to sell it or to sell any rights to it. This is a common status in the real world. For example, when someone rents a car, he has possession of it and the right to use it (for a limited time), but he does not have the right to sell it. We make precise the notions of ownership and possession through the following axiomatic definitions.

**Definition 5 (Ownership Axiom).** *We take ownership of a good to be synonymous with owning the right of setting the properties of the good to any values physically possible. We call this the Fundamental Axiom of Ownership.*

Our framework implements the Fundamental Axiom of Ownership by issuing ownership of a right of the form

$$\text{right}(\#action, \text{accessible}(\#action, \#agent))$$

to every agent  $\#agent$  joining a virtual market, where  $\text{accessible}(\#action, \#agent)$  holds exactly when action  $\#action$  only affects properties of goods owned by agent  $\#agent$ . By exercising this right, the owner of an apple can sell or give possession of the apple, since the effects of these actions are only affecting the *owned\_by* and *held\_by* properties of the apple. Notice that in the latter case, the owner can actually take the apple back, since he still owns the right of setting the possessor of the apple. In particular, this implies that an agent owning a right, but not holding it, can still rightfully execute an action, since the agent can always reclaim possession of the right, execute the action, and then return the right to its previous possessor, all within a single transaction.

**Definition 6 (Possession Axiom).** *We take possession of a good to imply possession of the right to use the good in a set of prescribed ways associated with the good's class. We call this the Fundamental Axiom of Possession.*

As before, our framework implements the Fundamental Axiom of Possession by issuing possession of a right of the following form to all participating agents

$$\text{right}(\#action, (\text{object}(\#object), \text{value}(\#object, [(\#property, \#value), \dots])), \text{member}(\#action, \#uses)))$$

where  $\text{object}(\#object)$  holds exactly when object  $\#object$  exists, and  $\text{value}(\#object, [(\#property, \#value), \dots])$  holds exactly when the property  $\#property$  of object  $\#object$  has value  $\#value$ , for every property-value pair in the list.

Notice that the rights associated with the Fundamental Axioms of Ownership and Possession are respectively owned and simply held by agents. Hence, in the former case the Fundamental Axiom of Ownership applies recursively on the associated right itself with the right being the owned object. So, an agent owning a car, not only owns the right to drive it, but the agent also owns the right to sell the right to drive the car, to some other agent. Selling the right to use an object without selling the object itself is extremely common. For example, you might sell someone the right to walk across your land without selling the land.

Rights in real life are often not given, but rather issued. When you give someone the right to walk on your land you still retain that right for yourself, exactly because you do not give that person your instance of the right, but rather you issue a new copy of the right. This is achieved through the use of an issuing action defined by our framework, and appeals to the following axiom.

**Definition 7 (Rights Axiom).** *We take ownership of a right to imply ownership of the right to issue ownership or possession of the former right (with not weaker conditions) to others. We call this the Fundamental Axiom of Rights.*

Other fundamental axioms are also defined in our framework, such as axioms relating to performing transactions (e.g., giving someone the right to exchange goods with you). All axioms are implemented by issuing suitable rights to agents.

## 6 Implementation Issues

Both the monitoring environment and the specification language are currently implemented in Prolog, whose goal-oriented computation is a natural fit with the computational tasks of our framework (e.g., checking if conditions are met).

Agents joining the monitoring environment are assigned a private channel, through which all subsequent communication is taking place, thus associating each exchanged message with a unique agent. Communication is taking place asynchronously, while the monitoring environment employs a continuous treatment of time, with actions occurring instantaneously.

In a typical execution, an agent is sent a Prolog list containing all the object properties of the current state that are visible to the agent. Given the received message, the agent reasons and chooses to invoke some action by replying with the predicate *invoke(#action)*. The monitoring environment records the invocation event and attempts to execute the action. Success or failure of actions is recorded and the state of the virtual world is updated and stored in a database that can be later used to review the evolution of a scenario. Periodically, the monitoring environment checks whether any obligation has been satisfied or violated, recording the event and enforcing the appropriate punitive sanction.

Regarding the scalability of our framework, we note that we are not concerned with the problem of planning, but with that of execution monitoring, which remains tractable as long as the conditions of actions, rights, and obligations are not inherently hard to begin with. Preliminary experimental results using agents and markets we have implemented, show that such intractability is not an issue.

## 7 Example Representations

In this section we represent a number of different auction markets within our framework. We remind the reader that these representations are not meant to describe the agents participating in an auction (which can be implemented in some arbitrary language), but rather the rules of the auction itself. For conciseness, we only present those rules that are concerned with defining the central actions available to the agents. We use boldface to indicate the main language operators, and underlining to indicate action names. We have also substituted certain parentheses with curly brackets, to enhance readability. Other than these cosmetic enhancements, the domains are presented in the actual Prolog implementation of the scripting language of our framework. The full domain descriptions can be found online at <http://www.eecs.harvard.edu/~loizos/norms.html>.

The object *clock* is an instance of the *event* class, and serves as a way to hold the time at which the current state of the world was instantiated. The various predicates used are provided by our framework and were already described in previous sections. The actions *sell*(*#good*,*#price*,*#receiver*) and *jail*(*#agent*) are imported from the appropriate libraries, with the latter retracting all the rights of an agent, when executed. The action *issue\_p*(*right*(*#action*,*#condition*),*#agent*) is the built-in action of issuing possession of rights to agents. We assume that agents have the right to open auctions on items they own, and unless otherwise stated, agents have the right (given to them at the opening of an auction) of viewing all the auction properties, and the properties of the auctioned item.

### 7.1 Open-Cry English Auction

In a typical open-cry English auction scenario, an agent owning an item invokes the action of opening an auction, in order to set up the auction parameters (through the *create\_auction* action not shown here). The auctioneer also gives all bidders the right to place bids, conditioned on the new price being higher than the current price. Finally, the auctioneer commits to closing the auction, and selling the item to the highest bidder soon after that.

```
action(Agent, open_auction(Auction, Item, OpeningPrice)) :- transaction([  
create_auction(Auction, Item, OpeningPrice),  
take_on(obligation( { value(Auction, status, closed) } , { value(clock, happened_at, Time),  
value(Auction, last_bid_time, LastBidTime), atleast(Time, LastBidTime+100) }  
, { jail(Agent) } )),  
take_on(obligation( { value(Auction, [ (status, closed), (highest_bid, HighestBid),  
(highest_bidder, HighestBidder) ]), object(Event), value(Event, [ (instance_of, event),  
(description, invoked(Agent, sell(Item, HighestBid, HighestBidder), successfully)) ] ) }  
, { value(Auction, [ (status, closed), (highest_bidder, HighestBidder), (closing_time,  
ClosingTime) ]), value(clock, happened_at, Time), atleast(Time, ClosingTime+100),  
HighestBidder \= Agent } , { jail(Agent) } )),  
issue_p(right( { place_bid(Auction, Bid) } , { value(Auction, [ (highest_bid, HighestBid),  
(status, open) ]), atleast(Bid, HighestBid+1) } ), Bidder) ]).
```

Bidders proceed to place bids by raising the current highest bid, and granting to the auctioneer the right to sell them the item.

```
action(Agent, place_bid(Auction, Bid)) :- transaction([ raise_bid(Auction, Bid),
issue_p(right( { sell(Item, Bid, Agent) } , { value(Auction, [ (status, closed), (highest_bid,
Bid), (highest_bidder, Agent) ]), value(clock, happened_at, Time),
atleast(ClosingTime+100, Time) } ), Auctioneer) where value(Auction, [ (auctioneer,
Auctioneer), (item, Item) ] ) ).
```

```
action(Agent, raise_bid(Auction, Bid)) :- preconditions([ object(Auction),
value(Auction, highest_bid, CurrentBid), atleast(Bid, CurrentBid+1) ]), effects([
set(Auction, highest_bid, Bid), set(Auction, highest_bidder, Agent), set(Auction,
last_bid_time, Time) where value(clock, happened_at, Time) ]).
```

At the end, the auctioneer closes the auction and continues to invoke the *sell* action, as obliged by the rules of the auction.

## 7.2 Sealed-Bid Second-Price Auction

In a typical sealed-bid second-price auction scenario, an agent opens an auction in the same manner as in the open-cry English auction scenario. In this case we assume that agents do not have the right to view the *set\_of\_bids* property of the auction. We omit the part of the domain that achieves this.

```
action(Agent, open_auction(Auction, Item, OpeningPrice)) :- transaction([
create_auction(Auction, Item, OpeningPrice),
take_on(obligation( { value(Auction, [ (status, closed), (set_of_bids, SetOfBids), (winner,
HighestBidder), (payment, SecondHighestBid) ]), get_second_price(SetOfBids,
SecondHighestBid), get_first_bidder(SetOfBids, HighestBidder) } , { value(clock,
happened_at, Time), value(Auction, last_bid_time, LastBidTime), atleast(Time,
LastBidTime+100) } , { jail(Agent) } )),
take_on(obligation( { value(Auction, [ (status, closed), (winner, HighestBidder), (payment,
SecondHighestBid) ]), object(Event), value(Event, [ (instance_of, event), (description,
invoked(Agent, sell(Item, SecondHighestBid, HighestBidder), successfully) ] ) }
, { value(Auction, [ (status, closed), (winner, HighestBidder), ]), value(clock, happened_at,
Time), atleast(Time, ClosingTime+100), HighestBidder \= Agent } , { jail(Agent) } )),
issue_p(right( { place_bid(Auction, Bid) } , { value(Auction, status, open) } ), Bidder) ]).
```

```
action(Agent, create_auction(Auction, Item, OpeningPrice)) :- preconditions([ \+
object(Auction) ]), effects([ create(Auction, sealed_auction), set(Auction, owned_by,
Agent), set(Auction, held_by, Agent), set(Auction, auctioneer, Agent), set(Auction, status,
open), set(Auction, item, Item), set(Auction, set_of_bids, [(Agent,OpeningPrice)]),
set(Auction, winner, undefined), set(Auction, payment, undefined), set(Auction,
last_bid_time, Time) where value(clock, happened_at, Time), set(Auction, closing_time,
undefined) ]).
```

Bidders proceed to submit their sealed-bids by updating the *set\_of\_bids* value, without however, ever seeing its contents. Placing a bid is conditional on this being the first bid placed by the bidder.

```

action(Agent, place_bid(Auction, Bid)) :- transaction([ submit_bid(Auction, Bid),
issue_p(right( { sell(Item, Bid, Agent) } , { value(Auction, [ (status, closed), (payment,
Bid), (winner, Agent) ]), value(clock, happened_at, Time), atleast(ClosingTime+100, Time)
} ), Auctioneer) where value(Auction, [ (auctioneer, Auctioneer), (item, Item) ] )]).

```

```

action(Agent, submit_bid(Auction, Bid)) :- preconditions([ object(Auction),
value(Auction, set_of_bids, SetOfBids), \+ member((Agent,AnyBid), SetOfBids) ]),
effects([ set(Auction, set_of_bids, [(Agent,Bid)|SetOfBids]) where value(Auction,
set_of_bids, SetOfBids), set(Auction, last_bid_time, Time) where value(clock,
happened_at, Time) ]]).

```

Finally, the auctioneer closes the auction by declaring a winner and a payment and continues to invoke the appropriate *sell* action.

### 7.3 Combinatorial Auction

The case of a combinatorial auction resembles the sealed-bid auction case, so we only briefly discuss the main points of difference. The auctioneer opens the auction for a set of items, invoking (amongst other things) the following action

```

take_on(obligation( { value(Auction, [ (status, closed), (set_of_bids, SetOfBids), (allocation,
Allocation), (prices, Prices), (payments, Payments), (marginal_allocations,
AllocationPerMarginalMarket), (marginal_prices, PricesPerMarginalMarket) ]),
AllAllocations = [Allocation|AllocationPerMarginalMarket], AllPrices =
[Prices|PricesPerMarginalMarket], checkOutcomeEfficiency(SetOfBids, AllAllocations,
AllPrices), checkVCGPayments(SetOfBids, AllAllocations, Payments) } , { value(clock,
happened_at, Time), value(Auction, last_bid_time, LastBidTime), atleast(Time,
LastBidTime+100) } , { jail(Agent) } ) ),

```

for committing to an efficient outcome and VCG payments (see e.g., [6]). Bids are then placed, each specifying an explicit subset of the auctioned items. Agents can submit multiple bids (while issuing the corresponding rights), any of which can be satisfied according to the semantics of the OR-bidding language. When closing the auction, the auctioneer computes (outside the monitoring environment) and provides an allocation and prices for goods, payments for each agent, as well as allocations and prices for every marginal market; marginal markets are those where a single agent is taken out of consideration. According to economic theory, for each market (including the marginal ones) one can use the bids, the allocation and the prices to verify that the outcome is an efficient one, by checking that the allocation is such that the total surplus equals the total value. These two quantities correspond to the objective functions of the primal and dual LP formulations of the combinatorial allocation problem (see e.g., [2]), and hence if they are equal (and feasible solutions) they imply that they are also the optimal solutions to the problem. This check can be performed in polynomial time, and it is taken care of by a call to the *checkOutcomeEfficiency* predicate. Given the outcomes of all the markets (including the marginal ones) are efficient, one can also easily check that the VCG payments are equal to the ones specified by the auctioneer. This is taken care of by a call to the *checkVCGPayments* predicate.

## 8 Conclusions

We argue that rights and obligations, important in human economies (and often enforced through legal remedies) will be important in agent-mediated economies. We have defined a formal language that allows the specification of market mechanisms and a monitoring environment that allows for the automatic checking of rights and the enforcement of sanctions based on failed obligations. Simulations, often in the form of competitions such as the Trading Agent Competition [14, 11] have often been used to explore market space and drive research into agent-based reasoning within electronic markets. We hope that the formal approach taken here, in which the semantics of markets are exposed to agents, will also prove useful in providing a well-founded world for the development of principled methods in agent-based reasoning within electronic markets.

We feel that the design principles implemented by our framework capture the main underlying assumptions of many virtual market designs and implementations (like eBay), and thus provide an infrastructure for the specifications of future virtual markets, and simulation platforms for testing agent designs.

## References

1. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *Proc. of Conf. on AAMAS*, 2002.
2. S. Bikhchandani and J. Ostroy. The Package Assignment Model. *Journal of Economic Theory*, 107(2), December 2002.
3. Aspasia Daskalopulu and T. S. E. Maibaum. Towards electronic contract performance. In *DEXA Workshop*, 2001.
4. C. Dellarocas, M. Klein, and J. A. Rodriguez-Aguilar. An exception-handling architecture for open electronic marketplaces of contractnet software agents. In *Proc. ACM Conf. on Electronic Commerce (EC'00)*, 2000.
5. Oliver Hart. *Firms, Contracts, and Financial Structure*. Oxford Univ. Press, 1995.
6. Matthew O. Jackson. Mechanism theory. In *The Encyclopedia of Life Support Systems*. EOLSS Publishers, 2000.
7. S. L. Peyton Jones and J-M. Eber. How to write a financial contract. In J. Gibbons and O. de Moor, editors, *The Fun of Programming*. Palgrave Macmillan, 2003.
8. L. Thorne McCarty. Permissions and obligations. In *Proc. IJCAI*, 1983.
9. J. J. Meyer and R. J. Wieringa, editors. *Deontic Logic in Computer Science*. John Wiley & Sons, 1993.
10. Julian Padget and Russell Bradford. A  $\pi$ -calculus model of a Spanish fish market. In *Proc. of First Int. Workshop on Agent Mediated Electronic Trading*, May 1998.
11. N. M. Sadeh, R. Arunachalam, J. Eriksson, N. Finne, and S. Janson. TAC'03: A supply chain trading competition. *AI Magazine*, 24(1), Spring 2003.
12. Yao-Hua Tan and Walter Thoen. A logical model of directed obligations and permissions to support electronic contracting. *IJEC*, 3(2), 1999.
13. Jean Tirole. Incomplete contracts: Where do we stand? *Econometrica*, 67(4), 1999.
14. M. P. Wellman, P. R. Wurman, K. O'Malley, R. Banger, S. Lin, D. Reeves, and W. E. Walsh. Designing the market game for a trading agent competition. *IEEE Internet Computing*, 2001.
15. M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of AAMAS*, 3(3), 2000.