A formal market specification language for general trading agents

Michael Thielscher
Department of Computer Science
Technische Universität Dresden, Germany
mit@inf.tu-dresden.de

Dongmo Zhang
Intelligent Systems Laboratory
University of Western Sydney, Australia
dongmo@scm.uws.edu.au

Abstract

A contemporary grand AI challenge, *General Game Playing* is concerned with systems that can understand the rules of arbitrary games and learn to play these games well without human intervention. This principle has the great potential to bring to a new level artificially intelligent systems in other application areas as well. In this paper, our specific interest lies in *General Trading Agents*, which are able to understand the rules of unknown markets and then to actively participate in them without human intervention. To this end, we develop the existing Game Description Language (GDL) into a language that allows to formally describe arbitrary markets such that these specifications can be automatically processed by a computer. We present both syntax and a transition-based semantics for this Market Specification Language and illustrate its expressive power by presenting axiomatizations of several well-known auction types.

1 Introduction

A novel and challenging research problem for Artificial Intelligence, General Game Playing is concerned with the development of systems that learn to play previously unknown games solely on the basis of the rules of that game [Genesereth *et al.*, 2006]. The Game Description Language (GDL) [Love *et al.*, 2006] has been developed to formalize any finite, information-symmetric n-player game. As a declarative language, GDL supports specifications that are modular and easy to develop, understand, and maintain. At the same time, these specifications can be fully automatically processed, thus allowing to develop systems that are able to play the games with hitherto unknown rules without human intervention.

The idea behind General Game Playing—to build systems that are intelligent and flexible enough to negotiate an unknown environment solely on the basis of the rules which govern it—has the great potential to bring to a new level artificially intelligent systems in other application areas as well. Our specific interest lies in *General Trading Agents*. These should be able to understand the rules of unknown markets and then to actively participate in them without human intervention. As a first step towards the design and implementation of this new generation of trading agents, in this paper we suggest a modification and extension of GDL into a *Market Specification Language* (MSL) that allows to formally describe arbitrary markets such that these specification can be automatically processed by a computer.

While extending GDL, MSL inherits the crucial property of being a decidable subset of logic programming. This implies that General Trading Agents require just a simple, standard reasoning module to be able to understand and effectively process a given set of rules. Moreover, due to the close relation between the two languages, we expect that existing techniques for successful General Game Playing systems, such as [Kuhlmann *et al.*, 2006; Clune, 2007; Schiffel and Thielscher, 2007; Finnsson and Björnsson, 2008], can be readily used to design and implement General Trading Agents, too.

The rest of the paper is organized as follows. In the next section, we define a general market model in form of a finite state machine, where state transitions are triggered by messages coming from traders and actions executed by the market maker. In Section 3, we define the syntax of MSL as a modification and extension of GDL. We illustrate the use of the language by giving a fully formal specification of the well-known English auction type. In Section 4, we turn to the semantics of MSL and show how any set of rules can be understood as an axiomatic description of a market model. In Section 5 we give a precise definition of the execution of a market, and in Section 6 we provide two further descriptions of typical markets to illustrate the use and expressivity of MSL as a general market specification language. We conclude in Section 7.

2 Market model

Markets are a central topic in economics and finance. A market is an institution or mechanism that allows buyers (demanders) and sellers (suppliers) to exchange commodities such as goods, services, securities, and information. Generally, there are two distinct roles in every market: the *market maker* and *traders*. The market maker facilitates trade and enables the exchange of rights (ownership) of commodities at certain prices. Traders are market participants who utilize facilitates of the market to sell or buy goods and services.

As an example, consider a simple market in which only one commodity is traded. There is a set of traders (agents) who have registered in the market, and the market is manipulated by a market maker m. Each trader can be a buyer, a seller, or even both. A buyer can send the market maker *bids* and a seller can send in *asks*. A bid may be denoted as $b(a, q, p)$, representing that buyer a requests to buy q units of the good at a maximum price of p. Similarly, $s(a, q, p)$ may represent trader a wanting to sell q units of the commodity at a price no less than p. Bids and asks are often called *offers* (or orders).

Suppose that $o_1 = s(a_1, q_1, p_1)$ is an ask and $o_2 = b(a_2, q_2, p_2)$ is a bid. We say that (o_1, o_2) is a *match* if $p_1 \leq p_2$, that is, the ask price is no higher than the bid price. In such a case, $q = \min\{q_1, q_2\}$ units of goods can be sold at some price p such that $p_1 \leq p \leq p_2$. We call an offer o *cleared* at price p_0 if $o = b(a, q, p)$ and $p \geq p_0$, or if $o = s(a, q, p)$ and $p \leq p_0$.

There is a remarkable diversity in trading mechanisms that have been used in real-world markets. However, the most common trading mechanism is that of an auction or variations thereof [Friedman, 1993]. The common formalization of the trading mechanism of an auction consists of an interaction protocol and a set of market policies. An interaction protocol specifies the sequence of communication between traders and the market maker. Figure 1 shows the interaction protocol of the English auction [FIPA00031, 2001] as an example. While these graphical protocols can be viewed as a formalization of a trading mechanism, they cannot be fully automatically processed by a computer. Hence, they are unsuited as a specification language that can be understood by General Trading Agents without human intervention.

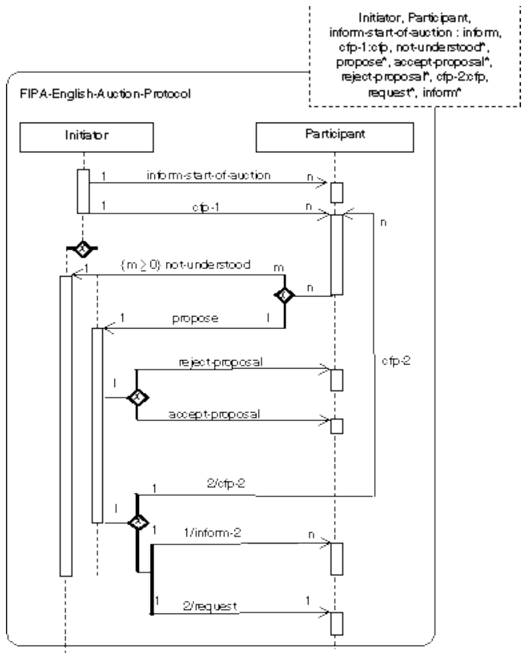
Initiator, Participant, inform-start-of-auction : inform, cfp-1:cfp, not-understood*, propose*, accept-proposal*, reject-proposal*, cfp-2:cfp, request*, inform*
FIPA-English-Auction-Protocol
Initiator
Participant
1
inform-start-of-auction
n
1
cfp-1
n
n
{m ≥ 0} not-understood
1
m
n
1
propose
l
reject-proposal
l
accept-proposal
cfp-2
1
2/cfp-2
l
1
1/inform-2
n
1
2/request
1

Figure 1: *FIPA English Auction Interaction Protocol.*

Market policies specify the rules that are used by the market maker to make decisions. These policies include accepting policy, matching policy, clearing policy, pricing policy, and so on. The accepting policy determines whether an offer is accepted or not. In many finical markets, market makers provide bid-ask quotations to traders to guide market price (called quote-driven [Madhavan, 1992]). The matching policy specifies how bids and asks are matched. For instance, the four-head matching policy always matches the highest matchable ask with the lowest matchable bid [Wurman *et al.*, 1998]. The clearing policy determines when matching is being made. An important distinction of auction types often made is that between continuous and periodic clearing policies [Madhavan, 1992]. In a continuous market, matching is made continuously whenever new offers arrive. In a periodic market, offers are accumulated for simultaneous execution at a single market clearing price.

A market is dynamic in the sense that whenever a new offer comes in or a transaction is executed, the market situation changes. Motivated by the formal semantics of GDL as a finite state machine [Schiffel and Thielscher, 2009], we propose to understand any market as a *state transition system*, in which the transitions are triggered by messages from the participating traders (say, bidding) and actions by the market maker (say, matching). To this end, a state transition system describing a market is given by the following constituents.

- s_0 —an initial state.
- T —a set of terminal states
- $l(a, s, t)$ —a relation defining a to be a possible action by the market maker in state s at time t (the *legality* relation).
- $u(a, m, s, t)$ —an *update* function defining the successor state when the market maker does action a or receives messages m in state s at time t [1].
- $o(a, m, s, t)$ —the messages (*output*) sent by the market maker when it does action a or receives messages m in state s at time t.

For the sake of simplicity, we assume that time is discretized and represented by the natural numbers. The time at the initial state is set to 0.

Take English auction as an example. In the initial state, the good for sale is unallocated and the bid pool is empty. The market maker *broadcasts* a call-for-proposals, which includes a so-called *reserve price* that thus becomes known to all participating traders. Whenever a new bid is received, the market maker updates the current state by the new highest bid price, provided the bid can be accepted. This continues for a fixed period of time, at the end of which the market maker announces the winner. The language defined in the following section will allow us to formally specify the actions, messages, and state transitions that characterize this type of auction, and in Section 3.2 we will give the full specification of English auction as an example.

[1]Note that the market state is updated only if the market maker performs an action or receives a message from a trader. See Section 4 for more details.

3 A Market Specification Language

Having defined a general market model, we proceed by showing how GDL can be modified to a suitable language that allows to specify an arbitrary market. A comparison of our market model with the game model shows that the *Market Specification Language* (MSL) needs to modify and extend GDL in the following ways.

- There is a special market maker, who acts (possibly nondeterministically) according to specified rules.
- Rather than making moves, traders send messages to the market maker.
- Rather than having complete state information, traders receive messages from the market maker according to specified rules.
- Time and real numbers, along with the standard arithmetic functions and relations, are pre-defined language elements.

In the following, we first define the formal syntax of MSL and then show how any set of MSL rules can be interpreted by our formal market model.

3.1 Syntax

Just like GDL, MSL is based on the standard syntax of clausal logic, including negation.

Definition 1

- *A* term *is either a variable, or a function symbol applied to terms as arguments (a* constant *is a function symbol with no argument).*
- *An* atom *is a predicate symbol applied to terms as arguments.*
- *A* literal *is an atom or its negation.*
- *A* clause *is an implication* $h \Leftarrow b_1 \wedge \ldots \wedge b_n$ *where* head h *is an atom and* body $b_1 \wedge \ldots \wedge b_n$ *a conjunction of literals* ($n \geq 0$)*.*

As a tailor-made specification language, MSL uses a few pre-defined predicate symbols. These are shown in Table 1 together with their informal meaning. In addition, we take both natural numbers and real numbers as pre-defined language elements. These are accompanied by the basic arithmetic functions $+, -, *, /, \mathtt{mod}$ and relations $<, \leq, =, \geq, >$ with the standard interpretation.

Throughout the paper, we adopt the Prolog convention according to which variables are denoted by uppercase letters and predicate and function symbols start with a lowercase letter. In the following, we illustrate the use of the keywords by giving a complete set of MSL rules describing a simple auction.

3.2 Example: English Auction

English auction is one of the most commonly used market models. Assume that there is a single item from a single seller. The market maker (auctioneer) accepts buyers to bid openly against one another, with each subsequent bid higher than the previous one. The market maker terminates the market either when a fixed clearing time is reached or when for three units of time no further bid is made. The following MSL rules describe this formally.

`trader(A)`	`A` is a trader
`message(A,M)`	trader `A` can send message `M`
`init(P)`	`P` holds in the initial state
`true(P)`	`P` holds in the current state
`next(P)`	`P` holds in the next state
`legal(A)`	market maker's action `A` is executable
`does(A)`	market maker does action `A`
`receive(A,M)`	receives message `M` from trader `A`
`send(A,M)`	send trader `A` message `M`
`time(T)`	`T` is the current time
`terminal`	the market is closed

Table 1: MSL keywords.

```
trader(a_1)⇐
...
trader(a_m)⇐

init(timer(0))⇐

accept(bid(A,P))⇐
    receive(A, my_bid(P)) ∧ ¬reject(P)
reject(P) ⇐ P≤RESERVE_PRICE
reject(P) ⇐ true(bid(A,P1)) ∧ P≤P1
reject(P) ⇐ receive(A, my_bid(P1)) ∧ P≤P1

reject(P) ⇐ true(timer(3))

legal(clearing(A,P))⇐
    true(timer(3)) ∧true(bid(A,P))
legal(call)⇐
    true(timer(T)) ∧T<3

next(B) ⇐ accept(B)
next(bid(A,P))⇐
    true(bid(A,P)) ∧ ¬outbid
next(timer(0)) ⇐ outbid
next(timer(T+1))⇐
    true(timer(T)) ∧ does(call) ∧¬outbid
outbid⇐accept(B)

message(A,my_bid(P)) ⇐
    trader(A) ∧ P≥ 0
send(A,bid_accepted(P))⇐
    accept(bid(A,P))
send(A,bid_rejected(P))⇐
    receive(A,my_bid(P)) ∧ reject(P)
send(A,call(T))⇐
    trader(A) ∧ true(timer(T)) ∧does(call)
send(A,best_price(P))⇐
    trader(A) ∧true(bid(A1,P))
send(A,winner(A1,P))⇐
    trader(A) ∧ does(clearing(A1,P))

terminal⇐ true(timer(4))
terminal⇐ time(MAX_TIME)
```

The above code in MSL specifies a set of market rules which compose an English auction. The rule for `accept(bid(A,P))`, in conjunction with the rules for `reject(P)`, specifies the accepting policy of the market maker: when it receives a bid from a trader (`receive(A,my_bid(P))`), the new bid price, `P`, must always be higher than the existing highest bid price (or, if it is the first bid, it needs to be no less than the given *RESERVE_PRICE*. Also, `P` must be higher than any other bid that arrives simultaneously, and the bid comes too late when the timer has reached 3 (it takes one unit of time for a bid to be processed after it is accepted).

The clearing policy is specified by the `legal` clause. We use a `call` action and a `timer` to facilitate the clearing action. The auctioneer makes a call for new bid whenever the market has not be cleared. If there is no outbid after three calls, the market is cleared.

The `next` clauses specify the state update, triggered either by a trader message, the `call` action, or the `clearing` action. Note that a bid takes effect at the next state after it is accepted (`next(B)` $\Leftarrow$ `accept(B)`). The statement `next(bid(A,P))` $\Leftarrow$ `true(bid(A,P))` $\wedge$ $\neg$ `outbid` is known as a *frame axiom* in reasoning about actions. It says that if there is no outbid, an existing bid remains in the next state. The timer is reseted whenever there is an outbid.

The `message` clause specifies the format and legality of incoming messages. The predicate `receive` indicates the received messages that are in the message pool. The clauses for `send` detail the outgoing messages. Together, these rules constitute a fully formal, logic-based specification of the interaction protocol of the market shown in Figure 2 [2]. Among the messages that the market maker sends to the traders, the notifications of acceptance and rejection are one-to-one. The others are announced to all traders (note the deference of trader variables).

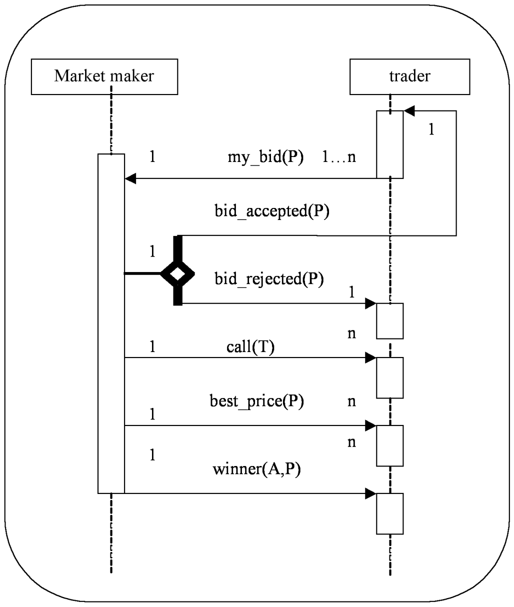

Figure 2: *Simplified interaction protocol of English auction.*

3.3 Syntactic restrictions

MSL imposes some syntactic restrictions on the use of the pre-defined predicates from Table 1 in much the same way GDL is restricted to ensure effective derivability of all information necessary for legal game play. These restrictions are based on the notion of a dependency graph for a given set of clauses (see, e.g., [Lloyd, 1987]).

Definition 2 *The* dependency graph *for a set G of clauses is a directed, labeled graph whose nodes are the predicate symbols that occur in G and where there is a* positive *edge $p \xrightarrow{+} q$ if G contains a clause $p(\vec{s}) \Leftarrow \ldots \wedge q(\vec{t}) \wedge \ldots$, and a* negative *edge $p \xrightarrow{-} q$ if G contains a clause $p(\vec{s}) \Leftarrow \ldots \wedge \neg q(\vec{t}) \wedge \ldots$.*

Definition 3 *A* valid MSL specification *is a finite set of clauses M that satisfies the following conditions.*

- `trader` *only appears in the head of clauses that have an empty body;*
- `init` *and* `message` *only appear as head of clauses and are not connected, in the dependency graph for G, to any of the keywords in Table 1 except for* `trader`*;*
- `true` *and* `time` *only appear in the body of clauses;*
- `does` *and* `receive` *only appear in the body of clauses and are not connected, in the dependency graph for G, to any of* `legal` *and* `terminal`*;*
- `next` *and* `send` *only appear as head of clauses.*

Moreover, in order to ensure effective derivability, M and the corresponding dependency graph Γ must obey the following restrictions.

1. *There are no cycles involving a negative edge in Γ (this is also known as being* stratified *[Apt* et al.*, 1987; van Gelder, 1989]);*
2. *Each variable in a clause occurs in at least one positive atom in the body (this is also known as being* allowed *[Lloyd and Topor, 1986]);*
3. *If p and q occur in a cycle in Γ and G contains a clause*

$$p(\vec{s}) \Leftarrow b_1(\vec{t_1}) \wedge \ldots \wedge q(v_1, \ldots, v_k) \wedge \ldots \wedge b_n(\vec{t_n})$$

then for every $i \in \{1, \ldots, k\}$,

- *v_i is variable-free, or*
- *v_i is one of $s_1, \ldots, s_m$ ($:= \vec{s}$), or*
- *v_i occurs in some $\vec{t_j}$ ($1 \leq j \leq n$) such that b_j does not occur in a cycle with p in Γ.*

Stratified logic programs are known to admit a specific *standard model*; we refer to [Apt *et al.*, 1987] for details and just mention the following properties:

[2]Note that it is slight simpler than the FIPA specification given in Figure 1.

1. To obtain the standard model, clauses with variables are replaced by their (possibly infinitely many) ground instances.
2. Clauses are interpreted as reverse implications.
3. The standard model is minimal while interpreting negation as non-derivability (the "negation-as-failure" principle [Clark, 1978]);

The further syntactic restrictions for MSL guarantee that agents can make effective use of a market specification by a simple derivation mechanism based on standard resolution for clausal logic (see again [Lloyd, 1987], for instance).

4 Semantics

We are now in a position to give a precise definition of how a valid MSL specification determines a market model. In the following, derivability means entailment via the standard model of a stratified set of clauses.

To begin with, the derivable instances of $\texttt{trader(A)}$ define the traders. The derivable instances of $\texttt{message(A,M)}$ define the possible messages $\texttt{M}$ for trader $\texttt{A}$ that are understood and processed by the market maker. The five components of the state transition system (cf. Section 2) are determined as follows.

1. The initial state s_0 is the set of all derivable instances of $\texttt{init(P)}$ along with timepoint 0.
2. In order to determine whether a state belongs to the set of terminal states T, this state (and the current timepoint) has to be encoded first using the keywords true and time. More precisely, let $s = \{p_1, \ldots, p_n\}$ be a finite set of terms (e.g., the derivable instances of $\texttt{init(P)}$ at the beginning) and $t \in \mathbb{N}$, then by $s_t^{\texttt{true}}$ we denote the clauses
$$\begin{array}{l}\texttt{true}(p_1) \Leftarrow \\ \ldots \\ \texttt{true}(p_n) \Leftarrow \\ \texttt{time}(t) \Leftarrow \end{array} \tag{1}$$
Let these be added to the given MSL specification, then state s at time t is terminal just in case terminal can be derived.
3. Similarly, the possible legal moves of the market maker in state s at time t—relation $l(a, s, t)$—is given by the derivable instances of $\texttt{legal}(a)$ after adding $s_t^{\texttt{true}}$ to the given market rules.
4. In order to determine a state update—function $u(a, M, s, t)$—the action a by the market maker and the messages M by the traders have to be encoded first, using the keywords does and receive. More precisely, let $M = \{(\alpha_1, m_1), \ldots, (\alpha_n, m_n)\}$ be a (possibly empty) set of (agent,message)-pairs and a an action by the market maker, then by $a^{\texttt{does}} \cup M^{\texttt{receive}}$ we denote the clauses
$$\begin{array}{l}\texttt{receive}(\alpha_1, m_1) \Leftarrow \\ \ldots \\ \texttt{receive}(\alpha_n, m_n) \Leftarrow \\ \texttt{does}(a) \Leftarrow \end{array} \tag{2}$$
(The market maker may also perform no action at the time of the state update, in which case the last clause is omitted.) Let these clauses, plus the clauses (1) for given state s and time t) be added to the given MSL specification, then the updated state $u(a, M, s, t)$ is given by all derivable instances of $\texttt{next}(p)$.
5. Similarly, the messages which the market maker sends to the traders when doing action a and receiving messages m in state s at time t—function $o(a, M, s, t)$—are given by the derivable instances of $\texttt{send}(a, m)$ after adding the clauses $s_t^{\texttt{true}}$ and $a^{\texttt{does}} \cup M^{\texttt{receive}}$ to the given market rules.

5 Market Model Execution

The execution of an MSL market subtly differs from the execution of a game model determined by a GDL specification, for two reasons. First, traders send messages asynchronously. Given discretized time, this means that at any timepoint a trader may or may not make a move. Second, while the conditions for the actions of the market maker are specified in the rules, the market maker may have the choice among several possibilities. This means that the market maker chooses exactly one among the possible legal actions whenever the triggering conditions for one or more of its actions are satisfied.

A possible execution of a market is therefore given by an evolving sequence of states
$$s_0 \to s_1 \to \ldots \to s_n$$
(where s_i denotes the state at time i) and messages
$$o_0, \ldots, o_{n-1}$$
(where o_i are the messages sent by the market maker at time i) such that

- s_0 is the initial state;
- $s_n \in T$ is the first terminal state in the sequence;
- let M be the set of all (agent,message)-pairs received by the market maker at time t, then
  - $s_{i+1} = s_i$ and o_i is empty if M is empty and no a satisfies $l(a, s_i, i)$,
  - $s_{i+1} = u(a, M, s, i)$ and $o_i = o(a, M, s, i)$ if M is not empty and/or a can be selected that satisfies $l(a, s_i, i)$.

6 Specifications of Typical Markets

In this section, we present three further examples of market specifications given in MSL in order to illustrate its general expressivity: one for *Sealed-Bid Auction*, one for *Call Markets* and the other for *Continuous Double Auction*. All three are commonly used in the real word.

6.1 Sealed-Bid Auction

Sealed-bid auction is one of the simplest market mechanisms used in the real world. It differs from English auction in that traders' bids are concealed from each other. The following

MSL code specifies a first-price sealed-bid auction where the highest bidder gets the award and pays the amount he bid.

```
trader(a_1)⇐
...
trader(a_m)⇐

accept(bid(A,P)) ⇐
  receive(A,my_bid(P)) ∧
  time(1)

legal(clearing(A,P)) ⇐
  true(bid(A,P)) ∧
  bestbid(P) ∧
  time(2)

next(bid(A,P)) ⇐ accept(bid(A,P))
next(bid(A,P)) ⇐ true(bid(A,P))

bestbid(P) ⇐
  true(bid(A,P)) ∧ ¬outbid(P)
outbid(P) ⇐
  true(bid(A,P1)) ∧ P1 > P

message(A,my_bid(P)) ⇐ trader(A) ∧ P≥ 0

send(A, call_for_bid) ⇐
  trader(A) ∧ time(0)
send(A, bid_received(P)) ⇐
  receive(A,my_bid(P))
send(A, winner(A1,P)) ⇐
  trader(A) ∧ does(clearing(A1,P))

terminal ⇐ time(3).
```

At time 0, the market maker sends a call-for-bid to all traders. Only the bids that are received at time 1 are accepted. Once a bid is accepted, a private acknowledgement is sent to the bidder who submitted the bid. The auction is cleared at time 2. The trader who sent in the highest bid wins the auction. Note that if there are more than one highest bids, the market maker choose one of them. The auction terminates at time 3.

We remark that although the market specification is public information for all market participants, the individual bids are private information, which can only be seen by the respective sender and the market maker. This is significant different from General Game Playing, in which each player's move is announced to every player. In the above example, the call-for-bid and winner announcement are sent to every trader but the acknowledgment of a bid is sent only to the trader who submitted the bid.

6.2 Call market

A call market, also known as clearing house (CH), is a market institution in which each transaction takes place at predetermined intervals and where all of the bids and asks are aggregated and transacted at once. The market maker determines the market clearing price based on the bids and asks received during this period [Amihud and Mendelson, 1987]. A call market is actually a type of periodic double auction. The following rules specify a simplified call market with a single type of commodities.

```
trader(a_1)⇐
...
trader(a_m)⇐

accept(ask(A,Q,P)) ⇐
  receive(A,my_ask(Q,P)) ∧ trader(A)
accept(bid(A,Q,P))⇐
  receive(A,my_bid(Q,P)) ∧ trader(A)

legal(clearing(P)) ⇐
  time(T) ∧
  T mod TIME_INTERVAL = TIME_INTERVAL-1 ∧
  PRICING_POLICY(P)

cleared(A,Q,P)⇐
  does(clearing(P1)) ∧ true(bid(A,Q,P))∧
  P ≥ P1
cleared(A,Q,P) ⇐
  does(clearing(P1)) ∧ true(ask(A,Q,P)) ∧
  P ≤ P1

next(B) ⇐ accept(B)
next(ask(A,Q,P)) ⇐
  true(ask(A,Q,P)) ∧ ¬cleared(A,Q,P)
next(bid(A,Q,P)) ⇐
  true(bid(A,Q,P)) ∧ ¬ cleared(A,Q,P)

message(A,my_ask(Q,P)) ⇐
  trader(A) ∧ Q>0 ∧ P≥0
message(A,my_bid(Q,P)) ⇐
  trader(A) ∧ Q>0 ∧ P≥0

send(A,quote(P)) ⇐
  trader(A) ∧ does(clearing(P))
send(A,cleared(Q,P)) ⇐
  cleared(A,Q,P)

terminal ⇐ time(MAX_TIME+1)
```

The specification shows that the market maker accepts any non-negative incoming bids and asks (accepting policy) and clears the market periodically using a single price for both bids and asks. Note that we did not specify the pricing policy `PRICING_POLICY(P)`. The market maker can keep it as private information (it could announce a quote based on the previous business period, as it is shown in the above code). The market maker can also publicize its pricing policy by releasing the algorithms of the function `PRICING_POLICY(P)`.

We remark that the specification has been simplified in that we did not consider limited orders: no restrictions have been put on quantity or price of incoming offers.

6.3 Continuous double auction

Continuous double auction (CDA) is the most commonly used market model in financial markets. In a continuous auction market, trading is carried out continuously through the market maker who collects bids and asks from traders and matches existing orders whenever possible.

```
trader(a_1)⇐
...
trader(a_m)⇐
```

```
accepts(ask(A,Q,P)) ⇐
  receive(A,my_ask(Q,P)) ∧
  P<ASK_QUOTE
accepts(bid(Id,A,Q,P)) ⇐
  receive(A,my_bid(Q,P))∧
  P>BID_QUOTE

legal(match(A1,Q1,P1,A2,Q2,P2,Q,P)) ⇐
  true(ask(A1,Q1,P1)) ∧
  true(bid(A2,Q2,P2)) ∧
  P1 ≤ P2 ∧ minimum(Q1,Q2,Q) ∧
  P1 ≤ P ∧ P ≤ P2

minimum(Q1,Q2,Q1) ⇐ Q1 ≤ Q2
minimum(Q1,Q2,Q2) ⇐ Q1 > Q2

cleared(ask(A1,Q1,P1)) ⇐
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P))
cleared(bid(A2,Q2,P2)) ⇐
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P))
next(O) ⇐ accepts(O)

next(O) ⇐
  true(O) ∧ ¬ cleared(O)

next(ask(A1,Q1-Q,P1)) ⇐
  true(ask(A1,Q1,P1))∧
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P))∧
  Q1 > Q
next(bid(A2,Q2-Q,P2)) ⇐
  true(bid(A2,Q,P2)) ∧
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P)) ∧
  Q2 > Q

message(A,my_ask(Q,P)) ⇐
  trader(A) ∧ Q>0 ∧ P>0
message(A,my_bid(Q,P)) ⇐
  trader(A) ∧ Q>0 ∧ P>0

send(A1,clearing(Q1,P1,Q,P)) ⇐
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P))
send(A2,clearing(Q2,P2,Q,P)) ⇐
  does(match(A1,Q1,P1,A2,Q2,P2,Q,P))

terminal ⇐ time(MAX_TIME+1)
```

According to this specification, the market maker sets an ASK_QUOTE and a BID_QUOTE as the threshold for accepting bids and asks. Similar to the pricing policy in a call market, the market maker can either keep the quotes as private information or release them by providing the algorithms for calculating the quotes.

Once an offer (bid or ask) is accepted, it is added to the next state (offer pool). The market maker continuously searches for possible matches among the existing offers. For each match, a fully satisfied offer is removed from the state while partially satisfied offers remain in the pool with the residual quantity. As for the preceding specification for call markets, the actual pricing policy is left underspecified.

7 Summary

We have introduced a general market specification language (MSL) by modifying and extending the Game Description Language that is used in the context of General Game Playing to formalize the rules of arbitrary games in a machine-processable fashion. We have specified syntax and semantics for MSL, and we have given formalizations of a set of standard auction types to illustrate the usefulness of this language as a formal basis for General Trading Agents .

This is an on-goning work with many aspects that have not been fully investigated. Firstly, the semantics of the interaction between the market maker and traders cannot be fully specified in MSL. As a general issue, there are a verity of formal languages that have been proposed for specifying agent communication protocols [Endriss *et al.*, 2003; Labrou and Finin, 1997; Mcginnis and Miller, 2008]. Although these languages are not especially designed for market specifications, the communication primitives that have been intensively discussed in the context of agent communication languages, such as tell, inform, ask, and etc., can be introduced to specify interaction in a market. Secondly, all examples we presented in this paper are concerned with the exchange of a single good. However, we strongly believe that the language is sufficiently expressive to describe more complicated markets, such as combinatorial auctions [Boutilier and Hoos, 2001; Cerquides *et al.*, 2007; Uckelman and Endriss, 2008]. Thirdly, the design and implementation of market policies for different business demand, especially e-business, has been intensively investigated in recent years [Wurman *et al.*, 2001; Niu *et al.*, 2008]. However, design market rules with a purely logical, programmable language has not been studied in general.

There are a variety of potential applications of MSL to be investigated. Firstly, the rules of an e-market can be specified in MSL and made publicly available. With a simple logical reasoning module, any autonomous trading agent can understand the specification and enter the market for business. Secondly, a market can change its rules dynamically as long as the new market specification is sent to all participating traders. Thirdly, the language can be used for designing market games such as the Trading Agent Competition (TAC) [Wellman *et al.*, 2007; Niu *et al.*, 2008]. MSL provides the basis for turning this competition into a much more challenging one where the detailed problem specification is no longer revealed in advance, requiring the participating agents—or teams of agents—to compete in a previously unknown setting.

ACKNOWLEDGMENTS

This research was partially supported by the Australian Research Council through Discovery Project DP0988750 and by *Deutsche Forschungsgemeinschaft* under Contract TH 541/16-1. Thanks to the anonymous reviewers for their valuable comments.

References

[Amihud and Mendelson, 1987] Y. Amihud and H. Mendelson. Trading mechanisms and stock returns: an empirical investigation. *The Journal of Finance*, Vol XLII, No 3, 1987.

[Apt *et al.*, 1987] Krzysztof Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1987.

[Boutilier and Hoos, 2001] Craig Boutilier and Holger H. Hoos. Bidding languages for combinatorial auctions. In *IJCAI*, pages 1211–1217, 2001.

[Cerquides *et al.*, 2007] Jesús Cerquides, Ulle Endriss, Andrea Giovannucci, and Juan A. Rodríguez-Aguilar. Bidding languages and winner determination for mixed multi-unit combinatorial auctions. In *IJCAI*, pages 1221–1226, 2007.

[Clark, 1978] Keith Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[Clune, 2007] Jim Clune. Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 1134–1139, Vancouver, July 2007. AAAI Press.

[Endriss *et al.*, 2003] Ulrich Endriss, Nicolas Maudet, Fariba Sadri, and Francesca Toni. Protocol conformance for logic-based agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 679–684. Morgan Kaufmann Publishers, 2003.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 259–264, Chicago, July 2008. AAAI Press.

[FIPA00031, 2001] FIPA00031 FIPA English Auction Interaction Protocol Specification. Foundation for Intelligent Physical Agents, 2001. `http://www.fipa.org/specs/fipa00031/`

[Friedman, 1993] D. Friedman. The double auction institution: A survey. In D. Friedman and J. Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, chapter 1, pages 325. Perseus Publishing, Cambridge, MA, 1993.

[Genesereth *et al.*, 2006] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing. *AI Magazine*, 26(2):73–84, 2006.

[Kuhlmann *et al.*, 2006] Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic heuristic construction in a complete general game player. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 1457–1462, Boston, July 2006. AAAI Press.

[Labrou and Finin, 1997] Yannis Labrou and Timothy W. Finin. Semantics and conversations for an agent communication language. In *IJCAI (1)*, pages 584–591, 1997.

[Lloyd and Topor, 1986] John Lloyd and R. Topor. A basis for deductive database systems II. *Journal of Logic Programming*, 3(1):55–67, 1986.

[Lloyd, 1987] John Lloyd. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition, 1987.

[Love *et al.*, 2006] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Stanford Logic Group, Computer Science Department, Stanford University, 353 Serra Mall, Stanford, CA 94305, 2006. Available at: `games.stanford.edu`.

[Madhavan, 1992] A. Madhavan. Trading mechanisms in securities markets. *The Journal of Finance*, Vol. XLVII, No. 2, 607-641, 1992.

[Mcginnis and Miller, 2008] Jarred Mcginnis and Tim Miller. Amongst first-class protocols. In *Engineering Societies in the Agents World VIII*, page 208223. Springer, 2008.

[Niu *et al.*, 2008] Jinzhong Niu, Kai Cai, Enrico Gerding, Peter McBurney, and Simon Parsons. Characterizing effective auction mechanisms: Insights from the 2007 TAC Mechanism Design Competition. In Padgham, Parkes, M ü ller, and Parsons, editors, *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 1079–1086, 2008.

[Schiffel and Thielscher, 2007] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 1191–1196, Vancouver, July 2007. AAAI Press.

[Schiffel and Thielscher, 2009] S. Schiffel and M. Thielscher. Specifying multiagent systems in the game description language. In J. Filipe, A. Fred, and B. Sharp, editors, *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 21–28, Porto, 2009.

[Uckelman and Endriss, 2008] Joel Uckelman and Ulle Endriss. Winner determination in combinatorial auctions with logic-based bidding languages. In *AAMAS (3)*, pages 1617–1620, 2008.

[van Gelder, 1989] A. van Gelder. The alternating fixpoint of logic programs with negation. In *Proceedings of the 8th Symposium on Principles of Database Systems*, pages 1–10. ACM SIGACT-SIGMOD, 1989.

[Wurman *et al.*, 1998] P. R. Wurman, W.E. Walsh and M. Wellman, Flexible double auctions for electronic commerce: theory and implementation, *Decision Support Systems* 24(1), 17-27, 1998.

[Wellman *et al.*, 2007] M. Wellman, A. Greenwald, and P. Stone, *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*, MIT Press.

[Wurman *et al.*, 2001] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A parameterization of the auction design space. *Games and Economic Behavior*, pages 304–338, 2001.