From General Game Descriptions to a Market Specification Language for General Trading Agents

Michael Thielscher [1] and Dongmo Zhang [2]

[1] The University of New South Wales, Australia,
mit@cse.unsw.edu.au
[2] University of Western Sydney, Australia,
dongmo@scm.uws.edu.au

Abstract. The idea behind General Game Playing is to build systems that, instead of being programmed for one specific task, are intelligent and flexible enough to negotiate an unknown environment solely on the basis of the rules which govern it. In this paper, we argue that this principle 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, which are able to understand the rules of unknown markets and then to actively participate in them without human intervention. To this end, we extend the general Game Description Language into a language that allows to formally describe arbitrary markets in such a way 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.

Key words: general trading agents, market specification language

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 [1]. The Game Description Language (GDL) [2] 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 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 in such a way that these specification can be automatically processed by a computer.

GDL is designed to describe discrete games of complete and symmetric information. A suitable description language for markets requires two principal additions to GDL:

– information asymmetry
– asynchronous actions

MSL accounts for information asymmetry by including the special role of an (all-knowing) *market maker* along with a system of message passing. The latter allows to exchange private information between the market maker and the participating agents (traders), which results in incomplete, asymmetric information. In addition, the actions by the market maker may be underspecified, which results in imperfect information. In order to account for asynchronous actions by the market participants, MSL uses explicit (discrete) time.

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 [3, 4, 5, 6], 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 three 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

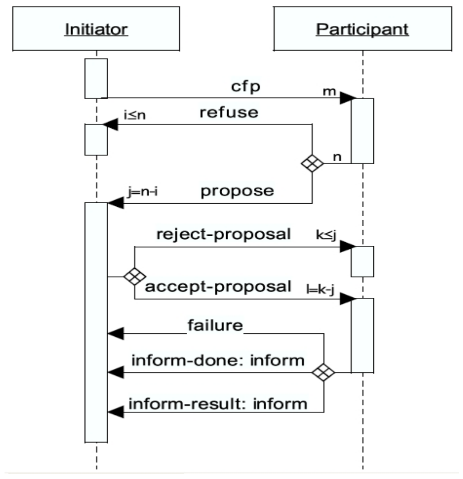

Fig. 1. *FIPA English Auction Interaction Protocol.*

commodities at certain prices. Traders are market participants who utilize the facilities of the market to sell or buy goods and services.

As an example, consider a very basic 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 that trader a wants 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 isn't 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 [7]. 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 sequential communication between traders and the market maker. As an example, Figure 1 shows the interaction protocol of the English auction [8]. 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.

Market policies specify the rules that are used by the market maker to make decisions. These include an 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 [9]). 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 [10]. 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 [9]. 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 [11], 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 asking) 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 and receives messages m in state s at time t.
– $o(a,m,s,t)$ —the messages (*output*) sent by the market maker when it does action a and 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.

State transition systems are sufficiently abstract to be used as a universal model for any market and the rules that govern it. 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 can then *broadcast* 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 can update the current state by the new highest bid price, provided the given market-specific acceptance conditions are met. This continues for a fixed period of time, at the end of which the market maker can announce the winner. The language defined in the following section will allow us to formally specify the actions, messages, and state transitions that characterize this or any other type of auction, and in Section 3.2 we will give the full specification of English auction as an example.

3 A Market Specification Language

Having defined an abstract 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. In GDL, all roles are treated symmetrically.
- Rather than making moves, traders send private messages to the market maker.
- Rather than maintaining complete state information, traders receive (private) 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. Note that time and arithmetic operations are not standard components in GDL.

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.

`trader(A)`	`A` is a trader
`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 can do action `A`
`does(A)`	market maker does action `A`
`message(A,M)`	trader `A` can send message `M`
`receive(A,M)`	receiving message `M` from trader `A`
`send(A,M)`	sending message `M` to trader `A`
`time(T)`	`T` is the current time
`terminal`	the market is closed

Table 1. MSL keywords.

In addition, we take both natural numbers and real numbers as pre-defined language elements. These are accompanied by the basic arithmetic functions $+, -, *, /,$ `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 very 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 specifies the auction mechanism formally.

```
trader(a_1) ⇐
...
trader(a_m) ⇐

init(counter(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(counter(3))

legal(clearing(A,P)) ⇐ true(counter(3)) ∧ true(bid(A,P))
legal(call)          ⇐ true(counter(C)) ∧ C < 3

next(B)            ⇐ accept(B)
next(bid(A,P))     ⇐ true(bid(A,P)) ∧ ¬outbid
next(counter(0))   ⇐ outbid
next(counter(C+1)) ⇐ true(counter(C)) ∧ 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(C))        ⇐ trader(A) ∧ true(counter(C)) ∧ 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(counter(4))
terminal ⇐ time(MAX_TIME)
```

The intuition behind this complete and fully formal specification of English auction is as follows. A state of this market is just a single `bid(A,P)` instance (the currently highest bid, initially none) along with an instance of the special feature `counter(C)` (modeling the usual calls "1" $\rightarrow$ "2" $\rightarrow$ "3" in an English auction, initially 0).

The rule for `accept(bid(A,P))`, in conjunction with the rules for `reject(P)`, specifies the accepting policy of the market: when a bid from a trader is received (`receive(A,my_bid(P))`), then the new bid price, `P`, must 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 counter has reached 3 (it takes one unit of time for a bid to be processed after accepting it).

The clearing policy is specified via predicate `legal`. The auctioneer makes a call for new bids whenever the market has not been cleared. Once a bid is not overbid after three calls, the market maker clears the market (the first `legal(A)` clause). Otherwise, the market maker issues the next call according to the second clause for `legal(A)`.

The `next` clauses specify the state update, triggered either by a trader message, the `call` action, or the `clearing` action: an accepted bid becomes the new highest one, whereas if none gets accepted then the previously highest bid stays. The counter is set back to 0 whenever a new bid is accepted, otherwise its value is incremented upon every `call` action.

The `message` clause specifies the format and legality of messages that can be sent to the market maker. The clauses for `send` detail the outgoing messages. Finally, the two clauses for `terminal` describe the conditions (on the current state and the global time) for the market to get closed.

Altogether, these rules constitute a fully formal, logic-based specification of the interaction protocol of the market shown in Figure 2. (Note that, for the purpose of illustration, this is a slightly simplified version of the FIPA specification given in Figure 1).

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., [12]).

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 $p(\mathbf{s}) \Leftarrow \ldots \wedge q(\mathbf{t}) \wedge \ldots$, and a* negative *edge $p \xrightarrow{-} q$ if G contains a clause $p(\mathbf{s}) \Leftarrow \ldots \wedge \neg q(\mathbf{t}) \wedge \ldots$.*

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

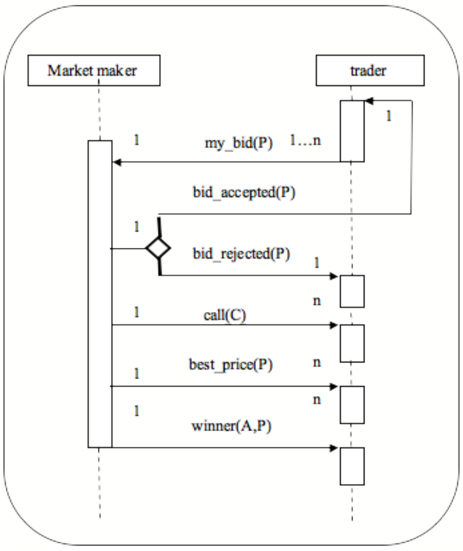

Fig. 2. *Simplified interaction protocol of English auction.*

– `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* `legal` *or* `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 [13, 14]*);*
2. *Each variable in a clause occurs in at least one positive atom in the body (this is also known as being* allowed [15]*);*
3. *If* p *and* q *occur in a cycle in* Γ *and* G *contains a clause*

$$p(\mathbf{s}) \Leftarrow b_1(\mathbf{t}_1) \wedge \ldots \wedge q(v_1, \ldots, v_k) \wedge \ldots \wedge b_n(\mathbf{t}_n)$$

then for every $i \in \{1, \ldots, k\}$,
– v_i *is variable-free, or*
– v_i *is one of* $s_1, \ldots, s_m$ *(*$:= \mathbf{s}$*), or*
– v_i *occurs in some* $\mathbf{t}_j$ *(*$1 \leq j \leq n$*) 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 [13] for details and just mention the following properties:

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 [16]).

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, e.g., [12]).

4 Semantics

We are now in a position to formally define 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 `trader(A)` define the traders. The derivable instances of `message(A,M)` define the possible messages `M` for trader `A` that are understood and processed by the market maker. The five components of the state transition system (cf. Section 2) are then formally determined as follows.

1. The initial state s_0 is the set of all derivable instances of `init(P)` along with timepoint 0.
2. In order to determine whether a state belongs to the set of terminal states T, this state (including 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 `init(P)` at the beginning) and $t \in \mathbb{N}$, then by $s_t^{\texttt{true}}$ we denote the facts

$$\begin{array}{l} \texttt{true}(p_1) \Leftarrow \\ \ldots \\ \texttt{true}(p_n) \Leftarrow \\ \texttt{time}(t) \Leftarrow \end{array} \qquad (1)$$

   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 $l(a, s, t)$—are given by the derivable instances of `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 from 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

$$\begin{array}{l}\texttt{receive}(\alpha_1, m_1) \Leftarrow \\ \dots \\ \texttt{receive}(\alpha_n, m_n) \Leftarrow \\ \texttt{does}(a) \Leftarrow \end{array} \tag{2}$$

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 `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 `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 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 time-point 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

$$s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$$

(where s_i denotes the state at time i) and messages

$$o_0, \dots, o_{n-1}$$

(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 i, then
  - $s_{i+1} = s_i$ and o_i is empty if M is empty and no a satisfies $l(a, s_i, i)$,
  - $s_{i+1} = u(a, M, s_i, i)$ and $o_i = o(a, M, s_i, i)$ if M is not empty and/or an action a can be selected (by the market maker) that satisfies $l(a, s_i, i)$.

It is worth pointing out that, while all traders start with the same, complete information about the initial state, the messages received and sent by the market maker are private. This will usually result in asymmetric, incomplete information about later states. Moreover, if the market maker has more than one legal action in a state, it makes an arbitrary selection (from the trader's point of view), which results in imperfect knowledge.

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 of them are commonly used in financial markets for exchanging securities or futures.

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 it. The auction is cleared at time 2. The trader who sent in the highest bid wins the auction. Note that if there is more than one highest bid, the market maker choose one of them. The auction terminates at time 3.

We remark that although the market specification is known to all market participants, the individual bids are private information, which can only be seen by the respective sender and the market maker. This is fundamentally different from General Game Playing, where 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 it.

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 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 [17]. 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 ∧ P>0

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 incoming bids and asks (accepting policy) and clears the market periodically using a single price. Note that the clearing price is public information broadcast to all agents at all times. However, the information of how the market maker decides the market price (pricing policy) is not given, which can be seen from the fact that the actual clearing price, viz. the argument in `clearing(P)`, is not fully specified. From the perspective of the participating agents, this action is nondeterministic. (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 an offer.)

6.3 Continuous double auction

Continuous double auction (CDA) is the most commonly used market model in financial markets like the New York Stock Exchange. Different from a call market, trading in a continuous auction market 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(Offer) ⇐  accepts(Offer)
next(Offer) ⇐  true(Offer) ∧  ¬ cleared(Offer)
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 (offers). 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 appears in 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 in 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 three standard auction types to illustrate the usefulness of this language as a foundation for General Trading Agents.

There is a variety of potential applications of MSL. 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) [18, 19]. 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.

There have been a number of attempts at building electronic markets under general market specifications [20, 21, 22]. However, none of them uses a logical language to describe market rules even though logical approach has been widely applied to the specification of extensive games or bargaining games [23, 24, 25].

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 the traders cannot be fully specified in MSL. As a general issue, there are a variety of formal languages that have been proposed for specifying agent communication protocols [26, 27, 28]. 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 [29, 30, 31]. Thirdly, the design and implementation of market policies for different business demand, especially e-business, has been intensively investigated in recent years [22, 18]. However, the design of market rules using a purely logical and machine-processable language has not been studied in general.

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.

References

1. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the AAAI competition. AI Magazine **26**(2) (2005) 62–72
2. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: 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`.
3. Kuhlmann, G., Dresner, K., Stone, P.: Automatic heuristic construction in a complete general game player. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Boston, AAAI Press (July 2006) 1457–1462
4. Clune, J.: Heuristic evaluation functions for general game playing. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Vancouver, AAAI Press (July 2007) 1134–1139
5. Schiffel, S., Thielscher, M.: Fluxplayer: A successful general game player. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Vancouver, AAAI Press (July 2007) 1191–1196
6. Björnsson, Y., Finnsson, H.: CADIAPLAYER: A simulation-based general game player. IEEE Transactions on Computational Intelligence and AI in Games **1**(1) (2009) 4–15
7. Friedman, D.: The double auction market institution: A survey. The Double Auction Market: Institutions, Theories, and Evidence (1993) 3–25
8. FIPA00031: Fipa english auction interaction protocol specification. Technical report, Foundation for Intelligent Physical Agents (2001)
9. Madhavan, A.: Trading mechanisms in securities markets. The Journal of Finance **XLVII**(2) (1992) 607–641
10. Wurman, P.R., Walsh, W.E., Wellman, M.P.: Flexible double auctions for electronic commerce: theory and implementation. Decision Support Systems **24**(1) (November 1998) 17–27
11. Schiffel, S., Thielscher, M.: A multiagent semantics for the game description language. In Filipe, J., Fred, A., Sharp, B., eds.: Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART), Porto, Springer (2009) 44–55
12. Lloyd, J.: Foundations of Logic Programming. second, extended edn. Series Symbolic Computation. Springer (1987)
13. Apt, K., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1987) 89–148
14. van Gelder, A.: The alternating fixpoint of logic programs with negation. In: Proceedings of the 8th Symposium on Principles of Database Systems, ACM SIGACT-SIGMOD (1989) 1–10
15. Lloyd, J., Topor, R.: A basis for deductive database systems II. Journal of Logic Programming **3**(1) (1986) 55–67
16. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and Data Bases. Plenum Press (1978) 293–322

17. Amihud, Y., Mendelson, H.: Trading mechanisms and stock returns: An empirical investigation. Journal of Finance **42**(3) (July 1987) 533–53
18. Niu, J., Cai, K., Gerding, E., McBurney, P., Parsons, S.: Characterizing effective auction mechanisms: Insights from the 2007 TAC Mechanism Design Competition. In Padgham, Parkes, M ü ller, Parsons, eds.: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems. (2008) 1079–1086
19. Wellman, M.P., Greenwald, A., Stone, P.: Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition. MIT Press (2007)
20. Esteva, M., de la Cruz, D., Sierra, C.: Islander: en electronic institutions editor. In Cristiano Castelfranchi, W.L.J., ed.: Proceedings of the First International Joint Conference on Auton omous Agents and Multiagent Systems. Volume 3., ACM PRESS (2002) 1045–1052
21. Fasli, M., Michalakopoulos, M.: e-game: A platform for developing auction-based market simulations. Decision Support Systems **44**(2) (2008) 469–481
22. Wurman, P.R., Wellman, M.P., Walsh, W.E.: A parameterization of the auction design space. Games and Economic Behavior **35**(1/2) (2001) 304–338
23. Koller, D., Pfeffer, A.: Representations and solutions for game-theoretic problems. Artificial Intelligence **94** (1997) 167–215
24. Kraus, S., Sycara, K., Evenchik, A.: Reaching agreements through argumentation: a logical model and implementation. Artificial Intelligence **104** (1998) 1–69
25. Zhang, D.: Reasoning about bargaining situations. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07). (2007) 154–159
26. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Protocol conformance for logic-based agents. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), Morgan Kaufmann Publishers (2003) 679–684
27. Labrou, Y., Finin, T.: Semantics and conversations for an agent communication language. In: Readings in agents. Morgan Kaufmann Publishers Inc. (1998) 235–242
28. Mcginnis, J., Miller, T.: Amongst first-class protocols. In: Engineering Societies in the Agents World VIII, Springer (2008) 208–223
29. Boutilier, C., Hoos, H.H.: Bidding languages for combinatorial auctions. In: Proceedings of the 17th international joint conference on Artificial intelligence (IJCAI'01), San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 1211–1217
30. Cerquides, J., Endriss, U., Giovannucci, A., Rodríguez-Aguilar, J.A.: Bidding languages and winner determination for mixed multi-unit combinatorial auctions. In: Proceedings of the 20th international joint conference on Artifical intelligence (IJCAI'07), San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2007) 1221–1226
31. Uckelman, J., Endriss, U.: Winner determination in combinatorial auctions with logic-based bidding languages. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS '08). (2008) 1617–1620