Model-Checking Auctions, Coalitions and Trust[*]

Matt Webster, Louise Dennis, and Michael Fisher

Department of Computer Science, University of Liverpool, Liverpool, UK

EMAIL: {M.P.Webster,L.A.Dennis,MFisher}@liverpool.ac.uk

January 2009

Abstract

In this paper we tackle the verification of basic auction models that are at the heart of many market-based multi-agent systems. Specifically, we program auctions in a BDI-based programming language and then use agent model checking to verify logical properties concerning time, beliefs and goals within the multi-agent system. The basic auction model is then extended with coalition formation and trust, and verification of these aspects is carried out.

1 Introduction

The basic idea of an *auction* [36, 26] is at the heart of many multi-agent scenarios [7]. Not only are auctions central to e-commerce applications [12, 21], but are implicit within many *market-based* approaches to agent computation. These include resource-allocation [23], telecommunications [22], electricity supply management [13], agent mobility [8], task allocation and scheduling [38, 33, 14]. However, although much work has been carried out on the deep analysis of auction mechanisms, such as through formal mechanism design [40], the analysis of *implementations* of auction mechanisms has lagged behind. While there has been *some* work on the formal verification of auction implementations, such as [19], this has been lacking an agent perspective. Thus, the more sophisticated agent aspects such as *goals*, *intentions*, *beliefs* and *deliberation* have not been verified within an auction context. It is this we aim to tackle in this paper.

We aim to implement simple and intuitive auction models in a BDI-based programming language and then apply agent verification techniques to these implementations. Although initially used sparsely, the BDI aspects will become more relevant as we extend and develop the auctions to incorporate coalition formation, trust, and dynamic aspects of both these properties. In addition, although the examples are small, they are clearly representative of many larger, and more sophisticated, multi-agent systems. The verification will be carried out using an agent model-checking system [4] and the (simple) properties verified will be given in a logic of belief, goals and time.

In carrying out this work, we point the way towards the future formal verification of sophisticated market-based approaches. As such approaches are increasingly used in both safety critical and business critical scenarios, deep analysis such as this will be vital in establishing the reliability, security and efficacy of such systems.

[*]Work supported by EPSRC (UK) grant EP/D052548.

1.1 Background: Agent Programming Languages

As the concept of an "agent" becomes more popular, so the variety of programming languages based upon this concept increases. These *agent-oriented* programming languages range from minimal extensions of Java through to logic-based languages for "intelligent" agents [3, 20]. In our work, we are particularly concerned with approaches based on *rational agent theories* [41], primarily the *BDI theory* developed by Rao and Georgeff [31]. Such languages not only incorporate the autonomous behaviour required for the agent concept, but also provide sophisticated mechanisms for instigating, controlling, and reasoning about goal-directed behaviours.

In the area of autonomous agents and multi-agent systems, AgentSpeak is one of the best known agent-oriented programming languages based on the BDI architecture. It is a logic-based agent-oriented programming language introduced by Rao [30], and subsequently extended and formalised in a series of papers by Bordini, Hübner, and colleagues[1]. Various *ad hoc* implementations of BDI-based (or "goal-directed") systems exist, but one important characteristic of AgentSpeak is its formal semantics. This aspect is particularly important, since BDI approaches are increasingly used in complex, critical applications such as space exploration [28, 9, 35]. Once we have a BDI language with clear formal semantics, then the possibility of applying *formal verification* techniques, especially within critical applications, is opened up. In particular, we utilise the GWENDOLEN programming language [15], which is small, tailored for use with our verification system, and essentially based on AgentSpeak.

1.2 Background: Formal Verification and Model Checking

By *formal verification* we mean carrying out a mathematical analysis in order to assess all the potential behaviours within a system. The behavioural requirements we have of complex systems can be specified using formulae from an appropriate formal logic. Importantly, the formal logic used can incorporate a wide range of concepts matching the view of the system being described, for example *time* for dynamic/evolving systems, *probability* for systems incorporating uncertainty, *goals* for autonomous systems, etc. This gives very great flexibility in the descriptive language that can be used.

Given such a specification, we can check this against models/views of the system under consideration in a number of ways. The most popular of these is *model checking* [10, 24], where the specification is checked against all possible executions of the system; if there are a finite number of such executions, then this check can often be carried out automatically. Indeed, the verification, via model checking, of both hardware systems (such as chip designs) and software systems (such as device drivers) has been very successful [1, 2], and has led to the recent application of verification techniques to autonomous agents [5, 6, 29, 39].

1.3 Background: AJPF — Model-Checking Agent Programs

We have been developing techniques for verifying multi-agent systems over a number of years [4, 5, 6]. The agents involved are rational (and, so, are capable of "intelligent" autonomous behaviour) and are represented in a high-level language describing their beliefs, intentions, etc. In particular, we have previously looked at the verification of BDI languages [17], comprising both homogeneous and heterogeneous multi-agent systems [18]. In order to carry out formal verification, formal descriptions of both the system being considered and the properties to be verified are required. Producing formal descriptions for multi-agent systems is well understood, with the BDI approach to rational agency being particularly popular [32, 31]. Our approach, based on *model checking* [11], is outlined in [16] and described in more detail in [4]; see Fig. 1 [4].

[1] *Jason* is a Java-based platform — `http://jason.sf.net`.

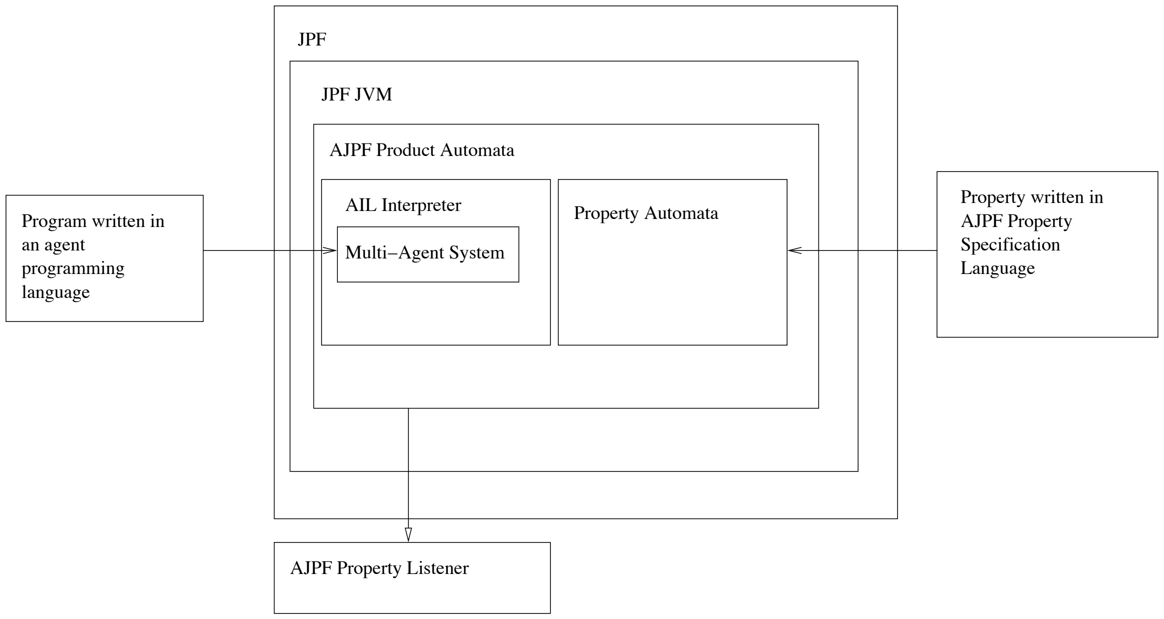

JPF
JPF JVM
AJPF Product Automata
AIL Interpreter
Multi–Agent System
Property Automata
Program written in an agent programming language
Property written in AJPF Property Specification Language
AJPF Property Listener

Figure 1: Outline of Approach.

A multi-agent program, originally programmed in some agent programming language and executed in an interpreter based on the Agent Infrastructure Layer (AIL), is represented above. Each agent uses AIL data structures to store its internal state comprising, for instance, a belief base, a plan library, a current intention, and a set of intentions (as well as other temporary state information). They also use the specific interpreter for the agent programming language built using AIL classes and methods.

The interpreter defines the reasoning cycle for the agent programming language which interacts with the model checker, essentially notifying it when a new state is reached that is relevant for verification. This allows the model checker, AJPF (*Agent JPF*), to create a Java product automata from the program and a property defined in the AJPF property specification language.

The product automata runs in the JPF (Java PathFinder [37, 25]) virtual machine. This is a Java virtual machine specially designed to maintain backtrack points and explore, for instance, all possible thread scheduling options (that can affect the result of the verification) [37]. The JPF model checker is extensible and configurable, which allows us to optimise its performance for AIL-based systems, for instance AJPF has a specialised *Property Listener* which terminates execution of the product automata if it detects that the property has been, and will remain, satisfied for the rest of the run.

1.4 Background: GWENDOLEN

Within this paper, we target one particular BDI language, namely GWENDOLEN [15]. Although this language is simple, it is designed to exhibit many features common to BDI languages, and is specifically tailored for use with AJPF. Agents are represented as sets of initial beliefs and goals together with a library of plans. A multi-agent system is a set of agents together with an environment through which communication occurs and in which actions are performed.

Plans are enabled when an agent has certain beliefs and goals and suggest a sequence of deeds to be performed in order to attain a goal. Plans may also be triggered by events, such as changes in belief following perception or the commitment to goals. We term such plans *triggered plans* whereas plans which depend on an agent's internal state alone are *untriggered plans*. Plans are therefore a triple of a triggering event (if relevant) a guard (that is checked against the agent's beliefs and goals) and a body of deeds to be performed.

GWENDOLEN agents also distinguish two sorts of goals. *Achievement goals* make statements about beliefs the agent wishes to hold. They remain goals until the agent gains an appropriate belief. *Perform goals* simply state a sequence of deeds to be performed and cease to be a goal as soon as that sequence is complete. When an agent takes an action it executes code specific to that action in the environment. Typically this code alters the set of propositions that are perceptable by agents. It may also cause messages to be added to an agent's inbox. Agents go through a specific perception phase when they check their beliefs against the environment's percepts and modify them accordingly. At this point agents also handle the messages currently in their inbox.

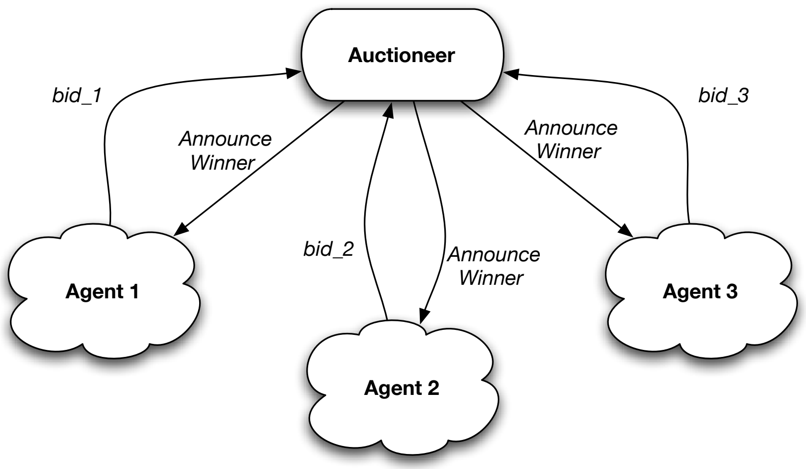
We use the syntax $\uparrow^a m$ to indicate the action of sending a message. Gwendolen agents maintain beliefs about the messages they have sent and received so the syntax $\uparrow^a m$ is also used in plan guards as is $\downarrow^a m$ to indicate that a message has been received.

2 Case Study — Bidding and Coalitions in Simple Auctions

The basic case study we will describe, program and verify is initially very simple. In the sections below we will describe the basic scenario, and then describe more sophisticated variants each becoming increasingly realistic.

2.1 Basic Auction Scenario

The basic idea is simple. A number of agents (in the diagram below, three) make bids of some value to an auctioneer agent. The auctioneer agent then awards the contract to the highest bidder and announces this. This cycle can then repeat, if necessary (note that, in our verified scenarios, the bidding process does *not* cycle).



The GWENDOLEN code for the four agent (one auctioneer agent and three bidding agents) is given below. Here, all agents believe what they are told. Agent `ag1`, who takes the role of the auctioneer, essentially records bids from the other agents and, when bids have been received from the other 3 agents (ag2, ag3, and ag4) notifies the agents of the winner. (Note that we have simplified this to assume that bids are either 100, 150, or 200, with the agent bidding 200 winning — we will describe changes to this later.) The bidding agents (ag2, ag3, and ag4) are essentially the same. Each has a (perform) goal to make a bid ($!_p bid$), and one plan to achieve this. This plan just allows each agent to make a bid of either 100, 150, or 200.

AGENT: ag1

Plans:

$+\downarrow^{Ag} \textbf{tell}(B) : \top$ <- $+B$
$+bid(Z,A) : bid(X_1,ag2) \wedge bid(X_2,ag3) \wedge bid(X_3,ag4) \wedge bid(200,A_1)$ <- $\uparrow^{A_1} \textbf{tell}(win)$

AGENT: ag2

Initial Beliefs: $my\_name(ag2)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag} \textbf{tell}(B) : \top$ <- $+B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1} \textbf{tell}(bid(100,Name))$ <- $\uparrow^{ag1} \textbf{tell}(bid(100,Name))$

AGENT: ag3

Initial Beliefs: $my\_name(ag3)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag} \textbf{tell}(B) : \top$ <- $+B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1} \textbf{tell}(bid(200,Name))$ <- $\uparrow^{ag1} \textbf{tell}(bid(200,Name))$

AGENT: ag4

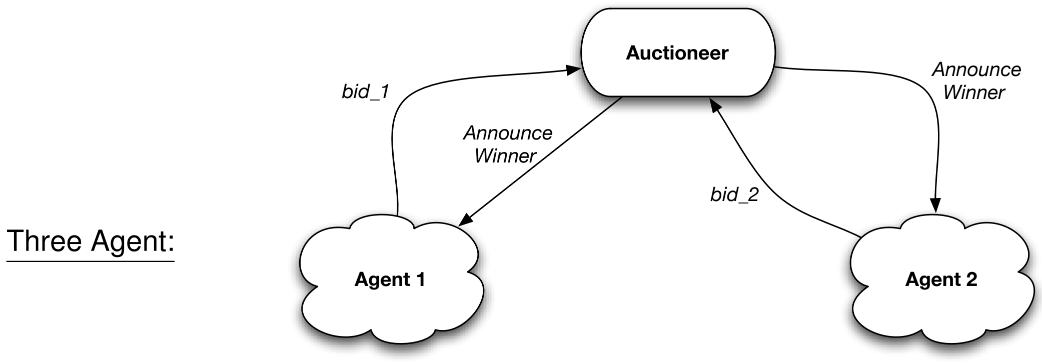
Initial Beliefs: $my\_name(ag4)$

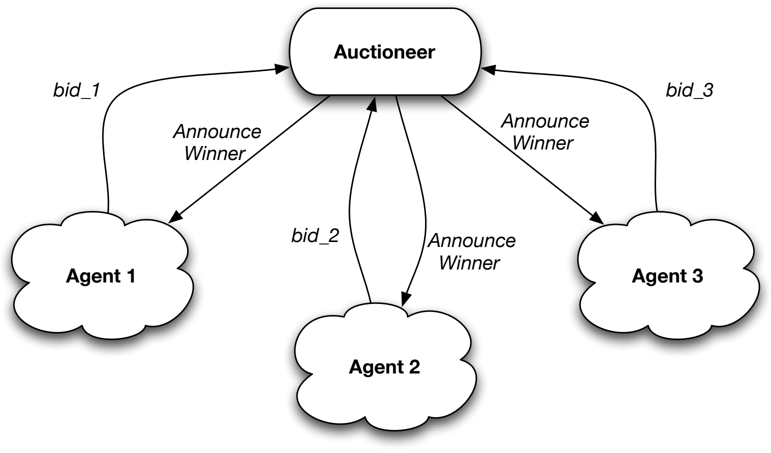
Initial Goals: $+!_p bid$

Plans:

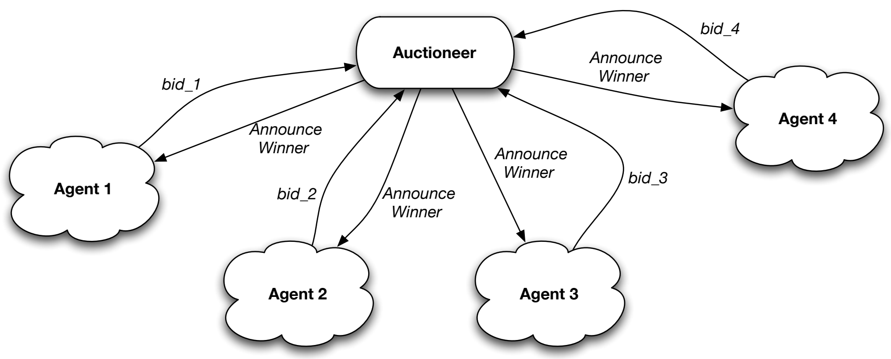
$\downarrow^{Ag} \textbf{tell}(B) : \top$ <- $+B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1} \textbf{tell}(bid(150,Name))$ <- $\uparrow^{ag1} \textbf{tell}(bid(150,Name))$

The verification carried out on this basic scenario and reported in Section 3 concerns scaled versions of this scenario:





Four Agent:

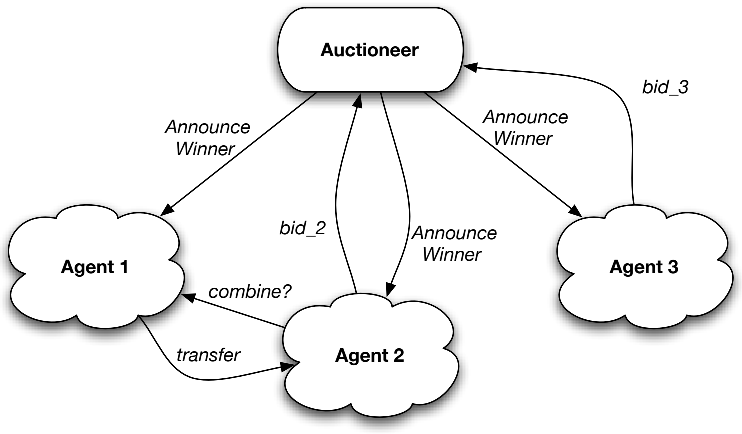


Five Agent:

2.2 Auction Coalition Scenario

The above basic scenario was next extended to include the possibility of *coalitions* [34, 27]. In our model, a coalition is when several agents collaborate by pooling their bid resources in order to win the auction. For example, if three agents x, y, z bid 100, 150 and 200 respectively, then z ought to win every time. However, if x and y form a coalition, their collective bid of 250 will be enough to win the auction.

A simple coalition scenario was implemented in GWENDOLEN with 4 agents: one auctioneer, and three bidders. Two of the bidders bid straight away, but one of the agents attempts to form a coalition by communicating with one of the other bidders. The contacted bidder agrees to form the coalition, and informs the coalition former of its bidding amount. The coalition instigator then combines its own bidding amount with that of its coalition partner, and submits this bid to the auctioneer. Then, having received all of the bids, the auctioneer announces the winner. Below, Agent 2 instigates the coalition:



The GWENDOLEN code for the four agent (one auctioneer agent, `ag1`, and three bidding agents (`ag2`, `ag3`, and `ag4`) coalition scenario is given below.

AGENT: `ag1`

Plans:

$+\downarrow^{Ag}\ \mathbf{tell}(B) : \top\ \texttt{<-}\ +B$
$+bid(Z,A) : bid(X_1,ag2) \wedge bid(X_2,ag3) \wedge bid(X_3,ag4) \wedge bid(250,A_1)\ \texttt{<-}\ \uparrow^{A_1}\ \mathbf{tell}(win)$

AGENT: `ag2`

Initial Beliefs: $my\_name(ag2)$

Initial Goals: $+!_p coalition$

Plans:

$\downarrow^{Ag}\ \mathbf{tell}(B) : \top\ \texttt{<-}\ +B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \mathbf{tell}(bid(250,Name))\ \texttt{<-}\ \uparrow^{ag1}\ \mathbf{tell}(bid(250,Name))$
$+!_p coalition : my\_name(Ag) \wedge \neg\uparrow^{ag4}\ \mathbf{tell}(coalition(Ag))\ \texttt{<-}\ \uparrow^{ag4}\ \mathbf{tell}(coalition(Ag))$
$+agree(A,X) : \top\ \texttt{<-}\ +!_p bid$

AGENT: `ag3`

Initial Beliefs: $my\_name(ag3)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \mathbf{tell}(B) : \top\ \texttt{<-}\ +B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \mathbf{tell}(bid(200,Name))\ \texttt{<-}\ \uparrow^{ag1}\ \mathbf{tell}(bid(200,Name))$

AGENT: `ag4`

Initial Beliefs: $my\_name(ag4)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \mathbf{tell}(B) : \top\ \texttt{<-}\ +B$
$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \mathbf{tell}(bid(150,Name))\ \texttt{<-}\ \uparrow^{ag1}\ \mathbf{tell}(bid(150,Name))$
$+coalition(A) : my\_name(Name) \wedge \neg\uparrow^{A}\ \mathbf{tell}(agree(Name,150))\ \texttt{<-}\ \uparrow^{A}\ \mathbf{tell}(agree(Name,150))$

The main difference in this scenario, as compared with our earlier one, is that one agent, `ag2`, has a goal to form a coalition ($!_p coalition$; we will see later that such goals need not be injected initially). Agent `ag2` then contacts `ag4` and proposes a coalition. If `ag4` agrees then `ag2` can now bid a winning 250 (i.e. $100+150$). Clearly, we would like to verify that this approach does, indeed, lead to `ag2` winning the auction. This is one of the properties we verify in Section 3.

2.3 Auction Coalition Scenario Variant

A further variant on the auction coalition scenario was implemented. In this case, a round of bidding takes place in which all agents bid. Then, after an agent discovers that it has lost the auction, it sends a message to another agent (not the winner) to form a coalition. Then, the agents bid again.

Below, we provide the code for *one* agent in this scenario (the auctioneer). We do not expect the reader to follow this in detail, but present it to show that the agent programs verified can become quite complex. In what follows $+$`lock` is a GWENDOLEN key word that forces the interpreter to continue processing that intention, to the exclusion of others, until it is unlocked (with $-$`lock`). $*B$ is a GWENDOLEN command that "suspends" that intention (i.e. excludes it from further processing) until B is believed, and $win(Ag,Am)$ is an action which posts information about the current winner to the environment, and $c\_winner(Ag_1,Am_w)$ records the current winner within the plan rules.

AGENT: `ag1`

Initial Beliefs: $my\_name(ag1)$

Rules: $allbids : -bid\_processed(ag2) \wedge bid\_processed(ag3) \wedge bid\_processed(ag4)$

Plans:

```
    $\downarrow^{Ag} bid(Ag,Am_1) : bid(Ag,Am_2)$  <-  $-bid(Ag,Am_2); +bid(Ag,Am_1)$
    $\downarrow^{Ag} bid(Ag,Am_1) : \neg bid(Ag,Am_2)$  <-  $+bid(Ag,Am_1)$
    $+bid(Ag,Am) : c\_winner(Ag_w,Am_w) \wedge Am_w < Am \wedge allbids$  <-  +lock;
                                                            $-c\_winner(Ag_1,Am_w)$;
                                                            $+ann\_winner$,
                                                            $+c\_winner(Ag,Am)$;
                                                            $win(Ag,Am)$;
                                                            -lock
    $+bid\_processed(Ag) : c\_winner(Ag_w,Am_w) \wedge allbids \wedge ann\_winner$  <-  +lock;
                                                                 $+ann\_winner$;
                                                                 $win(Ag_1,Am_2)$;
                                                                 -lock
    $+bid(Ag,Am) : \neg c\_winner(Ag_w,Am_w)$  <-  $+c\_winner(Ag,Am); +bid\_processed(Ag)$
    $+bid(Ag,Am) : c\_winner(Ag_w,Am_w) \wedge Am_w < Am \wedge \neg allbids$  <-  +lock;
                                                              $+c\_winner(Ag,Am)$;
                                                              $+bid\_processed(Ag)$;
                                                              $-c\_winner(Ag_1,Am_w)$;
                                                              -lock
    $+bid(Ag,Am) : c\_winner(Ag_w,Am_w) \wedge Am < Am_w \wedge \neg allbids$  <-  $+bid\_processed(Ag)$
```

2.4 Coalition Trust Scenario

This auction scenario is similar to that described in Section 2.2, except the coalition forming agent now has a belief about which other agent(s) it can trust, i.e., the other agents with which it would *prefer* to form a coalition. This trust aspect is static, that is, the coalition-forming agent starts the auction with belief(s) about which agents it can trust, and these do not change during the auction.

The GWENDOLEN code for the five agent (one auctioneer agent, `ag1`, and four bidding agents, `ag2`, `ag3`, `ag4`, and `ag5`) coalition trust scenario is given below. As we can see, agent `ag2` 'trusts' `ag4` and so, even though both `ag4` and `ag5` offer coalitions, `ag2` will only work with `ag4`.

AGENT: `ag1`

Plans:

$+\downarrow^{Ag}\ \textbf{tell}(B) : \top\ \texttt{<-}\ \ +B$

$$+bid(Z,A) : \left[\begin{array}{l} bid(X_1,ag2) \wedge bid(X_2,ag3) \wedge bid(X_3,ag4) \\ \wedge(bid(Z,A) \geq bid(X_1,ag2)) \wedge (bid(Z,A) \geq bid(X_1,ag3)) \\ \wedge(bid(Z,A) \geq bid(X_1,ag4)) \end{array}\right] \ \texttt{<-}\ \left[\begin{array}{l} \uparrow^{ag2}\ \textbf{tell}(win(A)); \\ \uparrow^{ag3}\ \textbf{tell}(win(A)); \\ \uparrow^{ag4}\ \textbf{tell}(win(A)) \end{array}\right]$$

AGENT: `ag2`

Initial Beliefs: $my\_name(ag2)$, $trust(ag4)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \textbf{tell}(B) : \top\ \texttt{<-}\ \ +B$

$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \textbf{tell}(bid(150,Name))\ \ \texttt{<-}\ \ \uparrow^{ag1}\ \textbf{tell}(bid(150,Name))$

$$+win(A) : \left[\begin{array}{l} my\_name(Name) \wedge \neg win(Name) \wedge trust(Ag) \\ \wedge\neg\uparrow^{Ag}\ \textbf{tell}(coalition(Name)) \end{array}\right] \ \texttt{<-}\ \uparrow^{Ag}\ \textbf{tell}(coalition(Name))$$

$+agree(A,X) : \top\ \ \texttt{<-}\ \ \uparrow^{ag1}\ \textbf{tell}(bid(300,ag2))$

AGENT: `ag3`

Initial Beliefs: $my\_name(ag3)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \textbf{tell}(B) : \top\ \texttt{<-}\ \ +B$

$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \textbf{tell}(bid(200,Name))\ \ \texttt{<-}\ \ \uparrow^{ag1}\ \textbf{tell}(bid(200,Name))$

AGENT: `ag4`

Initial Beliefs: $my\_name(ag4)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \textbf{tell}(B) : \top\ \texttt{<-}\ \ +B$

$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \textbf{tell}(bid(150,Name))\ \ \texttt{<-}\ \ \uparrow^{ag1}\ \textbf{tell}(bid(150,Name))$

$+coalition(A) : my\_name(Name) \wedge \neg\uparrow^{A}\ \textbf{tell}(agree(Name,150))\ \ \texttt{<-}\ \ \uparrow^{A}\ \textbf{tell}(agree(Name,150))$

AGENT: `ag5`

Initial Beliefs: $my\_name(ag5)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\ \textbf{tell}(B) : \top\ \texttt{<-}\ \ +B$

$+!_p bid : my\_name(Name) \wedge \neg\uparrow^{ag1}\ \textbf{tell}(bid(150,Name))\ \ \texttt{<-}\ \ \uparrow^{ag1}\ \textbf{tell}(bid(150,Name))$

$+coalition(A) : my\_name(Name) \wedge \neg\uparrow^{A}\ \textbf{tell}(agree(Name,150))\ \ \texttt{<-}\ \ \uparrow^{A}\ \textbf{tell}(agree(Name,150))$

2.5 Dynamic Trust Scenario

This final auction scenario builds upon the previous one. Here, if the coalition-forming agent loses the auction, it tries to form a coalition with an agent it trusts. Then, if its coalition is successful in winning the auction, it stops. However, if its coalition is *unsuccessful* then it no longer believes that it can trust the other agent in the coalition, and will try to form another coalition with another agent it trusts (except the winner).

AGENT: `ag1`

Plans:

$+\downarrow^{Ag}\textbf{tell}(B) : \top \ \texttt{<-}\ +B$

$$+bid(Z,A) : \begin{bmatrix} bid(X_1,ag2) \land bid(X_2,ag3) \land bid(X_3,ag4) \\ \land(bid(Z,A) \geq bid(X_1,ag2)) \land (bid(Z,A) \geq bid(X_1,ag3)) \\ \land(bid(Z,A) \geq bid(X_1,ag4)) \end{bmatrix} \ \texttt{<-}\ \begin{bmatrix} \uparrow^{ag2}\textbf{tell}(win(A)); \\ \uparrow^{ag3}\textbf{tell}(win(A)); \\ \uparrow^{ag4}\textbf{tell}(win(A)) \end{bmatrix}$$

AGENT: `ag2`

Initial Beliefs: $my\_name(ag2), trust(ag4), trust(ag5)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\textbf{tell}(B) : \top \ \texttt{<-}\ +B$

$+!_p bid : my\_name(Name) \land \neg\uparrow^{ag1}\textbf{tell}(bid(150,Name)) \ \texttt{<-}\ \uparrow^{ag1}\textbf{tell}(bid(150,Name))$

$$+win(A) : \begin{bmatrix} my\_name(Name) \land \neg win(Name) \land trust(Ag) \\ \land \neg formed\_coalition(Ag') \land \neg\uparrow^{Ag}\textbf{tell}(coalition(Name)) \end{bmatrix} \ \texttt{<-}\ \begin{bmatrix} \uparrow^{Ag}\textbf{tell}(coalition(Ag)); \\ +formed\_coalition(Ag) \end{bmatrix}$$

$$+win(A) : \begin{bmatrix} my\_name(Name) \land \neg win(Name) \land trust(Ag) \\ \land formed\_coalition(Ag') \land \neg\uparrow^{Ag}\textbf{tell}(coalition(Name)) \end{bmatrix} \ \texttt{<-}\ \begin{bmatrix} \uparrow^{Ag}\textbf{tell}(coalition(Ag)); \\ +formed\_coalition(Ag); \\ -trust(Ag') \end{bmatrix}$$

$+agree(A,X) : \top \ \texttt{<-}\ \uparrow^{ag1}\textbf{tell}(bid(300,ag2))$

AGENT: `ag3`

Initial Beliefs: $my\_name(ag3)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\textbf{tell}(B) : \top \ \texttt{<-}\ +B$

$+!_p bid : my\_name(Name) \land \neg\uparrow^{ag1}\textbf{tell}(bid(200,Name)) \ \texttt{<-}\ \uparrow^{ag1}\textbf{tell}(bid(200,Name))$

AGENT: `ag4`

Initial Beliefs: $my\_name(ag4)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\textbf{tell}(B) : \top \ \texttt{<-}\ +B$

$+!_p bid : my\_name(Name) \land \neg\uparrow^{ag1}\textbf{tell}(bid(150,Name)) \ \texttt{<-}\ \uparrow^{ag1}\textbf{tell}(bid(150,Name))$

$+coalition(A) : my\_name(Name) \land \neg\uparrow^{A}\textbf{tell}(agree(Name,150)) \ \texttt{<-}\ \uparrow^{A}\textbf{tell}(agree(Name,150))$

AGENT: `ag5`

Initial Beliefs: $my\_name(ag5)$

Initial Goals: $+!_p bid$

Plans:

$\downarrow^{Ag}\textbf{tell}(B) : \top \ \texttt{<-}\ +B$

$+!_p bid : my\_name(Name) \land \neg\uparrow^{ag1}\textbf{tell}(bid(150,Name)) \ \texttt{<-}\ \uparrow^{ag1}\textbf{tell}(bid(150,Name))$

$+coalition(A) : my\_name(Name) \land \neg\uparrow^{A}\textbf{tell}(agree(Name,150)) \ \texttt{<-}\ \uparrow^{A}\textbf{tell}(agree(Name,150))$

3 Verification

We now discuss the results of verifying our various auction scenarios. In particular we will discuss the property that was verified for each scenario and provide some time, state space and memory statistics for the verification attempt. Our use of AJPF, optimised as it is to the agent scenario and the AIL, considerably reduces the state space compared to the state space that would be searched by JPF alone. However, it should be noted that although the state space is smaller, the transitions between states often require considerable computation in the JPF JVM. This virtual machine is not optimised for speed. This accounts for the relatively large time taken per state that is shown by our figures compared to many model checkers (see Section 4 for comment on efficiency).

3.1 Properties of the Basic Scenario (from Section 2.1)

The following property was verified:

$$\lozenge \mathsf{B}(ag_i, win)$$

where ag_i is the agent with the highest bid, 'B' is the *belief* operator, and '$\lozenge$' means "at some time in the future". In other words, the agent with the highest bid will eventually believe it has won. This scenario was tackled with 3, 4 and 5 agents, with the following statistics (here, timings are given in the form *hours*:*minutes*:*seconds*).

	1 auctioneer 2 bidders	1 auctioneer 3 bidders	1 auctioneer 4 bidders
Elapsed Time	0:00:55	0:06:58	1:06:01
Size of State Space	44	265	1706
Max Memory	17MB	19MB	20MB

3.2 Properties of the Auction Coalition Scenario (from Section 2.2)

In this scenario, we verified the property

$$\lozenge \mathsf{B}(a_j, win)$$

where a_j is the coalition-forming agent. Recall that here, two agents who could not win on their own form a coalition; the verification shows that they are, indeed, successful.

	1 auctioneer 3 bidders; 1 coalition	1 auctioneer 4 bidders; 1 coalition
Elapsed Time	0:06:35	1:54:04
Size of State Space	575	4571
Max Memory	21MB	22MB

3.3 Properties of the Auction Coalition Scenario Variant (from Section 2.3)

Recall that, in this scenario, the coalition formation was not "hard-wired" from the start. Agents bid and, when they find they have lost, might then attempt to form a coalition for the next auction. We have tackled a number of variants of this, both in terms of the number of bidding agents (3 or 4), the number of coalitions that can be formed amongst the bidders (1 or 2), and the property verified. In particular we verified both properties involving time and belief (as previously) and time and *goals*.

Belief Property.

The property verified here was

$$\Diamond \mathsf{B}(a_k, win)$$

where a_k is the agent that forms the winning coalition.

	1 auctioneer 3 bidders 1 coalition	1 auctioneer 3 bidders 2 coalitions	1 auctioneer 4 bidders 1 coalition	1 auctioneer 4 bidders 2 coalitions
Elapsed Time	0:06:48	0:13:26	0:53:15	2:17:17
Size of State Space	429	903	2496	6728
Max Memory	22MB	24MB	23MB	25MB

Goal Property.

The property verified was

$$\Diamond \mathsf{G}(a_k, coalition)$$

where a_k was the (last) coalition forming agent. Here, 'G' is the 'goal' operator. In verifying this, we are not checking that the agent necessarily achieved any win, but that it (at some point in its execution) adopted the goal to form a coalition.

	1 auctioneer 3 bidders 1 coalition	1 auctioneer 3 bidders 2 coalitions	1 auctioneer 4 bidders 1 coalition	1 auctioneer 4 bidders 2 coalitions
Elapsed Time	0:04:43	0:07:40	0:40:34	0:58:08
Size of State Space	234	469	1452	2780
Max Memory	20MB	22MB	21MB	23MB

3.4 Coalition Trust Scenario (from Section 2.4)

We now turn to the scenarios involving *trust*. In this scenario, the agent would only form a coalition with a trusted agent, and its idea of trust was prescribed initially. Four agents were used in the first example: an auctioneer, and three bidding agents a_1, a_2, a_3 with bids 100, 200 and 150. The property verified was

$$\Diamond \mathsf{B}(a_1, win)$$

where a_1 forms a coalition with a_3 after losing the auction to a_2. The results were:

Elapsed Time	0:03:18
Size of State Space	297
Max Memory	20MB

In the second example, five agents were used: an auctioneer, and four bidding agents a_1, a_2, a_3, a_4 with bids 100, 200, 150 and 150. The property verified was

$$\Diamond \mathsf{B}(a_1, win)$$

where a_1 forms a coalition with one of the trusted agents, a_4 and a_5, after losing the auction to a_2. The results were:

Elapsed Time	0:25:37
Size of State Space	2065
Max Memory	22MB

3.5 Dynamic Trust Scenario (from Section 2.5)

Finally, we have the dynamic trust scenario, where an agent loses trust in another if they fail in a coalition together. Five agents were used: an auctioneer, and four bidding agents a_1, a_2, a_3, a_4 with bids 100, 200, 25 and 150. The property verified was

$$\lozenge \mathsf{B}(a_1, win)$$

where a_1 forms a coalition with one of the trusted agents after losing the auction to a_2. If it chooses a_4 first, it wins and stops. If it chooses a_3 first, it loses the auction and distrusts that agent, trying subsequently with a_4. It then wins the auction. The results were:

Elapsed Time	0:20:30
Size of State Space	2065
Max Memory	22MB

4 Concluding Remarks

In this paper we have discussed the verification by model-checking of agent-based auction software. We have focussed on a series of scenarios of increasing complexity in order to demonstrate that, although the complexity of the model checking task increases with the complexity of the scenario it is nevertheless a realistic proposition to model-check the properties of interesting multi-agent implementations within a reasonable time. For example, as we reach the situation where agents form coalitions together, based on a dynamic notion of trust, in order to compete with other agents, then we are not far from realistic agent systems.

Clearly, for bigger scenarios improved efficiency will be required (and, indeed, this is something we are actively working on), but the examples implemented and verified in this paper show that non-trivial market-based multi-agent systems can be automatically verified.

References

[1] T. Ball and S. K. Rajamani. The SLAM Toolkit. In *Proc. 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 260–264. Springer, 2001.

[2] S. Berezin, E. M. Clarke, A. Biere, and Y. Zhu. Verification of Out-Of-Order Processor Designs Using Model Checking and a Light-Weight Completion Function. *Formal Methods in System Design*, 20(2):159–186, 2002.

[3] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer-Verlag, 2005.

[4] R. H. Bordini, L. A. Dennis, B. Farwer, and M. Fisher. Automated Verification of Multi-Agent Programs. In *Proc. 23rd Int. Conf. Automated Software Engineering (ASE)*, 2008.

[5] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model Checking Rational Agents. *IEEE Intelligent Systems*, 19(5):46–52, September/October 2004.

[6] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying Multi-Agent Programs by Model Checking. *J. Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[7] C. Boutilier, Y. Shoham, and M. P. Wellman. Economic Principles of Multi-Agent Systems. *Artif. Intell.*, 94(1-2):1–6, 1997.

[8] J. Bredin, D. Kotz, D. Rus, R. T. Maheswaran, Ç. Imer, and T. Basar. Computational Markets to Regulate Mobile-Agent Systems. *J. Autonomous Agents and Multi-Agent Systems*, 6(3):235–263, 2003.

[9] W. Clancey, M. Sierhuis, C. Kaskiris, and R. van Hoof. Advantages of Brahms for Specifying and Implementing a Multiagent Human-Robotic Exploration System. In *Proc. 16th Int. Conf. Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 7–11. AAAI Press, 2003.

[10] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Dec. 1999.

[11] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Prog. Lang. Syst.*, 8(2):244–263, 1986.

[12] J. Collins, P. Faratin, S. Parsons, J. A. Rodríguez-Aguilar, N. M. Sadeh, O. Shehory, and E. Sklar, editors. *Agent-Mediated Electronic Commerce and Trading Agent Design and Analysis (AMEC/TADA) — Selected and Revised Papers from AAMAS workshops*, volume 13 of *LNBIP*. Springer, 2009.

[13] J. M. Corera, I. Laresgoiti, and N. R. Jennings. Using Archon, Part 2: Electricity Transportation Management. *IEEE Expert*, 11(6):71–79, 1996.

[14] R. K. Dash, P. Vytelingum, A. Rogers, E. David, and N. R. Jennings. Market-Based Task Allocation Mechanisms for Limited-Capacity Suppliers. *IEEE Trans. Systems, Man, and Cybernetics (A)*, 37(3):391–405, 2007.

[15] L. A. Dennis and B. Farwer. Gwendolen: A BDI Language for Verifiable Agents. In *Logic and the Simulation of Interaction and Reasoning*, AISB, 2008.

[16] L. A. Dennis, B. Farwer, R. H. Bordini, and M. Fisher. A Flexible Framework for Verifying Agent Programs. In *Proc. 7th Int. Conf. Autonomous Agents and Multiagent Systems (AAMAS)*. ACM Press, 2008. (Short paper).

[17] L. A. Dennis, B. Farwer, R. H. Bordini, M. Fisher, and M. Wooldridge. A Common Semantic Basis for BDI Languages. In *Proc. 5th Int. Workshop on Programming Multiagent Systems (ProMAS)*, volume 4908 of *LNAI*, pages 124–139. Springer, 2008.

[18] L. A. Dennis and M. Fisher. Programming Verifiable Heterogeneous Agent Systems. In *Proc. 6th Int. Workshop on Programming in Multi-Agent Systems (ProMAS)*, 2008.

[19] I. Doghri. Formal verification of WAHS: an autonomous and wireless P2P auction handling system. In *Proc. 8th Int. Conf. New Technologies in Distributed Systems (NOTERE)*, pages 1–10. ACM Press, 2008.

[20] M. Fisher, R. Bordini, B. Hirsch, and P. Torroni. Computational Logics and Agents: A Roadmap of Current Technologies and Future Trends. *Computational Intelligence*, 23(1):61–91, 2007.

[21] L. Fortnow, J. Riedl, and T. Sandholm, editors. *Proc. 9th ACM Conference on Electronic Commerce (EC)*. ACM, 2008.

[22] M. A. Gibney, N. R. Jennings, N. J. Vriend, and J.-M. Griffiths. Market-Based Call Routing in Telecommunications Networks Using Adaptive Pricing and Real Bidding. In *Proc. 3rd Int. Workshop on Intelligent Agents for Telecommunication Applications (IATA)*, volume 1699 of *LNCS*, pages 46–61. Springer, 1999.

[23] N. Haque, N. R. Jennings, and L. Moreau. Resource Allocation in Communication Networks using Market-Based Agents. *Knowl.-Based Syst.*, 18(4-5):163–170, 2005.

[24] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004. (2nd Edition).

[25] Java PathFinder. `http://javapathfinder.sourceforge.net`.

[26] P. Klemperer. *Auctions: Theory and Practice*. Princeton University Press, Princeton, USA, 2004. See also `http://www.nuff.ox.ac.uk/users/klemperer/VirtualBook/VBCrevisedv2.asp`.

[27] H. Konishi and D. Ray. Coalition Formation as a Dynamic Process. *Journal of Economic Theory*, 110(1):1 – 41, 2003.

[28] N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103(1-2):5–48, 1998.

[29] F. Raimondi and A. Lomuscio. Automatic Verification of Multi-agent Systems by Model Checking via Ordered Binary Decision Diagrams. *J. Applied Logic*, 5(2):235–251, 2007.

[30] A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.

[31] A. S. Rao and M. Georgeff. BDI Agents: From Theory to Practice. In *Proc. 1st Int. Conf. Multi-Agent Systems (ICMAS)*, 1995.

[32] A. S. Rao and M. P. Georgeff. Modeling Agents within a BDI-Architecture. In *Proc. Int. Conf. Principles of Knowledge Representation and Reasoning (KR)*. Morgan Kaufmann, 1991.

[33] D. M. Reeves, M. P. Wellman, J. K. MacKie-Mason, and A. Osepayshvili. Exploring Bidding Strategies for Market-Based Scheduling. *Decision Support Systems*, 39(1):67–85, 2005.

[34] T. Sandholm and V. R. Lesser. Coalitions Among Computationally Bounded Agents. *Artificial Intelligence*, 94(1-2):99–137, 1997.

[35] M. Sierhuis. Multiagent Modeling and Simulation in Human-Robot Mission Operations. 2006 (`http://ic.arc.nasa.gov/ic/publications`).

[36] W. Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *J. Finance*, 16(1):8–37, 1961.

[37] W. Visser, K. Havelund, G. P. Brat, S. Park, and F. Lerda. Model Checking Programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[38] W. E. Walsh and M. P. Wellman. A Market Protocol for Decentralized Task Allocation. In *Proc. 3rd Int. Conf. Multiagent Systems (ICMAS)*, pages 325–332. IEEE Computer Society, 1998.

[39] C. D. Walton. Verifiable Agent Dialogues. *J. Applied Logic*, 5(2):197–213, 2007.

[40] M. Wooldridge, T. Ågotnes, P. E. Dunne, and W. van der Hoek. Logic for Automated Mechanism Design - A Progress Report. In *Proc. 22nd National Conference on Artificial Intelligence (AAAI)*, pages 9–. AAAI Press, 2007.

[41] M. Wooldridge and A. Rao, editors. *Foundations of Rational Agency*. Kluwer Academic Publishers, 1999.