Realising Common Knowledge Assumptions in Agent Auctions

Frank Guerin and Emmanuel M. Tadjouddine
Department of Computing Science, King's College,
University of Aberdeen, Aberdeen AB24 3UE, Scotland.
email {etadjoud,fguerin}@csd.abdn.ac.uk

Abstract

Game theory is popular in agent systems for designing auctions with desirable properties. However, many of these properties will only hold if the game and its properties are common knowledge among the agents. For example, in an auction where truthful bidding is an equilibrium strategy, unless this is common knowledge, it may not be rational for an agent to bid truthfully. It is currently not clear how this state of common knowledge can be achieved, especially in open agent societies where agents may encounter previously unseen auction specifications. We need a method for communicating the rules of the game to the agents, and the agents need to be able to determine its properties. We present a machine-readable language in which the rules of the game can be written. We show that it is not feasible for an agent to determine the properties of any arbitrary specification, unless information about the properties is communicated and/or certain restrictions are placed on the specification. We look at two special cases where common knowledge is achievable: auctions with identical players where the two highest bidders determine the price, and Groves mechanisms with a restriction on the pricing rule.

1. Introduction

The vision of agent mediated eCommerce is of an open society of software agents engaging in financial transactions and entering legally binding contracts in much the same way as humans do in the real world. Many aspects of the agent vision mirror current human-based systems. For example, agents may be acting on behalf of different individuals or organisations who may have conflicting interests; agents may choose to gain advantage by deception, or they may make choices which are suboptimal for the welfare of the society but which maximise their own personal gain. Some solutions to these problems have already been developed for human scenarios; game theory can be used to design rules for an interaction so as to give the participants an incentive to achieve the outcome desired by the designer. For example, game theory can design mechanisms which encourage truth telling, or which encourage the participants to achieve the optimal social welfare. Agent researchers are keen to exploit these techniques. However, there remain several unsolved problems, which will need to be tackled in order to make game theory practically usable in open agent systems. A number of these problems are outlined by Dash et al. [2] in their "Call to Arms". This paper is to some extent a response to that call.

The problem we will tackle is the problem of the "common knowledge assumptions" which most game theoretic solutions rely on. This is easily explained by an example: Game theory could be used to design an auction where truthful bidding is an equilibrium. This means that if all agents bid truthfully, then no single agent will have an incentive to deviate from truthful bidding. However, the agents' incentives to follow the equilibrium strategy (truthful bidding) rest on the assumption that the rules of the game and the equilibrium are common knowledge amongst the agents. Moreover, it is not enough to simply tell them the rules; the agents must have a rational basis for believing that the auction will indeed be implemented according to those rules. If agents do not have guarantees of compliance then it would not be rational for them to expect the equilibrium to be played. Thus the common knowledge includes not only a knowledge of the rules and the equilibrium, but also a knowledge that all agents will play by the rules. Clearly, achieving the state of common knowledge required to effectively implement these game theoretic solutions is a non-trivial problem in open agent systems.

The problem can be broken down into two parts: (i) Semantic interoperation: There is a need for machine-understandable specification of the rules and properties of games. (ii) Compliance: There is a need for ways to guarantee that the participants will abide by the game's rules.

These are in fact two of the outstanding problems outlined by Dash et al. [2]. Both of these problems are consequences of the open nature of proposed agent mediated eCommerce systems. For the first problem, by "open" we mean that foreign agents will be free to enter and leave dif-

ferent agent systems at will, and so will need to be able to work with previously unseen protocols. There is by now a significant body of work concerned with the design of auctions for use in multi-agent trading scenarios [6, 14, 9, 16]. However, these auctions are typically described in research papers using a diverse combination of natural language, logics and pseudocode descriptions of algorithms. Only a human researcher can understand these specifications of auctions. The vision of agent mediated e-commerce will require agents themselves to understand specifications of auctions. This is because agents will need to be able to move between different electronic auction houses where different auctions are in use; when an agent joins an auction house it will need to consult some specification to learn about and understand the rules of engagement for that auction house. It needs to do this in order to decide whether to participate, and if so, with what strategy. Enabling agents to understand the specifications of (previously unseen) auctions is a problem that has not yet been solved.

The second problem also stems from the open nature of the system. Constituent agents may be owned by different individuals or organisations. This means that they may have conflicting interests and so it might be in their interest to cheat each other by not following the rules of the system. The classic case of this is when the auctioneer in a sealed bid auction inserts fictitious bids, and so extracts more revenue from the winner. If any participant in a game has an opportunity to break the rules, then the properties of the game are likely to be destroyed. There is clearly a need to provide guarantees that the game will be implemented as advertised, so that its properties will hold.

In Section 2 we give an overview of the components of our proposed solution. In Section 3 we describe our specification language, which allows the rules of the game to be made public. In Section 4 we look at how agents could achieve common knowledge of a Bayesian Nash equilibrium. In Section 5 we look at how to achieve common knowledge of strategyproofness and ex post individual rationality via Groves mechanisms. Section 6 concludes.

2. Overview of Proposed Approach

Our proposed solution to the problem is in two parts: Machine Understandable Auctions (MUA) and the Universal Auction Machine (UAM). These correspond to the two aspects of the problem as outlined in the introduction: Semantic Interoperation and Compliance.

2.1. Machine-Understandable Auctions (MUA)

The idea of the Machine-Understandable Auction comes from the need for agents to have the ability to understand the rules and properties of auctions which they may not have seen before. Consider the following scenarios:

1. An agent may move to a new auction house and download a published specification for an auction.

2. Individual agents may have their own libraries of mechanisms, and may swap proposed mechanisms, until agreement is reached on which one to use.

3. Agents may engage in a phase of negotiation over the rules of the game, until agreeing on its final form. This currently happens in many human scenarios.

4. A tailor-made mechanism may be designed by one of the agents, to fit precisely the parameters of the scenario. This type of *Automated Mechanism Design* has been shown to be feasible in certain cases [12, 13].

All of these scenarios have a common feature: the particular mechanism proposed is likely to be novel to most of the agents who are being asked to participate in it. These agents will need to be able to understand the rules and properties of the auction to make a decision about whether or not to participate, and if participating they must determine their optimal strategy.

By *understanding* we mean: firstly, an understanding of what the participants are allowed to do at any stage in the auction; secondly, an understanding of the game-theoretic properties of the auction.

The first requirement can be met by having a standard machine-readable specification language for auction protocols. This language must allow an auction to be specified by stating the rules which participating agents must follow; this includes specifying how the winner is determined, and the price to be paid. These rules must be sufficiently rigorous to ensure that a system of agents complying with the rules will enjoy the properties which the auction was designed to have. Such a language could simply be a general purpose programming language; this would be sufficiently expressive to capture any auctions which could be implemented. However, the more general the language is (i.e. free from any constructs specific to particular auction classes), the more difficult it will be to deduce the properties of the mechanisms encoded in it. Hence the choice of language is related to the second requirement.

The second requirement can be met by giving the agents some algorithms which they can use to deduce the properties of mechanisms. The types of game theoretic properties which the agents must understand include pareto efficiency, optimality of revenue, individual rationality, strategyproofness, etc. [10]. Two extreme solutions can be envisaged: On the one hand we could have a completely general auction language and give agents procedures which they can use to analyse an arbitrary mechanism and deduce which

properties hold for it; on the other hand we could forego the auction specification language and instead provide a trusted library of specific mechanisms, annotated with a description of their properties. The first solution is not feasible computationally; the second solution precludes the possibility of using any mechanism which is not in the trusted library, hence it is too restrictive for the types of agent scenarios described above. We advocate a compromise solution: The language will be designed for certain classes of mechanisms and their properties. Once specific class of auction and desired property is chosen, the language allows the details of the particular mechanism to be specified. When parts of an auction are customisable, the desired property should still be verifiable. Furthermore, certain components of a mechanism can be specified in a trusted library, while other details can be left open and specified through the language. For example, we may have a (proven) optimal winner determination algorithm in the library, and a new mechanism may choose to use this, while specifying its own customised pricing rules.

2.2. Universal Auction Machine (UAM)

To verify that agents are going to comply with the specification of an auction it is unlikely that we will be able to directly inspect their code and test it. Agents in the system may be developed by different vendors and these vendors might desire to keep the internal code of their agents secret. Even if the code is published, the agents might be based on very different architectures making it too difficult to verify their behaviour by analysis of the code.

The idea of the Universal Auction Machine (UAM) is of a trusted third party providing a virtual auctioneer agent that can be given a machine-readable specification for an auction, and will run the auction according to that specification; i.e. it will ask the agents for their bids, running repeated rounds if required, and it will finally determine the winner and the price to be paid. The UAM's code should be open source so that it is open to inspection by all parties; they can verify that it correctly implements any auction specification it is given. The auction specifications which the UAM accepts are the same as the specification of the rules used in the MUA. In this way the UAM can execute specifications of auctions which may not be known at the time the UAM is written. This fits our scenarios above where agents might agree to use a new tailor made auction for their situation: once they agree on the rules of the mechanism, an instance of the UAM can be spawned to run it. The use of a trusted third party to run the auction is in fact the only way to enable the rules of the auction to be enforced without revealing any of the participants' private information to each other, for example in a sealed bid auction.

3. Auction Specification Language

This section presents the syntax and semantics for SMPL (Simple Mechanism Programming Language). This is quite a general mechanism programming language which can capture static or dynamic games of perfect or imperfect information. It is slightly simplified from the version introduced in [5]. SMPL allows us to explicitly describe the information exchanged during the course of the game. This is achieved by having a module for each player, and using special variables which the players read from or write to, in order to communicate. An SMPL program has the following modular syntax: $[M_0 :: [S_0] \parallel \ldots \parallel M_n :: [S_n]]$
It consists of $n+1$ parallel modules, representing the principal (module M_0) and each of the players. Each M_i is an identifier for a player in the game and each S_i is a statement which may itself be composed of other statements. An example of a program can be seen in Section 4.5. The allowed statements are as follows:

Basic Statement	***Description***
idle	no operation
$u := e$	*assign* value e to variable u
choose $c_1..c_2$	*choose* a value for out variable
if c **then** S_1 **else** S_2	*conditional* statement
if c **then** S	*abbreviates* **if** c **then** S **else idle**
while c **do** S	*repetition* of S
wait c	*abbreviates* **while** c **do idle**
$S_1; \ldots; S_k$	*sequential* execution

Sub statements within sequential execution statements are separated by semicolons which we omit if there is a line break. The SMPL program has two special variables in and out for communication. These are tuples, having one component for each player module M_i which has access to in_i and out_i (a subscript is used to specify an element of a tuple); no other player module can refer to these components, apart from the principal. The principal can "send" a message to player i by writing a nonzero value to in_i; player i then reads it and resets in_i to zero. Player i can "send" a message to the principal by writing a nonzero value to out_i; the principal then reads it and resets it to zero. We allow no communication between other individual agents; everything must go through the principal. This does not restrict the class of games that can be represented, for example we could represent an open cry English auction by having a bidder submit a bid to the principal, and having the principal communicate the bid to all other players. Note that the Players make choices in the game by means of the **choose** statements. It is required that the module of exactly one player will begin with a **choose** statement. No **choose** statements may appear in module M_0. Module M_0 has a private variable P to which it writes the final outcome of the game; this is a tuple where P_i is the outcome for agent i.

3.1. SMPL Program Semantics

The semantics are defined via a transition system $\langle V, \mathcal{T} \rangle$. V is the set of system variables, which are either integers, reals or tuples of these; one of these is the control variable π which represents the location of the next statement to be executed, the remainder represent program variables. π is an $(n+1)$-tuple, where n is the number of players in the program ($+1$ for the principal); π has one part of its tuple to point to the current location within each player's module; π initially points to the start of each module, and all the remaining variables are initially assigned value zero, this is the state from which the system can start running. A transition is a relation which relates a state of the system to its possible successor states. $\mathcal{T}$ is a set of transitions including one transition corresponding to each statement in the program, as follows. Primed variables refer to the value in the successor state, while unprimed variables refer to the current state. ℓ is a statement's label and $\hat{\ell}$ its post-label. The abbreviation *pres* means that all variables not referred to in the transition relation preserve their previous values.

Statement	***Transition Relation***
idle	$\pi_i = \ell \wedge \pi_i' = \hat{\ell} \wedge pres$
$u := e$	$\pi_i = \ell \wedge \pi_i' = \hat{\ell} \wedge u' = e \wedge pres$
choose $c_1..c_2$	$\pi_i = \ell \wedge \pi_i' = \hat{\ell} \wedge pres \wedge$ $[out_i' = c_1 \vee \ldots \vee out_i' = c_2]$
if c **then** ℓ_1: S_1 **else** ℓ_2: S_2	$[\pi_i = \ell \wedge \pi_i' = \ell_1 \wedge c \wedge pres] \vee$ $[\pi_i = \ell \wedge \pi_i' = \ell_2 \wedge \neg c \wedge pres]$
while c **do** $[\ell_1$: $S]$	$[\pi_i = \ell \wedge \pi_i' = \hat{\ell} \wedge \neg c \wedge pres] \vee$ $[\pi_i = \ell \wedge \pi_i' = \ell_1 \wedge c \wedge pres]$ Note the post-location of S is ℓ :

The i which appears as a subscript on π comes from the module in which the statement is located. If a transition τ maps a state s to a non-empty set of possible successor states, then τ is enabled on s; if it maps s to the null set then the transition is disabled on state s. A terminal state is one where no transition is enabled, and no location in the control variable is pointing to a **choose** statement; i.e. the program is not waiting for any player to make a choice.

Given a fixed decision for each player's choice points, an SMPL program's behaviour should be deterministic; otherwise it is not a valid SMPL program. This means that at any state, all the players, except one, should be at a **wait** statement, or should have terminated. This restriction ensures that we have a unique history of communication corresponding to a single state of the game.

SMPL is a Turing Complete language, and is sufficiently expressive to represent any game with a countable action space. One limitation relates to probabilities; there is no way to represent alternative outcomes which are chosen according to some probability distribution. To simulate this it would be necessary to have an external module which can input randomly generated numbers to the principal.

4. Bayesian Nash Equilibrium

Properties such as optimality of revenue or efficiency are typically implemented by means of an equilibrium; the game is designed so that the equilibrium play leads to the desired outcome. Two approaches to this are through dominant strategies or through a Bayesian Nash equilibrium (BNE). A common criticism of BNE implementation is that it is reliant on common knowledge of the probability distributions of the participants' types; this criticism is the "Wilson doctrine" [11, Section 8.2]. This is unlikely to be achieved in many scenarios, for example an auction for a rarely sold item. This would suggest that dominant strategy mechanisms should be preferred. Despite the possible criticisms, the BNE is thought to be useful in many scenarios. In scenarios in which sales of the same items are repeated frequently, a good knowledge of probabilities of bidders' types can be gathered. In this section we will take BNE as our example for the mechanism property our MUA language will capture, because it turns out that it is easier to check a BNE than a dominant strategy profile.

If we simply publish the rules of the game as the specification, then to determine the best strategy for participation, an agent could "solve" the game to find the equilibrium strategies. However, computing Nash equilibria is an open problem [8] and can be difficult [4, 15]. Even if an equilibrium is found, many games have multiple equilibria, in which case it is not clear which one the agents should follow. The solution proposed here is to augment the specification of the mechanism with information about its properties. In the case of an equilibrium, a part of the specification will include the designer's recommended equilibrium strategy for each agent; we use the SMPL both for specifying the rules of the game and the equilibrium strategies. In open systems there is no guarantee that a devious agent will not publish false information; therefore agents will need to verify for themselves that the published recommendation is indeed an equilibrium. We analyse the complexity of this verification problem here.

Note that we do not include the probability distributions of the players' types in the specification. Recall that the purpose of the MUA language is to allow agents to communicate mechanisms and their properties. If agents were to communicate the probability distributions then there would be no reason for the receiving agents to believe they were accurate, and indeed there would be an incentive for the transmitting agent to distort them. Therefore we must assume that the probability distributions are a prior common knowledge.

4.1. Verifying a Bayesian Nash Equilibrium

A static game G with n players is described by the tuple $\langle A_1, \ldots, A_n; T_1, \ldots, T_n; p; u_1, \ldots, u_n \rangle$; A_i is player i's action space, T_i his type space, p is the (common knowledge) probability distribution over players' types and u_i is player i's utility function. Player i will only be aware of his own type t_i; the type profile of all the other agents is given by some t_{-i}, and player i can calculate the likelihood of any particular t_{-i} occurring by computing $p(t_{-i}|t_i)$. The utility u_i is a function of all the actions taken by all players, and the type of player i, i.e. $u(a_1, \ldots, a_n; t_i)$. An agent i's (pure) strategy is a function mapping each of his possible types $t_i \in T_i$ to an action $a_i \in A_i$. The strategy profile $s^* = (s_1^*, \ldots, s_n^*)$ is a BNE if for each player i and for each of i's types $t_i \in T_i$, $s_i^*(t_i)$ gives the action $a_i \in A_i$ which maximises the expected value of player i's utility:

$$\sum_{t_{-i} \in T_{-i}} u_i(s_1^*(t_1), \ldots, a_i, \ldots, s_n^*(t_n); t_i) p(t_{-i}, t_i)$$

Which we abbreviate as: $E(u_i(a_i, s_{-i}^*(t_{-i}); t_i))$ (expected utility). If a_i maximises player i's expected utility, then player i will have no incentive to deviate from the action recommended by s^*. Calculating this sum will require us to consider every single possible configuration of the opponents' types t_{-i}. If there are n players each having m possible types, then there are m^{n-1} possible configurations for t_{-i}. If we are presented with a purported BNE and asked to check if it is indeed a BNE, we will need to calculate (for each player i and for each of i's types) the expected utility for each action he could take, and to compare it with expected utility of the recommended action. Algorithm 4.1 below checks a BNE, given as input a strategy profile s^*, and a static game G (as above).

Algorithm 4.1 (Check BNE)

1. for each player i do
2. for each type $t_i \in T_i$ do
3. for each action $a_i \in A_i \setminus \{s_i^*(t_i)\}$ do
4. calculate $E(u_i(a_i, s_{-i}^*(t_{-i}); t_i))$
5. and verify that it is $\leq E(u_i(s_i^*(t_i), s_{-i}^*(t_{-i}); t_i))$
6. od; od; od

Let us assume n players each having m possible types and a possible actions to choose from. This means that steps 4 and 5 are executed nma times. However, it is step 4 that dominates the complexity as it involves checking the m^{n-1} possible configurations for t_{-i}. Thus it is exponential in the number of players. In the worst case, s^* could assign a different action to every possible type, and the utility functions u_i could assign a different utility to every possible profile of actions, making the problem intractable. [1] Note that this is not as difficult as finding an unknown equilibrium however; the saving here is due to the fact that at step 4 we only check every possible configuration of *types* for the other agents, not their possible *strategies*; we assume that all the other agents are playing s^*, so their action is fixed once their type is known. If we had been searching for an equilibrium we should additionally have to consider every possible strategy profile which the opponents might play.

The above worst case complexity for verifying an equilibrium suggests that it is only practical in scenarios with a small number of agents. (See e.g. Section 4.2.) Despite this, in many practical cases the utility function is quite simple, and large chunks of the space of all possible configurations for t_{-i} are assigned the same utility. (See e.g. Section 4.5.)

4.2. Example: Two Player Double Auction

This auction example is from Gibbons [3, p. 158]. There is one buyer and one seller. The buyer's offer price is p_b and the seller's asking price is p_s. If $p_b \geq p_s$ then trade occurs at price $p = (p_b + p_s)/2$, otherwise no trade occurs. We specify this mechanism with the following SMPL program. The players' valuations are drawn from independent uniform distributions on [0,1]. Note that for automated checking we can only use integer ranges, so we replace the original type (and price) ranges of [0,1] with the integer range 0..999. The outcome here is a tuple for each agent. $\langle\langle 1, -p_t\rangle, \langle -1, p_t\rangle\rangle$ means that agent 1 gets (1) the item and pays $-p_t$ and agent 2 gives (-1) the item and receives p_t.

$$M_0 :: \left[\begin{array}{l} \textbf{wait } out_1 = 0;\ in_2 := 1;\ \textbf{wait } out_2 = 0 \\ p_b := out_1; p_s := out_2; p_t := (p_b + p_s)/2; \\ \textbf{if } p_b \geq p_s \textbf{ then } P := \langle\langle 1, -p_t\rangle, \langle -1, p_t\rangle\rangle \\ \textbf{else } P := \langle\langle 0, 0\rangle, \langle 0, 0\rangle\rangle \end{array} \right]$$
$$\Big\| M_1 :: \left[\textbf{choose } 0..999 \right]$$
$$\Big\| M_2 :: \left[\textbf{wait } in_2 = 0;\ \textbf{choose } 0..999 \right]$$

This mechanism has many BNEs. The following equilibrium gives the best expected gains for the players.

$$p_b(v_b) = \frac{2}{3} v_b + \frac{1}{12},\ \ p_s(v_s) = \frac{2}{3} v_s + \frac{1}{4}$$

Where v_b and v_s are the valuations (types) of the buyer and seller. This gives the following strategy programs:

$$Strat_1 :: \left[\ action := \tfrac{2}{3} \times value + \tfrac{1}{12} \right]$$
$$Strat_2 :: \left[\ action := \tfrac{2}{3} \times value + \tfrac{1}{4} \right]$$

In the case of this small number (two) of players, it is clearly feasible to calculate the expected utility of taking an action, as we have a single opponent. We simply go through each of our opponent's possible types, and find the action he would take according to the specified strategy, and then we can calculate our utility from the SMPL program.

[1] The same result holds for checking a dominant strategy profile, in this case we need to check over all possible bids for the opponents (rather than their types).

4.3. Games With Identical Players

A significant reduction in the number of possible configurations for t_{-i} can be achieved by restricting our attention to games with identical players. In our examples, players' types are independent so we can simply write $p_i(t_i)$ for the probability that player i takes type t_i. All the p_i are common knowledge. If all players have identical type spaces $T_i = T$, and probability distributions p_i, then the number of possible distinct type profiles for the $n-1$ opponents becomes equivalent to selecting $n-1$ objects from a choice of $|T|$ objects where each type can be chosen more than once. This is

$$\frac{(|T|+(n-1)-1)!}{(n-1)!(|T|-1)!} = \frac{(|T|+n-2)!}{(n-1)!(|T|-1)!}$$

Note that with the addition of each new player this increases by a factor of $\frac{|T|+n-1}{n}$, whereas in the non-identical players' worst case scenario the corresponding increase is $|T|$. In the identical case, when n becomes greater than $|T|$, increasing n increases the overall complexity by a factor of less than two, which is a significant saving. However, for large values of n the problem is still unlikely to be feasible.

4.4. Two Players Determine Outcome

We now look at the class of single-item auctions with n identical bidders, where a bidder wins only if he is the highest bidder, and the price the winner pays is determined by the first and second highest bids. No losing bidder makes or receives any payments. The bidders' types are simply valuations of the item in this auction; i.e. the bidder prefers outcomes where he wins the auction and the difference between his valuation for the item and what he has to pay for it is greatest. We also restrict the strategy space to those strategies where the bid is a monotonically increasing function of the valuation. This is a reasonable assumption in an auction. To simplify the analysis we construct the mechanism so that if there are two or more bidders having equal highest bids then no trade occurs. This is unrealistic in auctions, but rectifying it would be simple; however we choose not to as it complicates the presentation. Types range over the integers from v_{lo} to v_{hi}. Bids range over the real numbers. Suppose we want to calculate the expected value for some bidder i if his value is v_i and his bid is b_i. Clearly i's utility is zero if he does not win the auction. The other cases to consider are each of those where he wins and the second highest bidder takes each possible value $< b_i$. We use s^* to find the highest valuation v for which $s^*(v) < b_i$. Because of our monotonic strategy function assumption we know that in order for i to win, all opponents must have a valuation $\leq v$. Now for each integer value in the range $[v_{lo}, v]$ we will calculate the probability of the second highest bidder taking that value, and multiply it by the utility that bidder i would receive.

Let us start with the value v. We will calculate the probability that the second highest bidder has valuation v. The probability that all the $n-1$ opponent bidders have values $\leq v$ is $(\sum_{x=v_{lo}}^{v} p(x))^{n-1}$. The probability that none of these bidders bids v is

$$\left(\sum_{x=v_{lo}}^{v-1} p(x) / \sum_{x=v_{lo}}^{v} p(x)\right)^{n-1}$$

Therefore the probability that at least one of them bids v is the complement of this event:

$$1 - \left(\sum_{x=v_{lo}}^{v-1} p(x) / \sum_{x=v_{lo}}^{v} p(x)\right)^{n-1}$$

Multiplying these we get the probability that the second highest bid is v, let us call this event $high2 = v$:

$$prob(high2 = v) =$$

$$\left(\sum_{x=v_{lo}}^{v} p(x)\right)^{n-1} \left(1 - \left(\frac{\sum_{x=v_{lo}}^{v-1} p(x)}{\sum_{x=v_{lo}}^{v} p(x)}\right)^{n-1}\right)$$

Now we multiply this by the utility obtained $v_i - s^*(v)$:

$$prob(high2 = v)(v_i - s^*(v))$$

i.e. this is the utility obtained by bidder i if he bids x, and the second highest bidder bids just under him, multiplied by the probability of this occurring. Now we need to sum for all the possible second highest bids, where the second highest bidder takes all valuations in the integer range $[v_{lo}, v]$:

$$E(u_i(b_i, s^*_{-i}(t_{-i}); v_i)) = \sum_{y=v_{lo}}^{v} prob(high2 = y)(v_i - s^*(y))$$

where v is the highest valuation for which $s^*(v) < b_i$. Now we will look at the time complexity of an algorithm to compute the expected utility. We are assuming that we have a vector which stores all the discrete $p(x)$ values for x in the integer range $v_{lo} \ldots v_{hi}$. Clearly the summations $\sum_{x=v_{lo}}^{z} p(x)$ for $z = v_{lo} \ldots v_{hi}$ are being used many times, so it will be efficient to calculate them all and store them in a vector Sp such that $Sp_z = \sum_{x=v_{lo}}^{z} p(x)$. Filling this vector with values can be done in $v_{hi} - v_{lo}$ steps. Similarly we can create a vector $P2$ such that $P2_z = prob(high2 = z)$. given that vector Sp is already computed, vector $P2$ can be computed in time polynomial in $n-1$. Now the final sum for $E(u_i(b_i, s^*_{-i}(t_{-i}); v_i))$ can be computed in polynomial time if the strategy function s^* can be computed in polynomial time (a reasonable assumption).

This result does not hold in the case of checking a dominant strategy profile. Dominant strategy profiles are harder

to check because we can make no assumptions about the opponents' strategies; we would need to ensure that our player's strategy gives him the best outcome, for whatever strategy profile is played by the opponents.

4.5. Example With Multiple Bidders

The following parameterised SMPL program represents a first price sealed bid auction. It is parameterised by n, the number of bidders. Again, we use the integer range 0..999 for the bids. Note that module M_{n+1} is the module deciding the price the winning bidder will pay. The principal module M_0 determines the winner, and records the two highest bids, passing them on to module M_{n+1} as a tuple. Module M_{n+1} then simply selects the first part of this tuple (first price) and returns it to M_0. The outcome here only assigns a value for the winning agent, all other P_i will remain zero.

$$
\begin{array}{l}
M_0 :: \left[\begin{array}{l}
\textbf{wait } out_1 = 0;\ i := 2; \\
\textbf{while } i \leq n \textbf{ do } \left[\begin{array}{l} in_i := 1;\ \textbf{wait } out_i = 0 \\ i := i + 1 \end{array}\right] \\
i := 1; \\
\textbf{while } i \leq n \textbf{ do} \\
\left[\begin{array}{l}
\textbf{if } out_i > high1 \textbf{ then } \left[\begin{array}{l} tie := 0 \\ high1 := out_i \\ winner := i \end{array}\right] \\
\textbf{else} \\
\left[\begin{array}{l} \textbf{if } out_i = high1 \textbf{ then } tie := 1 \\ \textbf{if } out_i > high2 \textbf{ then } high2 := out_i \end{array}\right] \\
i := i + 1
\end{array}\right] \\
in_{n+1} := \langle high1, high2 \rangle;\ \textbf{wait } out_{n+1} = 0 \\
\textbf{if } tie = 0 \textbf{ then } P_{winner} := \langle 1, -out_{n+1} \rangle \\
\textbf{else } P_{winner} := \langle 0, 0 \rangle
\end{array}\right] \\
\Big\| \ M_1 :: \left[\ \textbf{choose } 0..999\right] \\
\Big\| \ M_2 :: \left[\ \textbf{wait } in_2 = 0;\ \textbf{choose } 0..999\right] \\
\quad \vdots \\
\Big\| \ M_n :: \left[\ \textbf{wait } in_n = 0;\ \textbf{choose } 0..999\right] \\
\Big\| \ M_{n+1} :: \left[\textbf{wait } in_{n+1} = 0;\ x := in_{n+1};\ out_{n+1} := x_1\right]
\end{array}
$$

The following is the equilibrium strategy for each agent, again parameterised by n:

$$Strat :: \left[\ action := (n-1) \times value/n\right]$$

The modularisation in the above auction allows it to be easily generalised to cover the class of single-item auctions with identical bidders, where a bidder wins only if he is the highest bidder, and the price the winner pays is determined by the two highest bids (i.e. any function of the two highest bids). Agents can call on the trusted library to generate a version of this specification, instantiating n for any desired number of agents, and leaving M_{n+1} blank. This means that the pricing rule can be customised as desired; it can only use the two highest bids however, as that is all that is input to the module M_{n+1}. Hence the mechanism's equilibrium can still be verified using the method of Section 4.4. We must also perform a check on the published strategy to ensure it is a monotonically increasing function of the valuation; to do this we simply check through all types that can be input to the strategy and verify that the action (bid) is a monotonically increasing function.

5. Groves Mechanisms

We can apply a similar modularisation technique to the class of Groves mechanisms. Groves mechanisms can be used in a wide variety of auctions, including combinatorial auctions. In this case we can have separate modules for the pricing rule and the allocation rule. The allocation problem for combinatorial auctions is known to be NP-complete and inapproximable [7]. Verifying this portion of the mechanism is therefore not feasible at run-time, and so agents could rely on a trusted module from the library to perform the allocation. The pricing rule is left open for the agents to specialise for their particular scenario, with some restrictions as described below. It is the pricing rule that will then determine the remaining properties of the auction.

Groves mechanisms have the property of being strategyproof (i.e. dominant strategy to bid truthfully); the allocation is done optimally (outcome o^*), and PT is the payment transferred to each agent i:

$$PT_i = \sum_{j \in N \setminus \{i\}} \hat{v}_j(o^*) - C_{-i}$$

Where N is the set of agents, $\hat{v}_j(o^*)$ is agent j's declared valuation on the allocation and C_{-i} is some quantity independent of agent i. Now we can generate a specification for a mechanism which has a customisable pricing module. The principal module M_0 of the mechanism polls the agents for their bids, and calls the allocation module. Then it calls the pricing module to calculate a price for each agent, but whenever the pricing module is called to calculate the payment for agent i, it is only passed the declared valuations of the agents excluding i. This guarantees that the mechanism as a whole is a Groves mechanism, and hence is strategyproof.

We would also like to guarantee ex-post individual rationality (this means that the agents are guaranteed not to get negative utility by participating in the mechanism). To guarantee this we need to ensure that the following holds:

$$\hat{v}_i(o^*) + PT_i \geq 0$$

This guarantee can be achieved by including a check in the module of the principal agent, such that after calling the pricing module to get the value C_{-i}, it determines if the inequality above holds with the transfer PT_i; if it does not hold then the principal cancels the auction and no sale is made, and all agents receive zero utility. In this way we can have a protocol in our library which has a fixed principal module and allocation rule, but which allows the agents to propose their own customised pricing module; with any arbitrary pricing module, the mechanism still guarantees ex-post individual rationality and is strategyproof. Most importantly, this is achieved without the agents needing to do any checking themselves at run-time. This could be used to implement a broad class of mechanisms, for example the VCG, or variants where additional transfers occur [1].

6. Conclusion and Future Work

We have broken the problem of achieving common knowledge in mechanisms into two parts: semantic interoperation, and guarantees of compliance. The guarantees of compliance can be achieved relatively easily, but the semantic interoperation is more problematic. This problem has been further decomposed into knowledge of the rules of a mechanism, and knowledge of its properties. We have provided a simple language which could be used to capture a class of mechanisms, and to communicate the rules to agents. We have shown that the problem of determining the properties of a mechanism is not feasible in the most general case; however, in certain cases we can achieve it by communicating some information about the properties and/or placing certain restrictions on the specification.

Throughout this work there is a tradeoff between having a specific mechanism with well known properties, and having the flexibility to use customised mechanisms (within some class of mechanisms), whose properties may be more difficult to determine. Future progress will need to push the boundaries: increasing both the flexibility and the knowledge of the properties. One extension would be to employ some of the abstraction techniques used by the model checking community. For example, our checking of the equilibrium exhaustively checks each possible bid value, when in fact all bids below or above a critical threshold could be lumped together. Ultimately we want to determine the limits of the approach. We expect that some classes of auctions will prove difficult to verify automatically, and that some auctions will only be verifiable if their complexity is limited. When these limits are determined, they can be used to inform the design of mechanisms for agent scenarios where semantic interoperation is desired.

Acknowledgements: We thank the UK EPSRC for funding this project under grant EP/D02949X/1. Thanks also to the anonymous reviewers for helpful suggestions.

References

[1] R. Cavallo. Optimal decision-making with minimal waste: Strategyproof redistribution of vcg payments. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, May*. ACM Press, 2006.

[2] R. Dash, N. Jennings, and D. C. Parkes. Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, pages 40–47, November 2003. Special Issue on Agents and Markets.

[3] R. Gibbons. *A Primer in Game Theory*. Prentice Hall, 1992.

[4] G. Gottlob, G. Greco, and F. Scarcello. Pure nash equilibria: Hard and easy games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.

[5] F. Guerin. An algorithmic approach to specifying and verifying subgame perfect equilibria. In *Proceedings of the Eighth Workshop on Game Theoretic and Decision Theoretic Agents (GTDT-2006), Hakodate, Japan*, 2006.

[6] M. He, N. Jennings, and L. Ho-Fung. On agent-mediated electronic commerce. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):985– 1003, July-Aug 2003.

[7] D. Lehmann, R. Mueller, and T. Sandholm. The winner determination problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, pages 93–118. MIT Press, 2006.

[8] C. Papadimitriou. Algorithms, games and the internet. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 749–753, 2001.

[9] D. C. Parkes. iBundle: an efficient ascending price bundle auction. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 148 – 157. ACM Press New York, NY, USA, 1999.

[10] T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts at Amherst, Dept. of Computer Science, 1996.

[11] T. Sandholm. Automated mechanism design: A new application area for search algorithms. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, 2003.

[12] T. Sandholm and A. Gilpin. Automated design of multistage mechanisms. In *Proceedings of the First International Workshop on Incentive Based Computing,*. ACM Press, 2005.

[13] T. Sandholm and A. Gilpin. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, May*. ACM Press, 2006.

[14] T. Sandholm and S. Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145(1-2):33–58, 2003.

[15] B. von Stengel. Computing equilibria for two-person games. In *Handbook of Game Theory with Economic Applications, Vol. 3, eds. Aumann and Hart*. Elsevier, Amsterdam, 2002.

[16] M. Yokoo, Y. Sakurai, and S. Matsubara. Bundle design in robust combinatorial auction protocol against false-name bids. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1095–1101. (IJCAI), Seattle, WA., 2001.