Automated Analysis of Auction Traces

Mark Hoogendoorn[1⋆] and Maria Gini[2⋆]

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
mhoogen@cs.vu.nl,
[2] University of Minnesota, Minneapolis, MN, USA
gini@cs.umn.edu

Abstract. When agents participate in an auction, either as buyers or sellers, it is important to be able to verify compliance to auction protocols and correctness of auction clearing. We propose a method for such a verification which is based on analyzing execution traces of the auction. Our method has the advantage that it does not require access to the internal of the agents, hence it is applicable to any auction, even auctions with human and agent participants, as long as the auction trace is available. The approach is based on an expressive temporal logic in which properties for auction types and for agent strategies are specified. Examples of trace analysis are used to illustrate the approach. Finally, experimental results are presented using synthetic data.

Key words: formal verification, auction protocals, trace-based

1 Introduction

Auctions are a popular means to distribute tasks or sell items within multi-agent environments (see e.g. [17] and [14]). A variety of auction types are available, such as single item first price, single item Vickrey auction [16], and combinatorial auctions [8]. Each auction type has specific constraints which specify the rules governing the auction.

Analyzing whether agents comply to the specified rules, and how effective the agents strategies are is essential for reliable and effective auctioning systems. One way of performing this analysis is to use model checking[2] techniques (see e.g. [3] and [10]). Unfortunately, model checking requires access to the internal specifications of the agents, which are generally unavailable, especially in systems open to participation from multiple agents and/or humans.

We present a trace-based approach to analyze compliance of agents to auction protocols. A trace consists of all the communications that occur between agents within the auctioning system. Hence, only external information is expressed in a trace. The approach uses an expressive temporal language, called temporal trace language (TTL) [5], which enables expressing properties with time parameters (for instance, that a bid is submitted before a certain deadline). We use a checking tool, called TTL Checker, for

⋆ Partial support provided by the National Science Foundation under grant IIS-0414466

[2] To avoid confusion with our approach, in this paper we interpret the term model checking as checking all possible execution paths. In principle our approach could be seen as model checking for a single execution trace.

automated analysis of the properties against such traces. Using this temporal logic, we specify properties of compliance to auction protocols for several auction types.

Our approach does not require the bidding to be done by agents, as long as auction traces are available. We envision incorporating trace analysis in a supervisor agent that acts as the security and exchange commission, or in agents that verify compliance to auction protocols in a auction testbed, such as e-bay. Note that our approach can show whether certain properties are satisfied for a given set of traces, but cannot guarantee these properties will be satisfied in all future auctions.

The paper is organized as follows. First, the temporal logic used throughout this paper is introduced, followed by the ontology, properties, and examples of analysis of traces for various types of auctions. This is followed by results of checking properties upon synthetic data, related work, conclusions and suggestions for future work.

2 The TTL Language

This Section introduces the temporal logic used to represent the desired properties in the auction. A temporal logic has been chosen because time parameters play an essential role within auctions; for instance, auctions often specify when offers can be sent out, or when the auction ends.

In TTL [5], ontologies for states are formalized as sets of symbols in sorted predicate logic. For any ontology Ont the ground atoms form the set of basic state properties BSTATPROP(ONT). Basic state properties can be defined by nullary predicates (or proposition symbols), such as auction, or by n-ary predicates (with $n > 0$), like bid_deadline(item_1, 5). The *state properties* based on an ontology ONT are formalized by the propositions made from BSTATPROP(ONT) using conjunction ($\&$), negation ($\neg$), disjunction ($|$), and implication ($\rightarrow$) as connectors. They constitute the set STATPROP(ONT).

In order to express dynamics in TTL, important concepts are *states*, *time points*, and *traces*. A *state* S is an indication of which basic state properties are true and which are false, i.e., a mapping S: BSTATPROP(ONT) $\rightarrow \{$TRUE, FALSE$\}$. The set of all possible states for ontology ONT is denoted by STATES(ONT). A fixed *time frame* T is assumed which is linearly ordered. Hence, a trace γ over a state ontology ONT and time frame T is a mapping γ : T $\rightarrow$ STATES(ONT), i.e., a sequence of states γ_t (T $\in$ T) in STATES(ONT). The set of all traces over ontology ONT is denoted by TRACES(ONT).

The set of *dynamic properties* DYNPROP(ONT) is the set of temporal statements that can be formulated with respect to traces based on the state ontology ONT in the following manner. Given a trace γ over state ontology ONT, a certain state at time point t is denoted by state(γ, t). States can be related to state properties via the formally defined satisfaction relation, indicated by the infix predicate $\models$, which is comparable to the HOLDS-predicate in Situation Calculus. Thus, state(γ, t) $\models$ p denotes that state property p holds in trace γ at time t. Likewise, state(γ, t) $\not\models$ p denotes that state property p does not hold in trace γ at time t. Based on these statements, dynamic properties can be formulated using the usual logical connectives such as $\neg$, $\&$, $|$, $\Rightarrow$ and the quantifiers $\forall$, $\exists$ (e.g., over traces, time and state properties).

Analysis of whether certain TTL properties are satisfied for a set of traces can be done in an automated fashion using the TTL Checker. For more details on the formal syntax and semantics of TTL and the TTL Checker software see [5].

3 Single Item First-Price Sealed-Bid Auction (SIFP)

The first auction type we describe is the single item first-price auction with sealed bids. In order to represent the interactions that take place, we use the sorts and predicates specified in Table 1.

Sort	**Explanation**
AGENT	An agent within the system
TIME	Sort representing time
ITEM_ID	Identifier of an item to be sold
PRICE	Sort representing the price
Predicate	**Explanation**
offer_item: AGENT × ITEM_ID	An agent offers a specific item
bid_deadline: ITEM_ID × TIME	The deadline for submitting bids for the item
earliest_consider_time: ITEM_ID × TIME	Time after which bids for the item are considered
earliest_bid_time: ITEM_ID × TIME	The earliest time at which bids can be sent for the item
send_bid: AGENT × ITEM_ID × PRICE	An agent sends a bid for the item with a certain price
send_bid_award: AGENT × AGENT × ITEM_ID	The first agent awards a bid for the item to the second agent
payment: AGENT × AGENT × ITEM_ID × PRICE	The first agent pays the second agent the specified price for the item

Table 1. Sorts and predicates for single item first-price auction

A number of properties can be specified using the ontology. In this paper we limit our presentation to properties on compliance to protocols and a property on the correctness of the winner determination process.

3.1 Compliance to Protocol

We show three properties related to protocol compliance. Property P1 states that bids cannot be submitted before the earliest offer time specified for the item. Property P2 states that awards for bids cannot be sent before the specified earliest consideration time. Property P3 states that the price offered for the bid which is awarded has to be paid to the seller.

P1SIFP: Non-early Submitting of Bid

$\forall\gamma$: TRACE, t1, t2:TIME, a1:AGENT, i:ITEM_ID
[[state(γ, t1) $\models$ offer_item(a1, i) $\&$ state(γ, t1) $\models$ earliest_bid_time(i, t2)]
$\Rightarrow \neg\exists$t':TIME<t2, a2:AGENT, p:PRICE [state(γ, t') $\models$ send_bid(a2, i, p)]]

P2SIFP: Non-early Awarding of Bid

$\forall\gamma$:TRACE, t1, t2:TIME, a1:AGENT, i:ITEM_ID
[[state(γ, t1) $\models$ offer_item(a1, i) $\&$ state(γ, t1) $\models$ earliest_consider_time(i, t2)]
$\Rightarrow \neg\exists$t':TIME<t2, a2:AGENT [state(γ, t') $\models$ send_bid_award(a1, a2, i)]]

P3SIFP: Payment of Offered Price

$\forall\gamma$:TRACE, t1, t2, t3:TIME, a1, a2:AGENT, i:ITEM_ID, p:PRICE
[[state(γ, t1) $\models$ offer_item(a1, i) $\&$ state(γ, t2) $\models$ send_bid(a2, i, p) $\&$
state(γ, t3) $\models$ send_bid_award(a1, a2, i)]
$\Rightarrow \exists$t':TIME>t3 [state(γ, t') $\models$ payment(a2, a1, i, p)]]

3.2 Correct Winner Determination

The correctness of winner determination is straightforward in a first-price sealed bid auction: the highest price bid should be selected. This is stated in property P4: If a certain item is offered, and a bid is awarded with price p1, then there should not exist an earlier time point at which a bid with a higher price was submitted. Note that this property does not specify full compliance to the protocol (e.g. whether awards are sent at the appropriate time points).

P4SIFP: Correct Winner Determination

$\forall\gamma$:TRACE, t1, t2, t3:TIME, a1, a2:AGENT, i:ITEM_ID, p1:PRICE
[[state(γ, t1) $\models$ offer_item(a1, i) $\&$ state(γ, t2) $\models$ send_bid(a2, i, p1) $\&$
state(γ, t3) $\models$ send_bid_award(a1, a2, i)]
$\Rightarrow \neg\exists$t':TIME<t3, a3:AGENT, p2:PRICE [state(γ, t') $\models$ send_bid(a3, i, p2) $\&$ p2 $>$ p1]]

3.3 Example of Trace Analysis

We show now how the properties expressed earlier can be checked against empirical traces. Figure 1 shows an example of such a trace. The left side shows the atoms expressed in the ontology introduced before, the right side shows a time line where a dark box indicates that the atom is true at that time point.

The trace shows a seller, called seller_a who offers a particular item (item_1):
offer_item(seller_a, item_1)

Time parameters are set for the auction, namely the earliest time at which bids can be made, the deadline for bidding, and the earliest time at which bids will be considered:
earliest_bid_time(item_1, 4)
bid_deadline(item_1, 5)
earliest_consider_time(item_1, 8)

Two buyers respond. One bids a price of 5 for the item, whereas the other bids 6:

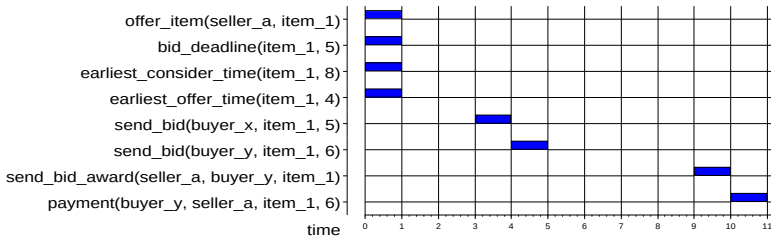

Fig. 1. A trace for first price sealed bid auction

send_bid(buyer_x, item_1, 5)
send_bid(buyer_y, item_1, 6)

The seller awards the bid to the buyer who bid a price of 6 for the item:

send_bid_award(seller_a, buyer_y, item_1)

Eventually, the buyer pays the money to the seller:

payment(buyer_y, seller_a, item_1, 6)

For this trace, properties P2SIFP-P4SIFP are satisfied, but property P1SIFP is not because buyer_x sends a bid before time point 4.

Figure 2 shows a similar trace, except here there is one more error. Property P1SIFP is again not satisfied since buyer_x did not submit its bid after time point 4 (but between 3 and 4). Furthermore, property P3SIFP does not hold since the payment is set to the second highest bid, which is not according to the definition of first-price auction.

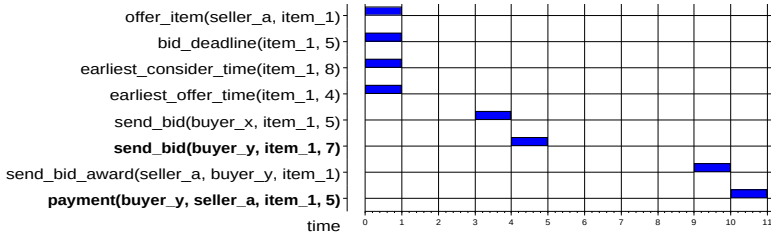

Fig. 2. A trace for first price sealed bid auction which does not satisfy properties P1 and P3

4 Reverse Sealed Bid Combinatorial Auction

This section shows properties for a more complex auction type, namely a reverse combinatorial auction. The auction is reverse since in this case the auctioneer is the buyer, and is combinatorial since each bid can include multiple items. The sorts and the ontology used to express the occurrences in such an auction are specified in Table 2.

The ontology is more general than strictly needed. For instance we included an identifier for the Request for Quotes (RFQ), so we can handle multiple RFQs by multiple agents. Next we will show example properties and example traces where the properties are checked.

Sort	**Explanation**
AGENT	An agent within the system
RFQ_ID	An identifier for a Request for Quotes (RFQ)
TIME	Sort representing time
ITEM_ID	Identifier of an item
BID_ID	Identifier of a bid
PRICE	Sort representing the price
Predicate	**Explanation**
send_rfq: AGENT × RFQ_ID	An agent sends an RFQ
rfq_bid_deadline: RFQ_ID × TIME	The deadline for the bidding process for items in the RFQ
rfq_earliest_consider_time: RFQ_ID × TIME	The time after which the bids for the RFQ will be considered
rfq_earliest_bid_time: RFQ_ID × TIME	The earliest time at which bids for tasks for items in the RFQ can be submitted
in_rfq: RFQ_ID × ITEM_ID	An item with a certain id is part of an RFQ
send_bid: AGENT × BID_ID × RFQ_ID	An agent sends a bid for items within an RFQ
bid_price: BID_ID × PRICE	The price of a bid
in_bid: BID_ID × TASK_ID	An item is included in a bid
send_bid_award: AGENT × BID_ID	A bid is awarded by an agent

Table 2. Sorts and predicates for reverse sealed bid combinatorial auction

4.1 Compliance to Protocol

Below, we specify two properties related to protocol compliance, which are the reverse combinatorial auction variants of the properties P1 and P2 for non-combinatorial auctions.

P1C: Non-early Submitting of Bid
$\forall\gamma$:TRACE, t1, t2:TIME, a1:AGENT, r:RFQ_ID
[[state(γ, t1) $\models$ send_rfq(a, r) & state(γ, t1) $\models$ rfq_earliest_bid_time(r, t2)]
$\Rightarrow \neg\exists$t':TIME $<$ t2, a2:AGENT, b:BID_ID [state(γ, t') $\models$ send_bid(a2, b)]]

P2C: Non-early Awarding of Bid
$\forall\gamma$:TRACE, t1, t2:TIME, a1:AGENT, r:RFQ_ID
[[state(γ, t1) $\models$ send_rfq(a, r) & state(γ, t1) $\models$ rfq_earliest_consider_time(r, t2)]
$\Rightarrow \neg\exists$t':TIME $<$ t2, b:BID_ID [state(γ, t') $\models$ send_bid_award(a, b)]]

4.2 Correct Winner Determination

Assuming that all the items have to be acquired, in order for an evaluation to be correct the set of awarded bids must fully cover the items specifi ed in the RFQ (and no more

than that), and must be the cheapest set with such full coverage. First we specify the combination of bids that has been awarded, which simply is the set of all bids awarded:

awarded_combination(γ:TRACE, t:TIME, r:RFQ_ID, a:AGENT, bc:BID_COMB) $\equiv$
$\forall$b:BID_ID
 [[$\exists$t1:TIME $\geq$ t, a2:AGENT
  state(γ, t1) $\models$ send_bid(a2, b, r) $\&$
  $\exists$t2:TIME $>$ t1 [state(γ, t2) $\models$ send_bid_award(a, b)]]
  $\Rightarrow$ b $\in$ bc]
 $\&$ [[$\exists$ t3:TIME $\geq$ t, a2:AGENT
   state(γ, t3) $\models$ send_bid(a2, b, r) $\&$
   $\neg\exists$t4:TIME $\geq$ t3 [state(γ, t4) $\models$ send_bid_award(a, b)]]
   $\Rightarrow$ b $\notin$ bc]

Next we specify the price of the combination. Note that the $case(a, b, c)$ operator works as follows: if condition a holds, it evaluates to b, and otherwise to c.

combination_price(γ:TRACE, t:TIME, bc:BID_COMB, p:PRICE) $\equiv$
($\sum_{\text{b:BID\_ID}\in\text{bc},\text{p2:PRICE},\text{t2:TIME}\geq\text{t}}$ case(state(γ, t2) $\models$ bid_price(b, p2), p2, 0))
= p

A bid combination is considered valid if all bids in the set have been sent after time t, and each item in the RFQ is included in at least one bid, and there is no other bid in the combination for which this holds.

valid_combination(γ:TRACE, t:TIME, r:RFQ_ID, bc:BID_COMB) $\equiv$
$\forall$b:BID_ID $\in$ bc
 [$\exists$a:AGENT, t2:TIME $\geq$ t
  state(γ, t2) $\models$ send_bid(a, b)] $\&$
$\forall$t:TASK_ID
 [state(γ, t) $\models$ in_rfq(r, t)
  $\Rightarrow$ [$\exists$t3:TIME, b2:BID $\in$ bc
   state(γ, t3) $\models$ in_bid(b2, t) $\&$
   $\neg\exists$t4:TIME, a2:AGENT, b3:BID $\neq$ b2
   [b3 $\in$ bc $\&$
   state(γ, t2) $\models$ send_bid(a2, b3) $\&$
   state(γ, t4) $\models$ in_bid(b3, t)]]

Given the definitions above, we can now specify property P3, which specifies that an evaluation is correct for the set of traces if there exists no combination that is awarded and that either is not valid or there exists another valid combination with a lower price.

P3C: Correct Winner Determination
$\forall\gamma$:TRACE t:TIME, a:AGENT, bc:BID_COMB, r:RFQ_ID, p:PRICE
 [[state(γ, t) $\models$ send_rfq(a, r) $\&$
  awarded_combination(γ, t, r, a, bc) $\&$
  combination_price(γ, t, bc, p)] $\Rightarrow$
  [valid_combination(γ, t, r, bc) $\&$
  $\neg\exists$bc2:BID_COMB $\neq$ bc, p2:PRICE
   [valid_combination(γ, t, r, bc2) $\&$ combination_price(γ, t, bc2, p2) $\&$ p2 $<$ p1]]]

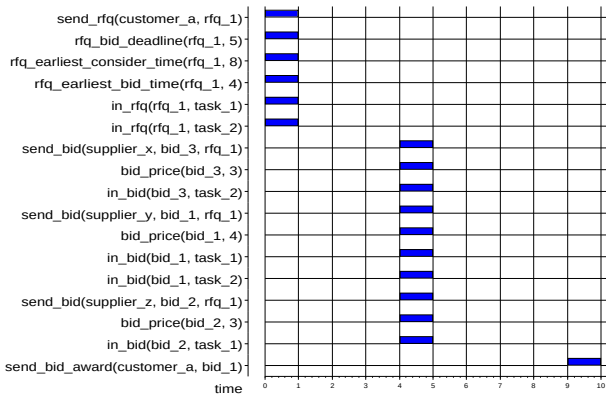

Fig. 3. A trace of a combinatorial auction

4.3 Example of Trace Analysis

This section shows how the properties expressed in the previous sections can be analyzed upon an empirical trace. Figure 3 shows an example of such an empirical trace.

All properties specified (i.e. properties P1C-P3C) have been checked against this trace. Properties P1C and P2C regarding the protocol hold since the seller acts according to the time line specified. Property P3C is also satisfied, because the offer of supplier_y is cheaper than the combination of the other two offers (i.e. a cost of 4 vs. 6).

5 Reverse First Price Sealed-Bid Combinatorial Auction with Time Windows (RCATW)

The second type of auction we consider in this paper is an auction of a combinatorial type with explicit representation of time windows and precedence constraints between tasks. The auction is reverse since in this case the auctioneer is the buyer, and is combinatorial since each bid can include multiple items. The MAGNET system [7] is an example of such an auction, which is used for task allocation. The sorts and the ontology used to express the occurrences in such an auction are specified in Table 3.

5.1 Compliance to Protocol

The properties for the compliance of protocol are in this case identical to the reverse combinatorial auction presented in Section 4.

5.2 Correct Winner Determination

Also here large part of the properties can be reused, except for the definition of a valid bid and a valid combination. To define a valid bid combination, first we define that a

Sort	**Explanation**
AGENT	An agent within the system
RFQ_ID	An identifier for a Request for Quotes (RFQ)
TIME	Sort representing time
TASK_ID	Identifier of a task
BID_ID	Identifier of a bid
PRICE	Sort representing the price
DURATION	Sort representing the duration of a task
Predicate	**Explanation**
send_rfq: AGENT × RFQ_ID	An agent sends an RFQ
rfq_bid_deadline: RFQ_ID × TIME	The deadline for submitting bids for the tasks in the RFQ
rfq_earliest_consider_time: RFQ_ID × TIME	The time after which bids for the RFQ will be considered
rfq_earliest_bid_time: RFQ_ID × TIME	The earliest time at which bids for tasks in the RFQ can be submitted
in_rfq: RFQ_ID × TASK_ID	A task is part of an RFQ
send_bid: AGENT × BID_ID × RFQ_ID	An agent sends a bid for tasks in an RFQ
bid_price: BID_ID × PRICE	The price of a bid
in_bid: BID_ID × TASK_ID	A task is included in a bid
send_bid_award: AGENT × BID_ID	A bid is awarded by an agent
rfq_precedence_constraint: RFQ_ID × TASK_ID × TASK_ID	The first task must end before the second starts
rfq_task_earliest_start_time: RFQ_ID × TASK_ID × TIME	The earliest time in the RFQ when execution of a task can start
rfq_task_latest_start_time: RFQ_ID × TASK_ID × TIME	The latest time in the RFQ when execution of a task can start
rfq_task_latest_end_time: RFQ_ID × TASK_ID × TIME	The latest time in the RFQ at which execution of a task can end
bid_task_earliest_start_time: BID_ID × TASK_ID × TIME	The earliest time in a bid the execution of the task will start
bid_task_latest_start_time: BID_ID × TASK_ID × TIME	The latest time in a bid the execution of the task will start
bid_task_duration: BID_ID × TASK_ID × DURATION	The duration in a bid of the execution of the task

Table 3. Predicates used in reverse first price sealed-bid combinatorial auction with time windows

bid is valid if the execution time points for all the tasks in the bid fit within the time windows specified in the RFQ.

valid_bid(γ:TRACE, t:TIME, r:RFQ_ID, b:BID_ID) $\equiv$

$\forall$tid:TASK_ID, t1:TIME
 [state(γ, t1) $\models$ in_bid(b, t)
 $\Rightarrow \exists$t2-t6:TIME, d:DURATION
    [state(γ, t1) $\models$ bid_task_earliest_start_time(b, tid, t2) $\&$
    state(γ, t1) $\models$ bid_task_latest_start_time(b, tid, t3) $\&$
    state(γ, t1) $\models$ bid_task_duration(b, tid, d) $\&$
    state(γ, t) $\models$ rfq_task_earliest_start_time(r, tid, t4) $\&$
    state(γ, t) $\models$ rfq_task_latest_start_time(r, tid, t5) $\&$
    state(γ, t) $\models$ rfq_task_latest_end_time(r, t6) $\&$
    (t2 $\geq$ t4) & (t3 $\leq$ t5) & (t6 $\geq$ (t3 + d))]]

A bid combination is considered valid if each bid has been sent and each bid is valid. In addition, all tasks specified in the RFQ should be covered by the bids (i.e. no free disposal), tasks should not occur multiple times within the bid combination, and the precedence constraints have to be met. Note that there can be multiple valid combinations of bids per auction.

valid_combination(γ:TRACE, t:TIME, r:RFQ_ID, bc:BID_COMB) $\equiv$

$\forall$b:BID_ID $\in$ bc
 [valid_bid(γ, t, r, b) $\&$
  $\exists$a:AGENT, t2:TIME $\geq$ t [state(γ, t2) $\models$ send_bid(a, b)]] $\&$
$\forall$tid:TASK_ID
 [state(γ, t) $\models$ in_rfq(r, tid)
 $\Rightarrow$ [$\exists$t3:TIME, b2:BID $\in$ bc
    state(γ, t3) $\models$ in_bid(b2, tif) $\&$
    $\neg\exists$t4:TIME, a2:AGENT, b3:BID $\neq$ b2
    [b3 $\in$ bc $\&$ state(γ, t2) $\models$ send_bid(a2, b3) $\&$
    state(γ, t4) $\models$ in_bid(b3, tid)] $\&$
    $\forall$tid2:TASK_ID $\neq$ tid
    [state(γ, t) $\models$ precedence_constraint(r, tid, tid2)
    $\Rightarrow \exists$ t5, t6, t7:TIME, d:DURATION, b4:BID_ID $\in$ bc
       [state(γ, t5) $\models$ in_bid(b4, tid2) $\&$
       state(γ, t5) $\models$ bid_task_earliest_start_time(b4, tid2, t6) $\&$
       state(γ, t3) $\models$ bid_task_latest_start_time(b2, tid, t7) $\&$
       state(γ, t3) $\models$ bid_task_duration(b2, tid, d) $\&$
        t6 $\geq$ (t7 + d)]]]]

Given the definitions above, property P3C can simply be reused.

5.3 Example of Trace Analysis

Figure 4 shows an example trace of a reverse first-price sealed-bid combinatorial auction with time windows and precedence constraints. As can be seen in the trace, the following time window constraints are specified for the tasks in the RFQ:

rfq_task_earliest_start_time(rfq_1, task_1, 10)
rfq_task_latest_start_time(rfq_1, task_1, 12)
rfq_task_latest_end_time(rfq_1, task_1, 15)

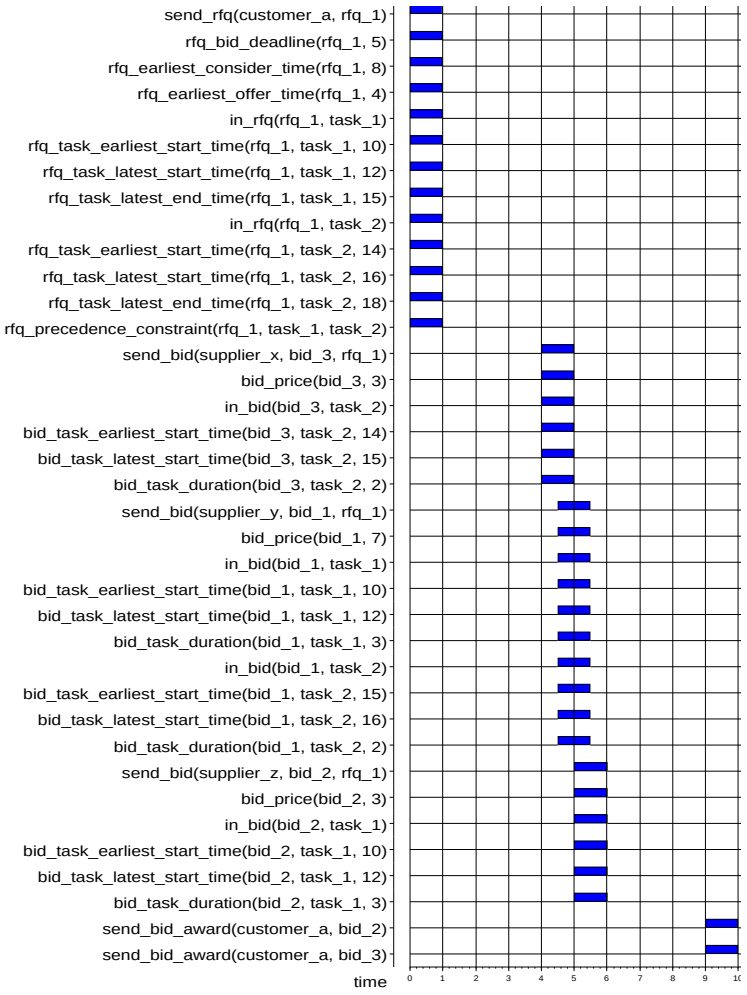

Fig. 4. A trace of a combinatorial auction with time windows

rfq_task_earliest_start_time(rfq_1, task_2, 14)
rfq_task_latest_start_time(rfq_1, task_2, 16)
rfq_task_latest_end_time(rfq_1, task_2, 18)

Furthermore, one precedence constraint is specified, indicating that task_1 should be completed before task_2 can start:

rfq_precedence_constraint(rfq_1, task_1, task_2)

Several bids are received in response to the RFQ. In bid_1 both tasks are included, whereas bid_2 and bid_3 merely cover task_1 and task_2 respectively. The time windows included in the bid can be seen in the trace. bid_2 and bid_3 are awarded for a total price of 6, which is cheaper than bid_1 which costs 7. Evaluation of the trace reveals that P1C and P2C are satisfied but P3C is not. Although all the time windows

included in the bids do comply with the times specified in the RFQ, the precedence constraint does not; the latest start time plus the expected duration of task_1 is later than the earliest start time for task_2. As a result, P3C is not satisfied.

6 Experiments

In order to investigate how scalable the approach is we have generated numerous synthetic traces and checked the properties specified in the previous sections using these traces. Two auction setting presented earlier are addressed, namely the single item first-price sealed-bid auction, and the reverse sealed-bid combinatorial auction with time windows. These have been chosen to show how the approach performs for a relatively simple auction and for the most complex one addressed in this paper.

6.1 Single Item First-Price Sealed-Bid Auction

The first auction type considered is the single item first price sealed bid auction. For this case, traces have been generated with a varying number of buyer agents participating in the auction. For each setting of the number of agents, 50 traces have been generated, and the following number of agents have been tested: $\{1, 5, 10, 25, 50, 75, 90\}$. Hereby, the agents bid a price which is generated from a random distribution. Furthermore, the times at which they submit their bids, and receive awards precisely comply to the times communicated by the seller (which are fixed throughout the runs). Finally, the evaluation of the bids by the buyer is done by exhaustive search, resulting in a correct evaluation. Hence, the generated traces are traces in which all properties are satisfied, so these are worst case scenarios (when a counter example can easily be found the computation time severely drops). The results are shown in Figure 5.

As can be seen in the figure, properties P1 and P2 scale up very well (linear), whereas properties P3 and P4 scale up in an exponential fashion. The fact that P3 and P4 do not scale up well has to do with the number of variables that are quantified in the properties, which is significantly smaller in properties P1 and P2 than it is for properties P3 and P4. But even for the maximum number of agents (90 in this case) the computation only takes several milliseconds, which still makes the approach useful for most auctions being investigated.

6.2 Reverse Sealed-Bid Combinatorial Auction with Time Windows

The second set of experiments have been conducted upon the reverse sealed-bid combinatorial auction with time windows. In this setting, there are multiple variations possible, namely vary the number of tasks, the number of bidders, and the average number of bids per bidder. In this case, we have decided to limit the number of variations to two elements, namely the number of tasks, and the number of bidders. Each agent submits one bid per trace. Hereby, one agent includes all the tasks in the bid (to make sure at least one combination of bids covers all tasks), and the other agents randomly select the tasks they bid upon (in this case they include a task in their bid with probability of 0.5).

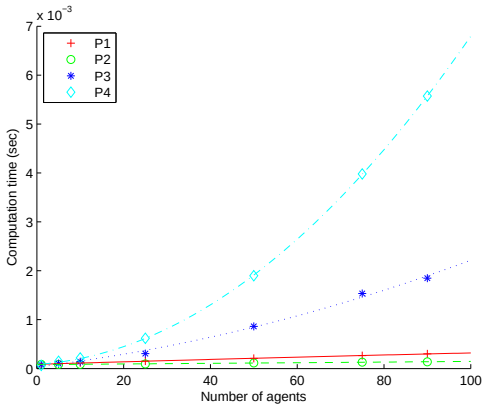

Fig. 5. Computation time needed to verify correctness of different properties for single item first price sealed bid auctions with varying numbers of agents. Results shown are averages of 50 traces for each setting.

Furthermore, the time windows included per task precisely comply to the constraints specifi ed in the RFQ. In Figure 6 and 7 we show the results for property P1 and P2. Hereby, the number of agents that bid have been varied between 1 and 50, whereas the number of tasks have been varied between 1 and 6. 50 runs have been performed for each setting.

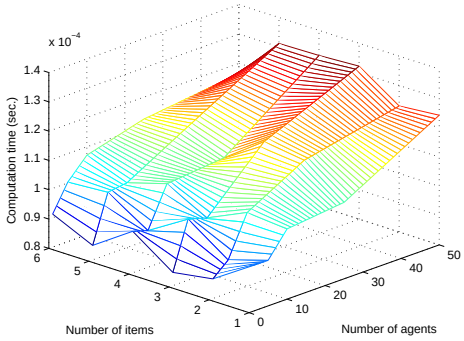

Fig. 6. Computation time needed for P1 in the RCATW

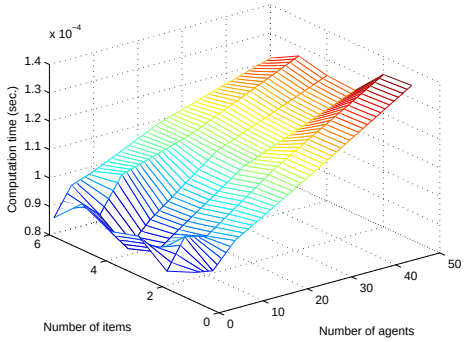

Fig. 7. Computation time needed for P2 in the RCATW

It can be seen in the fi gures that the same patterns are present as seen in Section 6.1. The computation time scales up in a linear fashion as the number of agents goes up. Furthermore, the number of tasks does not influence the overall computation time needed.

For properties P3 and P4 results are shown for smaller number of agents (1 to 5 bidders). Hereby, the number of tasks again does not influence the overall computation time needed, whereas an increase of the number of agents causes an exponential growth of the computation time. This is consistent with known complexity results for clearing combinatorial auctions that show there is no polynomial-time solution, nor even a polynomial-time bounded approximation [15]. These results can however be improved by for example performing pre-processing of the trace by splitting the trace up into multiple parts, and running the checks in parallel. This is part of future work.

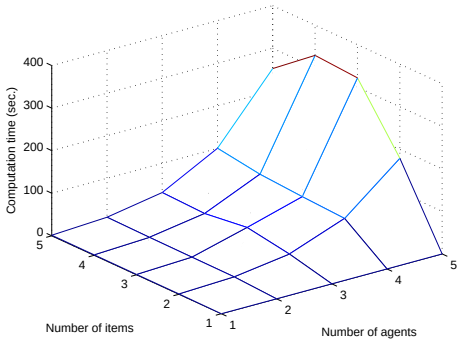

Fig. 8. Computation time needed for P3 in the RCATW

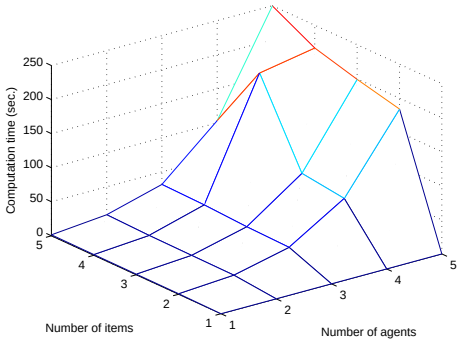

Fig. 9. Computation time needed for P4 in the RCATW

7 Related Work

Analysis of compliance of agents to certain desired properties has been studied extensively. Typically however, such properties are not studied by analysing empirical traces but by proving that given certain agent behaviors, some particular desired behaviors are guaranteed. There is a large body of work on model checking of multi-agent systems, see e.g. [10] and [3], where typically model checking is used for conformance testing, i.e. to verify that the implementation of an agent respects an abstract protocol definition for agent interactions. In [13] model checking is used to verify agent systems implemented using the logic-based AgentSpeak language. An auction system is presented as a case study to show how BDI auction specifications are satisfied and can be verified using a model checker. However, in open systems knowledge of the internals of the agents is generally not available, so model checking cannot be applied widely.

The specification of protocols using temporal logic has been addressed, e.g. in [9]. The verification of such properties upon auctioning traces has however not been addressed before. In [2] a framework is introduced for the specification of properties for open systems, as well as reasoning and verification of these properties. They use the contract net protocol as a running example. They take a normative systems view,

specifying social constraints, social roles, and social states. The approach focuses on verification at an abstract organizational level, not on specific empirical traces of agent behavior.

A tool called *SOCS-SI* aimed at verifying compliance of agent interactions is presented in [1]. In their approach a *history manager* composes an event history which is checked in the *social compliance verifier*. Some example properties related to auctions are presented, but the paper focuses on the verification approach more than on verification of auctions. This method, as ours, does not require access to the internals of the agents, but checks compliance only by examining the interaction protocols. In [4] properties are expressed for evaluating traces of human negotiation, also using TTL. The scope of the paper is however limited to multi-issue negotiation with a specific protocol, and not to specification of properties for auctions in general, which is the aim of this paper.

Current work to increase trust in auctions addresses almost exclusively the issue of verification of the identity of buyers and sellers (see, for instance, [11] for a study on trust and reputation on eBay). Alternatively, secure protocols are proposed (see, for instance, [12, 6]) to ensure that communications between the agents and the auctioneer are protected. There is an implicit assumption that the auction clearing houses act properly. We believe that with the proliferation of auction houses the need to verify the correctness of their operations will increase. This could be done using the approach we present in this paper.

8 Conclusions

We introduced an approach to analyze auction traces. We have adopted an expressive trace-based temporal logic (cf. [5]), which enables the specification of desired properties that include specific time points. Using this temporal logic, we presented ontologies that represent the specific interactions between the agents that participate in the auctions, properties of the auction protocols. All the properties are specified in a modular fashion, allowing re-usability. We have illustrated the analysis process by means of example traces. To automate the analysis, the properties have been implemented in the TTL Checker software (cf. [5]).

The approach presented analyzes traces of communications that occurred during auction sessions. The approach does not need any knowledge of the internals of the agents, and is therefore suitable for open environments and for mixed humans/agents auction systems. The assumption that execution traces are available is not unrealistic and could be added as a requirement to web based auction sites to verify rule compliance.

The scaling of the verification process itself is an important aspect. The fact that we do not use model checking techniques, but focus on traces of negotiation behavior, makes the approach more scalable compared to model checking techniques. The results have shown that for simple properties, the approach scales up in a linear fashion whereas for the more complex cases the approach scales up exponentially. For these complex traces however, it is possible to analyze subsets of such traces in parallel. Analyzing the improvements of such parallel checking is future work. General scalability of the

checking of properties against traces using TTL and the accompanying software tool has been described in [5]. Future work is to verify these properties on real auction data, to investigate whether the protocol was always followed.

References

1. M. Alberti, M. Gavanelli, E. Lamma, F. Chesani, P. Mello, and P. Torroni. Compliance verification of agent interaction: A logic-based software tool. *Applied Artificial Intelligence*, 20:133–157, 2006.
2. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *Proc. First Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1053–1062. ACM, 2002.
3. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking Agentspeak. In *Proc. Second Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 409–416. ACM, 2003.
4. T. Bosse, C. M. Jonker, and J. Treur. Experiments in human multi-issue negotiation: Analysis and support. In *Proc. Third Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 672–679. IEEE Computer Society, 2004.
5. T. Bosse, C. M. Jonker, L. van der Meij, A. Sharpanskykh, and J. Treur. Specification and verification of dynamics in cognitive agent models. In *Proc. Sixth Int'l Conf. on Intelligent Agent Technology (IAT 2002)*, pages 247–254. IEEE Computer Society, 2006.
6. Y. F. Chunga, K. H. Huanga, H. H. Leeb, F. Laia, and T. S. Chen. Bidder-anonymous english auction scheme with privacy and public verifiability. *Journal of Systems and Software*, 81(1):113–119, January 2008.
7. J. Collins, W. Ketter, and M. Gini. A multi-agent negotiation testbed for contracting tasks with temporal and precedence constraints. *Int'l Journal of Electronic Commerce*, 7(1):35–57, 2002.
8. P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. MIT Press, 2006.
9. M. Fisher and M. Wooldridge. Specifying and executing protocols for cooperative action. In *Proc. Int'l Working Conf. on Cooperating Knowledge-Based Systems*, 1994.
10. F. Guerin and J. Pitt. Guaranteeing properties for e-commerce systems. In *Agent-Mediated Electronic Commerce IV. Designing Mechanisms and Systems*, pages 397–413. Springer Verlag, 2002.
11. A. Hortacsu. Trust and reputation on ebay: Micro and macro perspectives. Technical report, Department of Economics, University of Chicago, 2005.
12. A. Jaiswal, Y. Kim, and M. Gini. Design and implementation of a secure multi-agent marketplace. *Electronic Commerce Research and Applications*, 3(4):355–368, 2004.
13. R. Podorozhny, S. Khurshid, D. Perry, and S. Zhang. Verification of cooperative multi-agent negotiation with Alloy. Technical Report TXSTATE-CS-TR-2006-4, Texas State University, San Marcos, TX, September 2006.
14. T. Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proc. of the Eleventh Nat'l Conf. on Artificial Intelligence*, pages 256–262, Washington, DC, 1993.
15. T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
16. W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.
17. M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.