Abstractions for Model-Checking Game-theoretic Properties of Auctions

(Short Paper)

E. M. Tadjouddine
University of Aberdeen
Department of Computing Science
AB24 3UE, Aberdeen, UK
e.tadjouddine@abdn.ac.uk

Frank Guerin
University of Aberdeen
Department of Computing Science
AB24 3UE, Aberdeen, UK
f.guerin@abdn.ac.uk

Wamberto Vasconcelos
University of Aberdeen
Department of Computing Science
AB24 3UE, Aberdeen, UK
wvasconcelos@acm.org

ABSTRACT

We are interested in verifying game-theoretic properties such as strategyproofness for auction protocols in open agent systems. Model checking provides an automatic way of carrying out such proofs. However it may suffer from state space explosion for large models. To improve the performance of model checking, abstractions were used along with the SPIN model checker. We applied the technique to the Vickrey auction. Numerical results showed the limits of relying solely on SPIN. To reduce the state space required by SPIN, two property-preserving abstraction methods were applied: firstly the classical program slicing technique, which removes irrelevant variables with respect to the property; the second replaces large data, possibly infinite values of variables, with smaller abstract values. This enabled us to model check the strategy-proofness property of the Vickrey auction for unbounded bid ranges and for any number of agents.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multi-agent systems, game theory*; F.3 [**Semantics of Programming Languages**]: General—*program analysis and verification*

General Terms

Algorithms, Economics, Verification

Keywords

Economic paradigms: electronic markets and institutions, game theory (cooperative and non-cooperative). Agent societies and Societal issues: trust and reputation.

1. INTRODUCTION

Trust is a major concern in agent-mediated eCommerce systems. To tackle this, much research has been carried out to develop game theory mechanisms which guarantee desirable properties for the system, even in the face of agents who are willing to lie or cheat; for example, there are mechanisms which can guarantee that a system is robust to agents bidding falsely or colluding. These mechanisms work perfectly well in a closed Multi-Agent System (MAS) when designers can program agents in full knowledge of the favourable properties of the mechanism. However, it is not clear how such mechanisms could be used in open systems where agents might have to interoperate between different institutions. A roaming agent arriving at an institution where a new, previously unseen, protocol is in use, will need to understand the rules of engagement in much the same way human agents can. This is a major stumbling block for agent-mediated eCommerce, which does indeed envisage a future with open systems of roaming agents; it has been recognised as a "major challenge facing computer scientists" [2].

In this paper we assume that there is some standard language in which the rules of the auction can be written and published. A roaming agent who arrives at a foreign institution can download a protocol and analyse it in order to make a decision about whether or not to participate, and what strategy to use. The challenge now is for the roaming agent, with bounded computing resources, to be able to automatically check some of the game-theoretic properties of the protocol. We focus on dominant strategy equilibrium, which means the game has the property of *strategy-proofness*; this gives agents an incentive to bid their true valuations. Game theoretic properties such as strategy-proofness rely on very strong assumptions; it is required that the property be common knowledge among the players. If the common knowledge of the equilibrium is not achieved, then agents cannot expect it to be played [3].

Model checking provides an automatic way of carrying out the verification of game-theoretic properties of a given auction mechanism. However it may suffer from state space explosion for large models. We considered the case of the Vickrey auction and checked its strategy-proofness using the SPIN model checker [4]. Numerical results showed the limits of relying solely on SPIN. To reduce the state space required by SPIN, two property-preserving abstraction methods were applied: the first is the classical program slicing technique [7], which removes irrelevant variables with respect to the property; the second replaces large data, possibly infinite values of variables with smaller abstract values. This enabled us to model check the strategy-proofness property of the Vickrey auction for unbounded bid ranges and for any number of agents.

Cite as: Abstractions for Model-Checking Game-theoretic Properties of Auctions (Short Paper), Tadjouddine, E. M., Guerin, F. and Vasconcelos, W., *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons(eds.),May,12-16.,2008,Estoril,Portugal,pp.1613-1616.

2. VERIFYING BY MODEL CHECKING

We have considered a simple Vickrey auction where n agents bid for a single item. Each agent has a private valuation v of the item. The highest bidder wins the item but pays the second highest bid p, getting the utility $u = v - p$. A losing bidder pays nothing and has a zero utility. We formally specified the auction using the Promela process modeling language [4]. We then verified some game-theoretic properties using the SPIN model checker [4]. These properties are expressed as Promela assertions in a model parameterized by the range of the bids (set A of actions) and the number of agents n. For a given agent i, we evaluate its utility u_i^* when bidding its true valuation and its utility u_i otherwise, and check the assertion $u_i^* \geq u_i$ in all the possible game configurations specified by the property. For the strategy-proofness property the number of possible configurations represents all possible strategy profiles, which is exponential in the number of agents, and hence computationally expensive.

Table 1 shows runtime statistics for the checking of the strategy-proofness property. We fixed the bid range and varied the number of agents. We observe that beyond a certain number of agents, the amount of the memory required by SPIN explodes, provoking a 'ran out of memory' error. However, this mechanism is simple enough to carry out a full space checking for up to 600 agents on the machine we have used. These results were obtained by compiling and running the models produced by SPIN on a PC Pentium Dual Processor 2.99 GHz with 2GB RAM, running Windows XP, using options `-DBITSTATE -DVETORSZ=`m where m is the size of the state vector chosen according to the size of the model at hand. For efficiency the states are stored in a hashtable; this turns an exponential search into a linear one and enhances the performance of the checking procedure.

	Number of players				
	100	300	500	600	700
Memory(Mb)	146.72	795.92	904.20	1083.30	–
CPU Time(s)	1.20	6.44	7.20	8.29	–

Table 1: Statistics for the strategy-proofness property in a Vickrey Auction (bids are from 0 to 1000).

3. ABSTRACT MODEL CHECKING

We used two property-preserving abstraction methods to reduce the costs involved in checking: firstly *program slicing*, used to remove portions of code in program analysis which are not relevant to a given criterion [7], and secondly *abstract interpretation* [1].

(1) Removal of Irrelevant Constructs: This is the *program slicing* technique used to remove portions of code in program analysis which are not relevant to a given criterion [7]. A typical criterion is a line of the program – the slice contains those commands which affect the variables in that line. Another criterion is a set of variables – the slice contains those commands affecting these variables. We identify *dependency relationships* among variables: $v_1 \prec v_2$, for variables v_1, v_2, holds if the computation of v_1 depends on the value of v_2. To illustrate this abstraction, let us consider the Vickrey auction example for two agents shown on the left-hand side of Figure 1. Let us suppose we want to check the assertion `u1t` $\geq$ `u1` where `u1t` and `u1` are, respectively, the utilities of agent 1 when it bids its valuation and any other number. The variable dependencies of the auction are `x1` $\prec$ `x2`, `x2` $\prec$ `x1`, `u1` $\prec$ `v1`, `u1` $\prec$ `x2`, `u2` $\prec$ `v2`, `u2` $\prec$ `x1`; they describe the flow of data among the variables and (in the case of the `if` test) how variables depend on one another to define the flow of execution. We show on the right-hand side of Figure 1 a slice of the Vickrey auction, in which all commands referring to `u2` have been removed.

```
if
  ::  (x1 >= x2) ->
    u1 = v1 - x2;
    u2 = 0;
  ::  else ->
    u2 = v2 - x1;
    u1 = 0;
fi;
```

```
if
  ::  (x1 >= x2) ->
    u1 = v1 - x2;

  ::  else ->

    u1 = 0;
fi;
```

Figure 1: Vickrey Auction (left) and its Slice (right)

(2) Redefining Strategy Space via Abstract Values: An abstraction provides a mapping of the original (concrete) domain (and associated search space) onto a less complex (abstract) domain, enabling us to eliminate irrelevant details. We perform the checking using the abstract model, and later decide if the property still holds in the concrete model. Since we represent games as computer programs, our search space is the *state domain*, that is, the execution state of the program containing the values of all its variables and the current point of the execution flow.

DEFINITION 3.1. *A finite game is a transition system* $\Sigma = \langle N, A, S, \theta, \rho, \eta \rangle$ *where* N, A, S *are non-empty sets of agents, strategies and states respectively;*

- $\theta : S \rightarrow$ *Boolean is true for at least one element of* S *(called initial state);*
- $\rho : S \times S \rightarrow$ *Boolean is a transition relation, and*
- $\eta : S \rightarrow$ *Boolean is true for at least one element of* S *(called final state)*

For each final state we associate a utility to each agent. A reachable state of Σ *is a state that can be reached following a finite sequence of transitions from an initial state. A reachable transition is a transition from a reachable state.*

We introduce the abstract version of the previous concept:

DEFINITION 3.2. *An abstract finite game* $\widehat{\Sigma} = \langle \widehat{N}, \widehat{A}, \widehat{S}, \widehat{\theta}, \widehat{\rho}, \widehat{\eta} \rangle$ *is an abstraction of* $\Sigma = \langle N, A, S, \theta, \rho, \eta \rangle$ *if there exists a mapping* $\alpha : S \rightarrow \widehat{S}$ *such that*

- $\forall s \in S,\ \theta(s) \rightarrow \widehat{\theta}(\alpha(s))$
- $\forall s, s',\ \rho(s, s') \rightarrow \widehat{\rho}(\alpha(s), \alpha(s'))$

The mapping α is called an *abstraction map*. Its inverse γ, associating an abstract state $\widehat{s} \in \widehat{S}$ and transition $\widehat{\rho}$ to its corresponding concrete state $s \in S$ and transition ρ is called a *concretization map*.

Abstraction maps usually rely on over-approximations to produce, for every point of the program, an abstract state $\widehat{s}$ such that $\gamma(\widehat{s})$ contains all the concrete reachable states at that location. Traditionally, these approximations are defined over lattices, defined as:

DEFINITION 3.3. *A lattice* $(L, \sqcup, \sqcap, \bot, \sqsubseteq)$ *is a complete partial order on set* L *by* $\sqsubseteq$ *in which any two elements* $x, y \in L$ *have a greatest lower bound* $(x \sqcap y) \in L$ *and a least upper bound* $(x \sqcup y) \in L$.

A lattice is *complete* if any two elements $x, y \in L$ have a greatest element ($x \sqcup y$) and a least element ($x \sqcap y$). An example of complete lattice is the power set domain with the usual set operators.

The game-theoretic properties we are interested in are first-order logic formulae (denoted as φ) that can be expressed in Σ and their abstract counterparts $\widehat{\varphi}$ in $\widehat{\Sigma}$. It is important to ensure that whenever a property φ is violated in the concrete domain Σ, its abstraction $\widehat{\varphi}$ is also violated in the abstract domain $\widehat{\Sigma}$.

DEFINITION 3.4. *An abstraction $\alpha : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ is sound if whenever $\widehat{\varphi}$ holds in $\widehat{\Sigma}$, then φ holds in Σ. An abstraction $\alpha : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ is complete if whenever φ holds in Σ, then $\widehat{\varphi}$ holds in $\widehat{\Sigma}$.*

3.1 Building Abstractions for Auctions

Finding an abstraction map is not an easy task and depends on the property to be checked. In our work, abstraction is a way of minimising the explosion on the number of states of the concrete model as illustrated in Section 2.

The principal cause of the explosion in the number of states observed in Section 2 is the exponential input data required by the strategy-proofness property. This input data describes all the strategy profiles for n players the auction. We reduce the possible strategy profiles by not considering every possible bid of an agent, but instead considering the significant ranges of bids. Given the valuation v_i of an agent i for a given single item, we can distinguish the following three strategies its opponents may adopt: (i) bid higher than v_i; (ii) bid exactly v_i; (iii) bid lower than v_i.

We define the following types for agents competing with agent i, corresponding to the strategies above:

$$\begin{aligned} T_h &= \{x \in \mathbb{R} \mid x > v_i \geq 0\} \\ T_e &= \{x \in \mathbb{R} \mid x = v_i\} \\ T_l &= \{x \in \mathbb{R} \mid 0 \leq x < v_i\} \end{aligned}$$

All the configurations of the game involving agent i with respect to its opponents can be described by agent i's bid against the bids of the typed agents in the set $T = \{T_h, T_e, T_l\}$. Consider the following three mappings with signature $A^{n-1} \rightarrow T$ and projecting each component of a vector $x_{-i} \in A^{n-1}$ to a type $t \in T$ as follows:

$$\begin{aligned} \text{proj}_h(x_{-i}) &= \{x_j \in T_h \mid j \neq i\} \\ \text{proj}_e(x_{-i}) &= \{x_j \in T_e \mid j \neq i\} \\ \text{proj}_l(x_{-i}) &= \{x_j \in T_l \mid j \neq i\} \end{aligned}$$

The mapping proj_h projects all components of the vector x_{-i} that are greater than v_i to the data type T_h. Similarly proj_e and proj_l are projections on T_e and T_l respectively. Let us consider $f : A^{n-1} \rightarrow 2^T$ mapping an element $x_{-i} \in A^{n-1}$ to an element $\widehat{x_{-i}} = f(x_{-i})$ of the powerset 2^T as follows:

$$\widehat{x_{-i}} = \begin{cases} T_h & \text{if } \text{proj}_e(x_{-i}) = \text{proj}_l(x_{-i}) = \emptyset \\ T_e & \text{if } \text{proj}_h(x_{-i}) = \text{proj}_l(x_{-i}) = \emptyset \\ T_l & \text{if } \text{proj}_e(x_{-i}) = \text{proj}_h(x_{-i}) = \emptyset \\ T_h \vee T_e & \text{if } \text{proj}_l(x_{-i}) = \emptyset \\ T_h \vee T_l & \text{if } \text{proj}_e(x_{-i}) = \emptyset \\ T_l \vee T_e & \text{if } \text{proj}_h(x_{-i}) = \emptyset \\ T_h \vee T_e \vee T_l & \text{otherwise} \end{cases}$$

By construction, f maps every vector of A^{n-1} to its equivalent type in the complete lattice $L = (2^T, \vee, \wedge, \emptyset, \sqsubseteq)$. The mapping f induces an equivalence relation whose equivalence classes represent state variables; these state variables correspond to elements of the powerset 2^T.

Let x_h, x_e, x_l be the equivalent classes associated to the types T_h, T_e, T_l respectively. x_h, x_e, x_l are abstract variables that will be used in the transformed (abstract) program. Concrete arithmetic operations, e.g., `+`, `-`, `*`, `<`, and `>`, must also be transformed so as to manipulate the abstract variables x_h, x_e, x_l. Moreover, the variables x_h, x_e, x_l cover real values that are greater than or equal to zero. However, the arithmetic operation "`-`" forces us to consider negative values as well, which we denote by x_n. We can therefore partition the set $\mathbb{R}$ into the subsets represented by the equivalent classes x_n, x_l, x_e, and x_h. The abstract variables x_n, x_l, x_e, and x_h represent the real-valued intervals $(-\infty, 0), [0, v_i), [v_i, v_i]$, and (v_i, ∞) respectively.

$-$	x_n	x_l	x_e	x_h
x_n	x_n, x_l, x_e, x_h	x_n	x_n	x_n
x_l	x_l, x_e, x_h	x_n, x_l	x_n	x_n
x_e	x_h	x_l	x_l	x_n
x_h	x_h	x_l, x_e, x_h	x_l, x_e, x_h	x_n, x_l, x_e, x_h

Table 2: Signature of Abstract Subtract $-_{\text{abs}}$

Table 2 shows the signature of the abstract operation $-_{\text{abs}}$ (the abstract counterpart of subtraction). The first column of the table shows the values of the first parameter of the operation $-_{\text{abs}}$; the top row contains the values of the second parameter; the various outcomes of the operation are the table cells. If the result of the abstract operation belongs to a set of equivalence classes (as opposed to a single equivalence class) then this indicates a lack of knowledge about the abstract variables since they over-approximate concrete values in the original program. This inaccuracy is modelled by the model checker SPIN as a non-deterministic choice over the set of values in the set.

It follows that the mapping f enables us to transform concrete data and operations from the original program into corresponding abstract data and operations in the transformed program. The states and transitions in the abstract program are defined to be those induced by the states and transitions in the abstract program. Consequently, we have built up an abstraction map $\alpha : \Sigma \rightarrow \widehat{\Sigma}$ from the concrete domain into the abstract domain. An important issue is whether the abstraction α is sound for the game-theoretic property to be checked. For that purpose, we need to express the property in the obtained abstract domain.

3.2 Abstracting Properties

In the abstract model, the valuations v_i, bids $b_i \in A_i$ $b_{-i} \in A_{-i}$, payments p_i and utilities u_i of the agents become abstract variables $\widehat{v_i}$, $\widehat{b_i} \in \widehat{A_i}$, $\widehat{b_{-i}} \in \widehat{A_{-i}}$, $\widehat{p_i}$ and $\widehat{u_i}$ respectively. All original operations are also transformed into abstract operations manipulating the defined abstract types. The strategy-proofness property becomes $\widehat{\varphi}$ as follows:

$$\forall i,\ \forall \widehat{v_i},\ \forall \widehat{b_i},\ \forall \widehat{b_{-i}},\ \widehat{u_i}(\widehat{v_i}, \widehat{v_i}, \widehat{b_{-i}}) \geq_{\text{abs}} \widehat{u_i}(\widehat{v_i}, \widehat{b_i}, \widehat{b_{-i}}). \quad (1)$$

Using our abstraction α, we have $\widehat{A_i} = \{x_h, x_e, x_l\}$, which is a partition of A_i and $\widehat{A_{-i}}$ is isomorphic to 2^T (which we denote by $\widehat{A_{-i}} \equiv 2^T$). We now need to prove that our abstraction α is sound. This is established by the following:

LEMMA 3.5. *The abstraction map* $\alpha : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ *is sound.*

Proof: We need to prove that if the strategy-proofness property $\widehat{\varphi}$ holds in $\widehat{\Sigma}$, then its equivalent version φ holds in Σ. We do so by proving that the behaviours in the abstract program over-approximate the behaviours in the concrete program. From the viewpoint of agent i, strategy-proofness in the abstract domain means:

- If i has opponents of one type T_h, T_e, or T_l, then the inequality (1) must be true for all $\widehat{b}_i \in \widehat{A}_i$ and $\widehat{b_{-i}}$ taking the values representing the equivalence classes x_h, x_e, or x_l of the abstraction α respectively.
- If i has opponents of two types $\{T_h, T_e\}, \{T_h, T_l\}$, or $\{T_e, T_l\}$, then the inequality (1) must be true for all $\widehat{b}_i \in \widehat{A}_i$ and $\widehat{b_{-i}}$ taking the values x_h and x_e, x_h and x_l, or x_e and x_l of the abstraction α respectively.
- If i has opponents of three types $\{T_h, T_e, T_l\}$, then the inequality (1) must be true for all $\widehat{b}_i \in \widehat{A}_i$ and $\widehat{b_{-i}}$ taking the values x_h, x_e, x_l.

By construction, the inverse f^{-1} of the mapping f associates each element of $\widehat{A_{-i}} \equiv 2^T$ to a subset of A_{-i} and clearly

$$\cup_{a \in \widehat{A_{-i}}} f^{-1}(a) = A_{-i}.$$

Furthermore, $\widehat{A}_i$ is a partition of A_i. It follows that the abstraction map α is sound.

□

3.3 An Abstract Model-Checking Algorithm

To check the strategy-proofness property for a given player amounts to checking bidding x_e gives the maximum utility in all the following settings: (i) its opponents of single type can bid a single value x_h, x_e, or x_l; (ii) its opponents of two types can bid the tuples (x_h, x_e), (x_h, x_l), or (x_e, x_l); (iii) its opponents of three types can bid the tuple (x_h, x_e, x_l).

This reduces the strategy space from the size $|A|^{nm}$ initially to $(3 \times 7)^m$, three for agent i and seven for its opponents. If the number of items $m = 1$ as, for example, in the Vickrey auction, this is easy to check. Notice that our abstraction is only sound – this means that if the property is true in the abstract domain, then, it is true is the concrete domain. If, however, the property does not hold in the abstract model, then SPIN will generate a counter-example. The generated counter-example may be due to spurious behaviour caused by approximations in the abstract model or it may be genuine. Techniques have been developed to cope with such scenarios, see for example [5, 6].

We have implemented this algorithm for checking strategy-proofness in the Vickrey auction. For this simple single item auction, we have designed and implemented the abstract variables and related operations, thus building up an abstract program modelling the auction. Then, we checked the abstracted strategy-proofness property using our algorithm. In the results shown in Table 3, AMCA stands for the abstract model checking algorithm hereby outlined and Slicing stands for the application of the program slicing optimisation. These results show that for the Vickrey auction, the number of players and the bid range cease to be a factor of state space explosion and that strategy-proofness can be checked using a small amount of computer resources. Moreover, the program slicing technique improved slightly the checking as expected in this case.

	AMCA	AMCA & Slicing
Memory (Mb)	3.65	3.02
CPU Time (s)	0.31	0.25

Table 3: Statistics for the strategy-proofness property in a Vickrey Auction with an unbounded number of players using the proposed two abstractions

4. CONCLUSION

We have considered auction mechanisms expressed in a formal language, and automatically checked desirable properties such as strategy-proofness. We have presented numerical results showing the computational limits of using a plain (exhaustive) model checking approach. These limits are due to the state space explosion problem. To enhance this approach, we have combined model checking and abstract interpretation. We have proposed two property-preserving abstractions. The first is the classical program slicing technique; the second is novel and tailored to the problem of verifying game equilibria. This allowed us to verify the Vickrey auction, regardless of the number of bidders and their bid range (which was not feasible by exhaustive model checking). Note that although the abstraction requires some creativity from the human designer, once the appropriate abstraction is found, it can be published along with the mechanism, to facilitate automatic checking by agents. In summary, we have shown that computationally verifiable mechanisms are feasible in principle. Such mechanisms are useful for agent scenarios wherein trust in the system must be guaranteed and entry-deterrence must be tackled in order to attract more participants.

5. REFERENCES

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *The Fourth Annual ACM SIGPLAN-SIGACT Symposium on POPL*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[2] R. K. Dash, N. R. Jennings, and D. C. Parkes. Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, pages 40–47, Nov. 2003.

[3] F. Guerin and E. M. Tadjouddine. Realising common knowledge assumptions in agent auctions. In *The IEEE/WIC/ACM Int'l Conf. on Intelligent Agent Technology*, pages 579–586, Hong Kong, China, 2006.

[4] G. J. Holzmann. *The SPIN Model checker: Primer and Reference Manual*. Addison, Boston, USA, 2004.

[5] C. S. Pasareanu, M. B. Dwyer, and W. Visser. Finding feasible abstract counter-examples. *Soft. Tools for Tech. Transfer*, 5(1):34–48, 2003.

[6] H. Saïdi. Model checking guided abstraction and analysis. In J. Palsberg, editor, *SAS*, volume 1824 of *LNCS*, pages 377–396. Springer, 2000.

[7] F. Tip. A Survey of Program Slicing Techniques. *Journal of Progr. Lang.*, 3(3):121–189, Sept. 1995.