A Qualitative Comparison of the Suitability of Four Theorem Provers for Basic Auction Theory*

Christoph Lange[1], Marco B. Caminati[2], Manfred Kerber[1], Till Mossakowski[3], Colin Rowat[4], Makarius Wenzel[5], and Wolfgang Windsteiger[6]

[1] Computer Science, University of Birmingham, UK
[2] http://caminati.net.tf, Italy
[3] University of Bremen and DFKI GmbH Bremen, Germany
[4] Economics, University of Birmingham, UK
[5] Univ. Paris-Sud, Laboratoire LRI, UMR8623, Orsay, F-91405, France
[6] RISC, Johannes Kepler University Linz (JKU), Austria
http://www.cs.bham.ac.uk/research/projects/formare/code/auction-theory/

Abstract Novel auction schemes are constantly being designed. Their design has significant consequences for the allocation of goods and the revenues generated. But how to tell whether a new design has the desired properties, such as efficiency, i.e. allocating goods to those bidders who value them most? We say: by formal, machine-checked proofs. We investigated the suitability of the Isabelle, Theorema, Mizar, and Hets/CASL/TPTP theorem provers for reproducing a key result of auction theory: Vickrey's 1961 theorem on the properties of second-price auctions. Based on our formalisation experience, taking an auction designer's perspective, we give recommendations on what system to use for formalising auctions, and outline further steps towards a complete auction theory toolbox.

1 Motivation: Why Formalise Auction Theory?

Auctions are a widely used mechanism for allocating goods and services[7], perhaps second in importance only to markets. They are used to allocate electromagnetic spectrum, airplane landing slots, oil fields, bankrupt firms, works of art, eBay items, and to establish exchange rates, treasury bill yields, and stock exchange opening prices. Novel auction schemes are constantly being designed, aiming to maximise the auctioneer's revenue, foster competition in subsequent markets, and to efficiently allocate resources.

Auction design can have significant consequences. Klemperer attributed the low revenues gained in some government auctions of the 3G radio spectrum in

* This work has been supported by EPSRC grant EP/J007498/1. We would like to thank Peter Cramton and Elizabeth Baldwin for sharing their auction designer's point, and Christian Maeder for his recent improvements to Hets.

[7] For the US, the National Auctioneers Association reported $268.5 billion for 2008 [1].

2000 (€20 per capita vs. €600 in other countries) to bad design [2]. Design practice outstrips theory, especially for complex modern auctions such as combinatorial ones, which accept bids on subsets of items (e.g. collections of spectrum). Designing a revenue-maximising auction is *NP*-complete [3] even with a single bidder. Important auctions often run 'in the wild' with few formal results [4]. We aim at convincing auction designers that investing into formalisation pays off with machine-checked proofs and a deeper understanding of the theory. To this end, we want to provide them with a toolbox of basic auction theory formalisations, on top of which they can formalise and verify their own auction designs – which typically combine standard building blocks, e.g. an ascending auction converting to a sealed-bid auction when the number of remaining bidders equals the number of items available. Given the ubiquity of specialist support across a range of service sectors, we conjecture that auction designers might be supported by formalisation experts, creating a niche for specially trained experts at the interface of the core mechanised reasoning community and auction designers.

Our ForMaRE project (formal mathematical reasoning in economics [5]) seeks to increase confidence in economics' theoretical results, to aid in discovering new results, and to foster interest in formal methods within economics. To formal methods, we seek to contribute new challenge problems and user experience feedback from new audiences. Auctions are representative of practically relevant fields of economics that have hardly been formalised so far.[8] Economics has been formalised before [7], particularly social choice theory (cf. §5 and [8]) and game theory (cf. [9] and our own work [10]). However, none of these formalisations involved economists. Formalising (mathematical) theories and applying mechanised reasoning tools remain novel to economics.[9]

§2 establishes requirements for the Auction Theory Toolbox (ATT); §3 explains our approach to building it. §4 is our main contribution: a qualitative comparison of how well four different theorem provers satisfy our requirements. §5 reviews related work, and §6 concludes and provides an outlook.

2 Requirements for an Auction Theory Toolbox

Conversations with auction designers established ATT requirements as follows:

D1 Formalise ready-to-use basic auction concepts, including their definitions and essential properties.

D2 Allow for extension and application to custom-designed auctions without requiring expert knowledge of the underlying mechanised reasoning system.

From a computer scientist's technical perspective, these translate to:

[8] Even code verification is typically not considered, although Leese, who worked on the UK's spectrum auctions, has called for auction software to be added to the Verified Software Repository at `http://vsr.sourceforge.net` [6].

[9] There is a field 'computational economics'; however, it is mainly concerned with the *numerical* computation of solutions or simulations (cf., e.g., [11]).

C1 Identify the right language to formalise auction theory. This language should (a) be sufficiently expressive for concisely capturing complex concepts, while supporting efficient proofs for the majority of problems, (b) be learnable for economists used to mathematical textbook notation, and (c) provide libraries of the mathematical foundations underlying auctions.

C2 Identify a mechanised reasoning system (a) that assists with cost-effective development of formalisations, (b) that facilitates reuse of formalisations already existing in the toolbox, (c) that creates comprehensible output to help users understand, e.g., why a proof attempt failed, or what knowledge was used in proving a goal, and (d) whose community is supportive towards users with little specific technical and theoretical background.

Note the conflicts of interest: a single language might not meet requirement C1a, and if it did, it might not be supported by a user-friendly system.

3 Approach to Building the Auction Theory Toolbox

To avoid a chicken-and-egg problem, we identify relevant domain problems in parallel to identifying languages and systems suitable for formalisation.

3.1 The Domain Problem: Vickrey's Theorem and Beyond

We started with Vickrey's 1961 theorem on the properties of second-price auctions of a single, indivisible good, whose bidders' private values are not publicly known. Each participant submits a sealed bid; one of the highest bidders wins, and pays the highest *remaining* bid; the losers pay nothing. Vickrey proved that 'truth-telling' – submitting a bid equal to one's actual valuation of the good – was a *weakly dominant* strategy, i.e. that no bidder can do strictly better by bidding above or below its valuation *whatever* the other bidders do. Thus, the auction is also *efficient*, allocating the item to the bidder with the highest valuation. Bidders only have to know their own valuations; in particular they need no information about others' valuations or the distributions these are drawn from.

As variants of Vickrey auctions are widely used (e.g. by eBay, Google and Yahoo! [12]), this formalisation will enable us to prove properties of contemporary auctions as well. The underlying theory is straightforward to understand even for non-economists and can be formalised with reasonable effort. Finally, formalising Vickrey provides a good introduction for domain experts to mechanised reasoning technology by serving as a small, self-contained showcase of a widely known result, helping to build trust in this new technology.

Maskin collected 13 theorems, including Vickrey's, in a review [13] of an influential auction theory textbook [14]. This sets the roadmap for building the ATT – a collaborative effort, to which we welcome community contributions [15].

3.2 Paper Elaboration to Prepare the Machine Formalisation

To prepare the machine formalisation, we refined the original paper source, aware that current mechanised reasoning systems typically require much more explicit

statements than commonly found on paper: automated provers must find proofs without running out of search space, whereas proof checkers require proofs at a certain level of detail, which in turn requires detailed statements. Maskin states Vickrey's theorem in two sentences and proves it in another six sentences [13, Proposition 1].[10] Our elaboration uses eight definitions specific to the domain problem plus an auxiliary one about maximum components of vectors, as follows:

$N = \{1, \ldots, n\}$ is a set of *participants*, often indexed by i. An *allocation* is a vector $\boldsymbol{x} \in \{0,1\}^n$ where $x_i = 1$ denotes participant i's award of the indivisible good to be auctioned (i.e. 'i wins'), and $x_j = 0$ otherwise. An *outcome* $(\boldsymbol{x}, \boldsymbol{p})$ specifies an allocation and a vector of payments, $\boldsymbol{p} \in \mathbb{R}^n$, made by each participant i. Participant i's *payoff* is $u_i \equiv v_i x_i - p_i$, where $v_i \in \mathbb{R}_+$ is i's valuation of the good. A *strategy profile* is a vector $\boldsymbol{b} \in \mathbb{R}^n$, where $b_i \geq 0$ is called i's *bid*.[11] For an n-vector $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$, let $\overline{y} \equiv \max_{j \in N} y_j$ and $\overline{y}_{-i} \equiv \max_{j \in N \setminus \{i\}} y_j$.

Definition 1 (Second-Price Auction). *Given* $M \equiv \left\{i \in N : b_i = \overline{b}\right\}$, *a* second-price auction *is an outcome* $(\boldsymbol{x}, \boldsymbol{p})$ *satisfying:*

1. $\forall j \in N \backslash M, x_j = p_j = 0$*; and*
2. *for one*[12] $i \in M$, $x_i = 1$ *and* $p_i = \overline{b}_{-i}$*, while,* $\forall j \in M \backslash \{i\}, x_j = p_j = 0$.

Definition 2 (Efficiency). *An efficient auction maximises* $\sum_{i \in N} v_i x_i$ *for a given* v*, i.e., for a single good,* $x_i = 1 \Rightarrow v_i = \overline{v}$.

Definition 3 (Weakly Dominant Strategy). *Given some auction, a strategy profile* $\boldsymbol{b}$ *supports an* equilibrium in weakly dominant strategies *if, for each* $i \in N$ *and any* $\hat{\boldsymbol{b}} \in \mathbb{R}^n$ *with* $\hat{b}_i \neq b_i$, $u_i\left(\hat{b}_1, \ldots, \hat{b}_{i-1}, b_i, \hat{b}_{i+1}, \ldots, \hat{b}_n\right) \geq u_i\left(\hat{\boldsymbol{b}}\right)$.[13] *I.e., whatever others do,* i *will not be better off by deviating from the original bid* b_i.

Theorem 1 (Vickrey 1961; Milgrom 2.1). *In a second-price auction, the strategy profile* $\boldsymbol{b} = \boldsymbol{v}$ *supports an equilibrium in weakly dominant strategies. Furthermore, the auction is efficient.*

The attempt to be close to a paper formalisation may introduce artefacts that unnecessarily complicate machine formalisation. E.g., the contiguous numeric participant indexing is merely a convention: formally any relation between participants' valuation, bid, allocation, and payment vectors suffices. Similarly, the product $v_i x_i$ recalls the general divisible good case ($x_i \in [0,1]$) and works around the lack of an easy and compact 'if–then–else' textbook notation.[14]

[10] The high level of Maskin's text is owed to its summative nature. Original proofs in auction theory are typically more thorough.

[11] This simplification is sufficient for proving the theorem. More precisely, all participants know that each v_i is an independent realisation of a random variable with distribution density f. A participant's *strategy* is a mapping g_i such that $b_i = g_i(v_i, f)$.

[12] When running an auction in practice, this i may be selected randomly, but this circumstance does not matter for the proof of Vickrey's theorem.

[13] The notation $u_i(\boldsymbol{b})$ is standard in economics but formally misleading. A more careful notation is $u_i(x_i, v_i, p_i)$, where x_i and p_i depend on $\boldsymbol{b}$ and the auction type.

[14] Case distinctions with curly braces consume at least two lines.

Proof. Suppose participant i bids $b_i = v_i$, whatever $\hat{b}_j$ the others bid. Let $\hat{\boldsymbol{b}}^{i \leftarrow v}$ abbreviate the overall vector $(\hat{b}_1, \ldots, \hat{b}_{i-1}, v_i, \hat{b}_{i+1}, \ldots, \hat{b}_n)$. There are two cases[15]:

1. i wins. This implies $b_i = v_i = \overline{\hat{\boldsymbol{b}}^{i \leftarrow v}}$, $p_i = \overline{\hat{\boldsymbol{b}}^{i \leftarrow v}}_{-i}$, and $u_i(\hat{\boldsymbol{b}}^{i \leftarrow v}) = v_i - p_i = \hat{\boldsymbol{b}}_i^{i \leftarrow v} - \overline{\hat{\boldsymbol{b}}^{i \leftarrow v}}_{-i} \geq 0$. Now consider i submitting an arbitrary bid $\hat{b}_i \neq b_i$, i.e. assume an overall bid vector $\hat{\boldsymbol{b}}$. This has two sub-cases:
   (a) i wins with the other bid, i.e. $u_i(\hat{\boldsymbol{b}}) = u_i(\hat{\boldsymbol{b}}^{i \leftarrow v})$, as the second highest bid has not changed.
   (b) i loses with the other bid, i.e. $u_i(\hat{\boldsymbol{b}}) = 0 \leq u_i(\hat{\boldsymbol{b}}^{i \leftarrow v})$.
2. i loses. This implies $p_i = 0$, $u_i(\hat{\boldsymbol{b}}^{i \leftarrow v}) = 0$, and $b_i \leq \overline{\hat{\boldsymbol{b}}^{i \leftarrow v}}_{-i}$; otherwise i would have won. This yields again two cases for i's alternative bid :
   (a) i wins, i.e. $u_i(\hat{\boldsymbol{b}}) = v_i - \overline{\hat{\boldsymbol{b}}}_{-i} = b_i - \overline{\hat{\boldsymbol{b}}^{i \leftarrow v}}_{-i} \leq 0 = u_i(\hat{\boldsymbol{b}}^{i \leftarrow v})$.
   (b) i loses, i.e. $u_i(\hat{\boldsymbol{b}}) = 0 = u_i(\hat{\boldsymbol{b}}^{i \leftarrow v})$.

By analogy for all i, $\boldsymbol{b} = \boldsymbol{v}$ supports an equilibrium in weakly dominant strategies. Efficiency is immediate: the highest bidder has the highest valuation. □

3.3 Choosing Language and System

In terms of *logic*, it is not immediately obvious whether Vickrey's theorem is inherently higher-order. Defining the maximum operator on arbitrarily sized finite sets of real-valued bids and proving its essential properties requires induction and thus exceeds first-order logic (FOL): similarly for the finiteness of a set[16] and a formalisation of real numbers.[17] However, if one takes real vectors and a maximum operation on them for granted, and explicitly requires the maximum to exist, FOL suffices to formalise the relevant domain concepts: single good auctions, second-price auctions, and the theorem statement.[18]

In terms of *syntax*, we assume that auction designers will prefer a language that is close to the textbook mathematics they are used to, rather than having a programming language flavour. We assume that at least optional type annotations support intuitive modelling of domain concepts (e.g. an auction as a function that takes bids and returns an allocation and payments) and prevent formalisation mistakes by cheap early checks (cf. [16]).

In terms of *user experience*, we study two paradigms: *automated provers* try, given a theorem and a knowledge base, to automatically find a proof, potentially appealing to our audience if the user just has to push a button (as with model checkers). *Interactive provers* interactively check a proof written by the user, which may be convenient when a paper proof already exists.

[15] Our initial elaboration of Maskin's proof, which distinguishes cases on the basis of participants' bids, resulted in nine leaf cases. Straightforward on paper, we found them tedious to formalise in Isabelle, which triggered the rearrangement shown here.

[16] Finiteness matters: the set $\{b_i = 1 - \frac{1}{i} : i = 1, 2, 3, \ldots\}$ has no maximum.

[17] Real numbers are not usually required for running auctions in *practice*. Even financial exchanges that allow 'sub-pennying' have a minimal discrete quantum of currency.

[18] For instance, our Mizar proof never invokes any second-order *scheme* directly. Two proof steps use the fact that a finite set of numbers includes its maximum, which is proved in the Mizar Mathematical Library (MML) using the induction scheme.

4 Qualitative Comparison of the Languages and Systems

We have formalised Vickrey's theorem in four systems, which differ in logic, syntax and user experience: Isabelle, followed by Mizar, CASL and Theorema. For each system at least one author has in-depth knowledge. The purpose of redoing formalisations from scratch is to understand the specific advantages and disadvantages of the systems and to obtain as idiomatic a formalisation as possible. The formalisations and instructions for using them are available from the ATT homepage [15]. Tab. 1 compares the features of the systems and their languages and shows the state of our work. The following subsections assess the languages and systems w.r.t. the technical requirements C* of §2. Tab. 2 at the end of this section summarises our findings to underpin our final recommendations.

4.1 Level of Detail and Explicitness Required (req. C1a)

All systems required greater detail and explicitness than the paper elaboration of §3.2. The Isabelle formalisation needs 3 additional definitions and 7 auxiliary lemmas. Guiding the automated provers of Theorema and Hets and Mizar's proof checker required similar numbers of auxiliary statements, plus, in Theorema and Hets, further ones to emulate proof steps (cf. §4.2). However, first steps beyond Vickrey's theorem suggests that these auxiliaries make it easier to formalise *further* notions. As our work involved beginners and experts[19], we can only approximately quantify the formalisation effort beyond the paper elaboration. The 'de Bruijn factor' [26], the formalisation size divided by the size of an informal TEX source, measured after stripping comments and *xz* compression, is around 1.5 for all formalisations[20] except Theorema[21]. This observation suggests that machine formalisation is generally still harder than elaboration on paper.

Even while explicit machine formalisation imposes tedious work on the author, it can also prove beneficial. On paper, it was neither immediately obvious that exactly one participant wins a second-price auction, nor that the outcome is a function of the bids. While obvious that at least two participants are required to define the 'second highest bid', the standard literature largely overlooks this, but formalisation forced us to choose whether to allow it (by, e.g., defining $\max \emptyset \equiv 0$) or to explicitly require $n \geq 2$.

4.2 Expressiveness vs. Efficiency (req. C1a)

As discussed in §3.2, we did not strictly take the elaborated paper source as a specification for the formalisation, but wrote idiomatic formalisations. In Isabelle and Mizar, we, e.g., avoided specific intervals $\{1, \ldots, n\}$ as sets of auction

[19] The Mizar formalisation was, e.g., completely written by an expert (Caminati), whereas the Isabelle formalisation was initially written by a first-time user with a general logic background (Lange), then largely rewritten by an expert (Wenzel).

[20] A typical average is 4, but our paper proof is particularly detailed.

[21] Determining a de Bruijn factor for Theorema does not make sense: single keystrokes or clicks may yield complex inputs, Mathematica notebooks store layout and maintenance information, and Theorema caches proofs in the notebook (cf. §4.6).

Table 1. Languages and systems we compared; state of our formalisations

Language	Logic	Prover	User Interface	Licence	Formalisation
Isabelle/HOL 2013 [17]	HOL (simply-typed set theory)	interactive[a]	document-oriented IDE (Isabelle/jEdit [18]) or programmer's text editor (Proof General Emacs [19])	BSD/LGPL/ GPL	complete incl. proof
Theorema 2.0 [20]	FOL + set theory[b]	automated[c]	textbook-style documents, proof management GUI (add-on for Mathematica CAS)	GPL[d]	statements complete, no proof[e]
Mizar 8.1.01 [21]	FOL[f] + set theory	batch verifier	CLI[g]; programmer's text editor (Emacs add-on)	freeware/GPL+ CC-BY-SA[h]	complete incl. proof
CASL/ TPTP[i] [22]	sorted FOL[i]	automated[c]	progr.'s text editor (Emacs add-on), proof mgmt. GUI+ CLI (Hets 0.98[j] [23]), web service (System on TPTP [24])	GPL	complete incl. proof

[a] Isabelle integrates internal and external automated provers.

[b] Theorema actually supports HOL. We, however, just needed FOL besides the built-in sets, tuples, and the max operator.

[c] The proof is largely automatic. However, Vickrey's theorem is too complex to for automated proving in one step. Thus, the proof script introduces auxiliary lemmas and selects suitable axioms and provers for proving them. Proof times range from fractions of seconds if the exact list of used axioms is known beforehand to hours if not. However, once a proof is found, the prover can output the list of used axioms and thus speed up subsequent replays of the proof.

[d] Theorema is under GPL but needs the commercial, closed-source Mathematica. Economists tend to be pragmatic about that.

[e] Theorema is in transition to the new 2.0. Its architecture, inference engine, and user interface are fully implemented, but its collection of *inference rules* is still incomplete. Therefore, the proof does not yet work.

[f] *Schemes* permit a limited degree of higher-order reasoning.

[g] The *verifier* produces a list of numerical errors codes and their source file positions. The ancillary utilities *errflag* and *addfmsg* decorate source files with this information, and optionally append terse textual explanations of the relevant error codes.

[h] The Mizar proof checker is closed-source; the MML is free.

[i] Common Algebraic Specification Language. 'CASL/TPTP' denotes our use of CASL as an input language for automated FOL provers (here: SPASS, E, Darwin) using the TPTP [25] exchange language. CASL features some second-order features, e.g. inductive datatypes.

[j] Heterogeneous tool set; gives access to a wide range of automated theorem provers. We use FOL provers, most of which share the unsorted TPTP FOF [25] as a common input format. Hets translates CASL to FOF by introducing auxiliary predicates for sorts.

participants: arbitrary (finite) sets of natural numbers simplify the formalisation, and the highest and second highest bids are determined using library set operations. In contrast, Theorema naturally indexes its built-in tuples from 1 to n and allows for restricting quantified variables to such ranges, e.g. $\forall_{i=1,\ldots,n}$.

The CASL formalisation confirms the assumption of §3.3 that FOL suffices for expressing and proving the essence of Vickrey's theorem. For many FOL provers, CASL's (sub)sorts[22] are mere syntactic sugar but allow us to stay close to the domain language, speaking, e.g., of 'valuation vectors', each of which also is a valid 'bid vector'. Note that we have avoided using partial functions (e.g., for modelling out-of-scope vector indices) because of the complex logic translations required for coding them out.

Isabelle and Mizar process the proof in a few seconds on a 2.5 GHz dual-core processor; Hets/TPTP need about an hour; in Theorema it is not yet complete. We used rather weak HOL features, e.g., no synthesisation of functions. Coinciding with earlier, general observations on HOL [28], the low processing time suggests that there is no disadvantage in choosing a rich logic, which allows for expressing relevant concepts (such as maxima of finite sets of real numbers) naturally. Our formalisations' small size (less than 5 K after compression) does not yet warrant a precise quantitative judgement of time efficiency. Particularly for FOL there exist highly optimised automated provers. They are conveniently accessible in Hets, via the System on TPTP [24] web service (accepting TPTP input that Hets can generate), but also from Isabelle/HOL via the Sledgehammer interface (see §4.3). Still, we observed a source of inefficiency in formalising for automated provers: the high share of preconditions with long conjunctions in our CASL formalisation makes it hard for the automated FOL provers to identify applicable axioms. Such conjunctions result from the absence of structured proofs in CASL. This requires, whenever a theorem is too complex for automated proving, to 'emulate' proofs steps via auxiliary lemmas, whose antecedents are conjunctions of all relevant assumptions in the current branch of the proof tree. Performance improvements by guiding provers through the search space can, however, be achieved with the extra effort of grouping frequently occurring conjunctions of assumptions into single abstract predicates, as in the following concrete case for the proof of Vickrey's theorem: $\mathit{spaWithTruthfulOrOtherBid}(n, x, p, v, \hat{\boldsymbol{b}}, i, \boldsymbol{b}) \Leftrightarrow \mathit{secondPriceAuction}(n, x, p) \wedge |v| = |\hat{\boldsymbol{b}}| = n \wedge \mathit{inRange}(n, i) \wedge \hat{\boldsymbol{b}}_i \neq v_i \wedge \boldsymbol{b} = \hat{\boldsymbol{b}}[i \leftarrow v]$.

4.3 Proof Development and Management (req. C2a)

The systems we studied offer different ways of invoking automated provers and keeping track of proof efforts in progress. The 'apparent' difference between automated and interactive theorem proving blurs at a closer look. The interactive prover Isabelle features various automated proof methods; furthermore Sledgehammer gives access to E, SPASS, and TPTP provers. One can configure the facts they should take into account (e.g. local assumptions and conclusions). For

[22] TPTP's typed first-order form (TFF [27]) is sorted, but without subsorts. We have not used it, as Hets cannot currently produce it from CASL.

Mizar, there are also automated external tools (MPTP, MoMM, MizAℝ) [29]. Theorema's automated proving workflow is conceptually similar: specifying the knowledge to be used, then configuring the prover.[23] Hets users can select axioms and previously proved theorems to be sent to an automated prover but have little control beyond. Isabelle's prover configuration is editable within the formalisation source. Theorema stores it in hidden fields within the formalisation and exposes it via a dedicated GUI. Configuring proof tools in Hets is separate from the formalisation: the proof management GUI does not currently store settings persistently; however one can write scripts to be processed on the command line.

Just as Isabelle requires complex statements to be proved in multiple steps, involving different proof methods, the automated provers of Theorema[24] and Hets also require guidance by explicit configuration at times, as can be seen from the `*.hpf` proof scripts in our Hets formalisation [15]. Often, a theorem $c : A \Rightarrow C$ was too complex for automated proving, whereas the job could be done by a script that first proved auxiliary lemmas $a : A \Rightarrow B$ and $b : B \Rightarrow C$, possibly with different provers, and then proved c providing only a and b as axioms. This is conceptually the same as in Isabelle but has four significant user experience differences: 1. Each additional 'proof step' has to be stated as a lemma with full assumptions on the left hand side (similar to the example in §4.2), 2. CASL, originally a specification rather than a prover language, does not syntactically distinguish theorems from lemmas, 3. the scripts have to be maintained separately from the formalisation, and 4. a multi-step proof takes many seconds longer, as Hets translates the input theory from CASL to the respective prover's native language before each proof.[25] This gives a clear incentive to eliminate unnecessary proof steps from a CASL formalisation. This experience also influenced our Isabelle formalisation, where writing multi-step proofs is comparatively painless. There, one lemma had a three-step proof, until experiments with the CASL formalisation made us attempt an automated proof. Thus we realised that we could reduce the Isabelle proof to a single step.[26]

Mizar differs by focusing, instead of built-in tactics and automated proof methods, on a natural deduction style which 'tries to "keep a low profile" in its logical foundations' and aims at 'clarity, human readability and closeness to standard mathematical proofs' [30]. Influenced by Mizar, the Isar language ('intelligible semi-automated reasoning') replaced Isabelle's original tactic interface. In the name of its readability focus, Mizar deliberately prevents users from extending the verifier's power [30, §2.1], often forcing them to justify trivial passages. Mizar's *registrations* do allow for custom automation [31]; however, these at times involute exploits often push registrations beyond their intended scope [32] and may result in implicit inferences and less readable proofs.

Particularly in developing the proof of a theorem as complex as Vickrey's top-down, it is useful to defer proofs of lemmas or proof steps, as to use them

[23] For Theorema, a prover is a *collection of inference rules* applied in a certain *strategy*.
[24] This assessment relies on experience with Theorema 1.
[25] This is necessary as, by default, each successful proof adds one theorem to the theory.
[26] As it makes use of one definition and two lemmas, this was not obvious a priori.

in a larger proof without the workaround of temporarily declaring them as axioms. Theorema proofs can use unproved theorems as knowledge. Isabelle's `sorry` keyword creates a fake proof. CASL theorems are formulas with the annotation `%implied`. When imported into a theory, (open) theorems become axioms, and Hets can use them without proof, but the open proof obligation is still visible in the imported theory. Mizar's verifier offers top-down proving for free by marking unaccepted inferences as errors *and then proceeding*. This results in a formal proof *sketch*, 'very close to informal mathematical English' but still close to a fully formalised proof [33]. Furthermore, one can prefix the keyword `proof` with `@` to expressly and silently skip a proof, or disable the verifier on arbitrary code portions using pragmas. Mizar's Emacs mode exposes these as one-touch macros, which speeds up the verification process and improves interaction [30].

4.4 Library Coverage and Searchability (reqs. C1c, C2b)

To a varying degree we have been able to reuse mathematical foundations from the systems' libraries. Isabelle can *find* reusable material by `find_theorems` queries; Sledgehammer helps to extract a sufficient set of lemmas from the library, which is then minimised towards a necessary set. MML Query is a search engine for the MML [34]. CASL's library is searchable as plain text; Theorema's is not.

Theorema has a built-in tuple type, including a maximum operation, we used it to formalise bid vectors. The CASL library provides inductive datatypes such as arrays [35] but no n-argument maximum operation. The Isabelle/HOL library provides a *Max* operation on finite sets, and various Cartesian product types suitable for representing bids. Given Isabelle's functional programming syntax we found it, however, most intuitive to model our own vectors as functions $\mathbb{N} \to \mathbb{R}$ evaluated up to a given n. Wrappers make the set maximum operator work on these vectors and prove the properties required subsequently. Our Mizar formalisation draws on generic relations and functions, which the MML richly covers. Thus, we only had to add a few interfacing lemmas.

4.5 Term Input Syntax (req. C1b)

Conversations with auction designers suggest that they find Theorema's term input syntax most accessible. The two-dimensional notation in Mathematica notebooks is similar to textbook notation, and our target audience is largely familiar with Mathematica. The syntax of Isabelle and CASL is closer to programming languages. Isabelle's functional type syntax $f : A \Rightarrow B \Rightarrow C$ looks less closely related to textbook notation than CASL's $f : A * B \to C$. Isabelle, CASL and Mizar allow for defining custom 'mixfix' operator notations. Isabelle provides rich translation mechanisms beyond that, but the layout remains one-dimensional, e.g. $\forall x \in A.\ B(x)$ instead of Theorema's $\underset{x \in A}{\forall} B[x]$ for bounded quantification. Isabelle Proof General and Isabelle/jEdit approximate textbook notation by Unicode symbols. Isabelle, Mizar and Hets can export LaTeX. Mizar uses ASCII; its lack of binders makes mathematical concepts such as limits and

sums cumbersome to denote [36]. A major reason for us not to cover the TPTP language is its technical, non-extensible ASCII syntax (using, e.g., `!/?` for $\forall/\exists$).

Theorema, CASL and Mizar support sharing common quantified variables across multiple statements, corresponding to the practice of starting a textbook section even of several axioms like 'let n, the number of participants, be a natural number ≥ 1'. This helps to avoid redundancy but is prone to copy/paste errors. For example, our CASL formalisation has sections with global quantifiers $\forall i, j$ (e.g. to accommodate the maximum and second-price auction definitions of §3.2), but these include axioms that only use i. Literally pasting into this axiom an expression using j does not cause an error, as j is bound in the current scope as well, but changes the semantics of the axiom in a way hard to detect.

4.6 Comprehensibility and Trustability of the Output (req. C2c)

Machine proofs may 'succeed' for unintended reasons, e.g. accidentally stating a tautology such as an implication with an unsatisfiable antecedent. Or they succeed as intended, but the user cannot follow the (automated) deduction. In such situations the prover's *output* is crucial. Isabelle provides tracing facilities for simplification rules and introduction and elimination rules used in standard reasoning steps.Its inference kernel can produce a full record (usually large and unreadable) of the internal reasoning of automated tools via explicit proof terms, e.g. for independent checking. By default the kernel relies on static ML type-discipline to achieve correctness by construction, without explicit proof terms. Theorema's proof data structure captures the entire proof generation according to the rules and strategy selected. It can be displayed as a structured textbook-style proof with configurable verbosity, and visualised as a browsable tree that distinguishes successful from failed branches. Mizar 'just' verifies what the user wrote according to natural deduction rules, hence he is unlikely to doubt the result. On the other hand, for the same reason, Mizar has no way to detect proofs succeeding for unintended reasons, and offers little help to a user clueless about a failing step. A correct Mizar proof can be improved by enhancer utilities [21, §4.6]: some report useful additional information (e.g., unneeded statements referred in a step, unneeded library files, unneeded lemmas); others cut steps that a human might want to see, impacting readability and possibly the original confidence the user had in the proof. Hets uniformly displays the success of a proof and the list of axioms used; however the latter output is only informative with SPASS. Otherwise, the raw technical output of the prover is displayed, which strongly differs across provers. E.g., SPASS uses resolution calculus, which looks different from a textbook proof. Similarly, System on TPTP outputs performance measures and the status of the given problem (e.g. 'Theorem' or 'Unsatisfiable'), but otherwise the raw prover output.

When a proof attempt fails because the statement was wrong, studying a counterexample may help. Isabelle has the Nitpick counterexample finder built in. Hets integrates several ones (Darwin is supported best [37]) and also employs them for consistency checking, as importing a theory whose axioms have no model results in vacuous truth. Both Isabelle and Hets can attempt a proof or

otherwise try to find a counterexample in the same run. Theorema and Mizar do not support counterexamples.

Before proving, all systems check whether the input is syntactically well-formed and well-typed. Isabelle/jEdit performs parsing, type checking and proof processing during editing, and attaches warnings and error messages like modern IDEs. The other systems require the user to explicitly initiate checking. Mizar and Hets check complete files, whereas in Theorema (which only checks syntax), one can individually check each notebook cell (typically containing one to a few statements). Mizar's verifier is particularly error resilient: it seldom aborts before the last input line, thus reporting errors for the whole file.

4.7 Online Community Support and Documentation (req. C2d)

Community support and documentation are major prerequisites for system adoption. We assume that users with little previous mechanised reasoning and formalisation knowledge will seek low-threshold support from tutorial documents or mailing lists rather than attending community meetings – which, in theorem proving, so far focus on scientific/technical aspects rather than applications.

We compare the community sizes, assuming that large communities are responsive even to non-experts: Isabelle is developed at multiple institutions; its user mailing list gets more than 100 posts a month, with over 1000 different authors since 2000. CASL, an international standard, has been subject of hundreds of publications but does not currently have a mailing list. Hets is mainly developed and used within a single institution; its user mailing list receives less than 10 posts a month. Recalling that Hets is an integrative environment, users can also request help from the communities of TPTP (subject of more than 1000 publications, no mailing list) and individual provers. Theorema is developed within a single institution and will not have a mailing list before the 2.0 release. Mizar is developed at one institution by a team that provides dedicated email user assistance: the 'Mizar User Service'. MML grows by 30–60 articles a year, with 241 contributors so far. The mailing list gets around 10 posts a month.

Isabelle and CASL feature comprehensive tutorials and reference manuals, Hets has a user guide, Mizar offers tutorials [38]. Theorema has partial built-in help texts and is documented in a few publications.

5 Related Work

§1 mentioned earlier efforts to *formalise economics*. Particularly Arrow's impossibility theorem, one of the most striking results in theoretical economics, has been a focus for formalisation efforts, including Nipkow's Isabelle and Wiedijk's Mizar formalisation [39, 40]. As in our case (cf. §3.2), they required initial paper elaboration; additionally, it helped them to identify omissions in their source [41]. This source states three alternative proofs, but Tang's/Lin's fourth, induction-based proof, allowed for obtaining insights on the general structure of social choice impossibility results using computer support [42].

Table 2. Performance (as far as results were comparable)

System/ Language	Proof speed	Textbook closeness		Top-down proofs	Library		Output			Community	Documentation	de Bruijn factor
		PI[a]	TI[a]		LC[a]	LS[a]	PO[a]	CE[a]	WF[a]			
	§4.2	§4.3	§4.5	§4.3	§4.4			§4.6		§4.7		§4.1
Isabelle/HOL	++[b]	++	+	++	++	++	○	++	++	++	++	1.3
Theorema	?	n/a[c]	++	++	+	—	++	n/a	–	—	–	n/a
Mizar	++	++	–	++	++	+	○	n/a	++	+	○	1.7
CASL/TPTP	○[d]	–	+	++	+	–	○	+	+	○	+	1.5

[a] PI/TI = proof/term input; LC/LS = library coverage/search; PO = proof output; CE = counterexamples (incl. consistency checks); WF = well-formedness check. [b] scores from very bad (—) to very good (++) [c] fully GUI-based [d] automated provers

The formal verification technique of model checking has been applied to *auctions*. Tadjouddine et al. proved the strategy statement of Vickrey's theorem via two abstractions to reduce the model checker's search space: program slicing to remove variables irrelevant w.r.t. the property, and discretising bid values (e.g. 'higher than someone's valuation v_i') [43]. Our formalisation is, to the best of our knowledge, the first for *theorem provers*; in the more expressive languages it has the comprehensibility advantage of preserving the structure of the original domain problem. From earlier economics formalisation efforts cited above, it differs in its goal to (ultimately) help economists to use formal methods themselves.

Our focus thus lies on *comparing* different provers by full parallel formalisation. Wiedijk compared Isabelle/HOL, Mizar, Theorema, and 14 other provers by general, technical criteria, studying the code resulting from experts formalising a pure mathematics theorem ($\sqrt{2} \notin \mathbb{Q}$), and comparing it to a detailed paper proof [44]. We complement this with the end user's perspective: our observations, e.g., on the closeness of the input syntax to textbook notation or the comprehensibility of the output are general, but we emphasised these criteria as they are important to auction designers. Griffioen's/Huisman's 1998 PVS and Isabelle/HOL comparison is, like Wiedijk's, independent from a specific application but closer to ours in its look at systems' weaknesses from a user's perspective [45]. Like us, they rate proof management and user support, but go into more detail up to the 'time it takes to fix a bug'. Their *findings* on user interfaces have been obsoleted by progress in developing textbook-like proof languages and editors with random access and asynchronous validation.

6 Conclusion and Outlook

Auctions allocate trillions of dollars in goods and services every year, but their design is still 'far less a science than an art' [13]. We aim at making it a science

by enabling auction designers to verify their designs. By parallel formalisation of the first major theorem in a toolbox for basic auction theory (ATT), we have investigated the suitability of four different theorem provers for this job, taking the perspective not only of experienced formalisers but also of our end users. Our contribution is 2×2-fold: 1. to auction designers we provide (a) a growing library to build their formalisations on, and (b) guidelines on what systems to use; 2. to the CICM community we provide (a) challenge problems[27] and (b) user experience feedback from a new audience. This paper focuses on 1b and 2b.

For a concrete application, our findings confirm the widespread intuitions that formalisation benefits from an initial paper elaboration, that the 'automated vs. interactive' distinction proves of little importance in practice, and that no single system satisfies all requirements. For now, our comparison results in Tab. 2 guide auction designers in choosing a system, given their formalisation requirements and experience. The ideal theorem proving environment would feature a *library* as versatile as in Isabelle or Mizar, a *prover* as efficient as those of Isabelle or Mizar, giving *error messages* as informative as in Isabelle/jEdit, further a *proof input language* as close to textbook style as those of Isabelle or Mizar, or an *interface to explore* automated proofs as informative as Theorema's, a *textbook-like term syntax* as Theorema's, an integration of diverse *tools* as in Isabelle or Hets, and a *community* as lively as Isabelle's. We have not yet exploited all strengths of the systems evaluated: maintaining a growing ATT with increasingly complex dependencies will benefit from stronger modularisation, as supported by Isabelle and even more so by the theory graph management of Hets/CASL. Regarding auction *practice*, we are working towards ways to check that formal definitions of auctions are well-defined functions ('for each admissible bid input there is a unique outcome, modulo some randomness'). Given a constructive proof of this property, it should be possible to obtain verified program code that determines the outcome of an auction given the bids. This may work using Isabelle's code generator, but we will also explore provers based on constructive type theory.

Broader conclusions about auction theory require further research. Bidding typically requires forming conjectures of others' beliefs, involving integration over conditional density functions (cf., e.g., Proposition 13 in Maskin's review [13]). We expect that much of the required foundations should already be available in the libraries of Isabelle and Mizar. Maskin limits his review to single good auctions, noting that few general results exist for multi-unit and combinatorial auctions.[28] Such auctions are often more economically critical (e.g. spectrum auctions, monetary policy [4]) but also more complicated. The real challenge for mechanised reasoning will be to demonstrate its use in this domain.[29]

[27] Our problems are not currently challenging systems' performance but the promises of their languages and libraries.

[28] The last two chapters of [14] address multi-unit auctions; multi-unit and combinatorial auctions are the focus of [46].

[29] Even more ambitiously, many results in auction theory are simplified or extended by explicit application of mechanism design; cf. [47].

References

1. *Auctions: The Past, Present and Future*. URL: http://realestateauctionglobalnetwork.blogspot.co.uk/2011/11/auctions-past-present-and-future.html.
2. Klemperer, P. *Auctions: theory and practice*. Princeton Univ. Press, 2004.
3. Conitzer, V. and T. Sandholm. 'Self-interested automated mechanism design and implications for optimal combinatorial auctions'. In: *Conference on Electronic commerce*. ACM, 2004.
4. Klemperer, P. 'The product-mix auction: a new auction design for differentiated goods'. In: *European Economic Association Journal* 8.2–3 (2010).
5. Lange, C., C. Rowat and M. Kerber. 'The ForMaRE Project – Formal Mathematical Reasoning in Economics'. In: *CICM*. LNCS. Springer, 2013.
6. Woodcock, J. et al. 'Formal method: practice and experience'. In: *ACM Computing Surveys* 41.4 (2009).
7. Kerber, M., C. Lange and C. Rowat. *An economist's guide to mechanized reasoning*. 2012. URL: http://cs.bham.ac.uk/research/projects/formare/.
8. Geist, C. and U. Endriss. 'Automated search for impossibility theorems in social choice theory: ranking sets of objects'. In: *Artificial Intelligence Research* 40 (2011).
9. Tang, P. and F. Lin. 'Discovering theorems in game theory: two-person games with unique pure Nash equilibrium payoffs'. In: *Artificial Intelligence* 175.14–15 (2011).
10. Kerber, M., C. Rowat and W. Windsteiger. 'Using *Theorema* in the Formalization of Theoretical Economics'. In: *CICM*. LNAI 6824. Springer, 2011.
11. *Initiative for Computational Economics*. URL: http://ice.uchicago.edu.
12. Wikipedia, ed. *Vickrey auction*. 2012. URL: http://en.wikipedia.org/w/index.php?title=Vickrey_auction&oldid=523230741.
13. Maskin, E. 'The unity of auction theory: Milgrom's master class'. In: *Economic Literature* 42.4 (2004).
14. Milgrom, P. *Putting auction theory to work*. Cambridge Univ. Press, 2004.
15. Lange, C. et al. *Auction Theory Toolbox*. 2013. URL: http://cs.bham.ac.uk/research/projects/formare/code/auction-theory/.
16. Lamport, L. and L. C. Paulson. 'Should your specification language be typed?' In: *ACM TOPLAS* 21.3 (1999).
17. *Isabelle*. URL: http://isabelle.in.tum.de.
18. Wenzel, M. 'Isabelle/jEdit – a Prover IDE within the PIDE framework'. In: *CICM*. LNAI 7362. Springer, 2012.
19. Aspinall, D. 'Proof General: A Generic Tool for Proof Development'. In: *TACAS*. LNCS 1785. Springer, 2000.
20. Windsteiger, W. 'Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System'. In: *UITP workshop at CICM*. 2012.
21. Grabowski, A., A. Korniłowicz and A. Naumowicz. 'Mizar in a Nutshell'. In: *Formalized Reasoning* 3.2 (2010).
22. *CASL*. URL: http://informatik.uni-bremen.de/cofi/wiki/index.php/CASL.
23. Mossakowski, T. *Hets: the Heterogeneous Tool Set*. URL: http://hets.dfki.de.

24. *System on TPTP*. URL: http://cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP.
25. Sutcliffe, G. 'The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0'. In: *Automated Reasoning* 43.4 (2009).
26. Wiedijk, F. *'De Bruijn factor'*. URL: http://cs.ru.nl/~freek/factor/.
27. Sutcliffe, G. et al. 'The TPTP Typed First-order Form with Arithmetic'. In: *LPAR*. LNAI 7180. Springer, 2012.
28. Farmer, W. M. 'The seven virtues of simple type theory'. In: *Applied Logic* 6.3 (2008).
29. Rudnicki, P., J. Urban et al. 'Escape to ATP for Mizar'. In: *Workshop Proof eXchange for Theorem Proving*. 2011.
30. Urban, J. 'MizarMode—an integrated proof assistance tool for the Mizar way of formalizing mathematics'. In: *Applied Logic* 4.4 (2006).
31. Caminati, M. B. and G. Rosolini. 'Custom automations in Mizar'. In: *Automated Reasoning* 50.2 (2013).
32. Korniłowicz, A. 'On Rewriting Rules in Mizar'. In: *Automated Reasoning* 50.2 (2013).
33. Wiedijk, F. 'Formal proof sketches'. In: *TYPES*. LNCS 3085. Springer, 2004.
34. Bancerek, G. 'Information Retrieval and Rendering with MML Query'. In: *MKM*. LNAI 4108. Springer, 2006.
35. Mosses, P. D., ed. *CASL Reference Manual*. LNCS 2960. Springer, 2004.
36. Wiedijk, F. 'The QED Manifesto Revisited'. In: *Studies in Logic, Grammar and Rhetoric* 10.23 (2007).
37. Mossakowski, T., C. Maeder and M. Codescu. *Hets User Guide*. Tech. rep. Version 0.98. DFKI Bremen, 2013. URL: http://informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/UserGuide.pdf.
38. *Mizar manuals*. 2011. URL: http://mizar.org/project/bibliography.html.
39. Nipkow, T. 'Social choice theory in HOL: Arrow and Gibbard-Satterthwaite'. In: *Automated Reasoning* 43.3 (2009).
40. Wiedijk, F. 'Formalizing Arrow's theorem'. In: *Sādhanā* 34.1 (2009).
41. Geanakoplos, J. D. *Three brief proofs of Arrow's impossibility theorem*. Discussion Paper 1123RRR. Cowles Foundation, 2001.
42. Tang, P. and F. Lin. 'Computer-aided proofs of Arrow's and other impossibility theorems'. In: *Artificial Intelligence* 173.11 (2009).
43. Tadjouddine, E. M., F. Guerin and W. Vasconcelos. 'Abstracting and Verifying Strategy-Proofness for Auction Mechanisms'. In: *Declarative Agent Languages and Technologies VI*. LNCS 5397. Springer, 2009.
44. Wiedijk, F., ed. *The Seventeen Provers of the World*. LNCS 3600. Springer, 2006.
45. Griffioen, D. and M. Huisman. 'A comparison of PVS and Isabelle/HOL'. In: *TPHOL*. LNCS 1479. Springer, 1998.
46. Cramton, P., Y. Shoham and R. Steinberg, eds. *Combinatorial auctions*. MIT Press, 2006.
47. Kirkegaard, R. 'A Mechanism Design Approach to Ranking Asymmetric Auctions'. In: *Econometrica* 80.5 (2012).