A Formal Proof of Vickrey's Theorem by Blast, Simp, and Rule

Department of Economics Discussion Paper 14-01

Manfred Kerber
Christoph Lange
Colin Rowat

A formal proof of Vickrey’s theorem by blast, simp, and rule

Manfred Kerber*

Christoph Lange[†]

Colin Rowat[‡]

January 5, 2014

Abstract

Formal methods use computers to verify proofs or even discover new theorems. Interest in applying formal methods to problems in economics has increased in the past decade, but – to date – none of this work has been published in economics journals. This paper applies formal methods to a familiar environment – Vickrey’s theorem on second-price auctions – and provides, as background, an introduction to formal methods.

Key words: formal proof, mechanized reasoning, auction theory

JEL classification numbers: B41; C63; C88; D44

1 Introduction

The production of theorems has been largely untouched by technological advances: a theorist manually produces a theorem and a proof, perhaps aided in certain calculations by computer algebra software; referees then manually check the proof. After publication, interested readers may seek to manually verify the proofs, leading to errata and corrigenda if not. For short proofs – as are typical in economic theory – this process has been accepted as offering a high level of reliability at relatively little effort: high level appeals may be made to symmetry and “obvious” steps, shortening proofs without much cost to their intelligibility. It also allows, through the citation of previous results, proofs to be reused.

While – for every individual involved in the process – the effort involved may be relatively minor, there may also be considerable duplication of effort.¹ Further, major new proofs in mathematics often run over one hundred pages long – well more than can be satisfactorily verified without enormous effort. Kepler’s conjecture that there is no denser packing of spheres in $\mathbb{R}^3$ than the face-centred cubic provides the most famous example: it took a team of 12 referees five years to become “99% certain” that Hales’ 120-page proof (excluding computer code that exceeded 500MB) was correct. Hales (2005) was not satisfied with this standard of proof, and founded *Project Flyspeck* to establish a *formal* proof of Kepler’s conjecture.

Formal proofs, in this sense, are encoded in an exacting way, typically to be read and checked by a software package called a *proof assistant*. Once the relevant terminology has

*School of Computer Science, University of Birmingham, UK

[†]Fraunhofer IAIS and University of Bonn, Germany

[‡]Department of Economics, University of Birmingham, UK. Corresponding author

¹Duplicated effort may not, of course, be wasted, but an important step in maintaining human capital.

been formally encoded, formal methods may allow automated search for proofs, or even for new results.

Inspired by Geanakoplos’s three brief proofs of Arrow’s impossibility theorem (Geanakoplos, 2005), computer scientists have recently become interested in applying formal proofs to economics. Initial work (Nipkow, 2009; Wiedijk, 2007; Wiedijk, 2009) used mechanized theorem provers to encode and verify them. Nipkow (2009) also used took advantage of the parallel proofs of Arrow’s and Gibbard-Satterthwaite’s theorems in Reny (2001) to formally derive the latter theorem as a corollary of Arrow’s.

Later work, drawing on the inductive proof of Arrow’s theorem in Suzumura (2000), has used formal methods to discover new theorems. Tang and Lin (2009) manually proved by induction that Arrow’s result held if it held on a base case of two agents and three alternatives. While even this small base case is too large to explore manually, a computer easily verified that none of the base cases allowed a social welfare function satisfying Arrow’s three axioms. Manual inspection of the base cases also allowed Tang and Lin (2009) to identify a new theorem subsuming Arrow’s. Tang and Lin (2011) used this approach – exhaustively generating and evaluating base cases, and then using a manual induction proof to generalize the results – to establish uniqueness conditions for pure strategy Nash equilibrium payoffs in two player static games. Geist and Endriss (2011) used the approach to generate 84 impossibility theorems in the ‘ranking sets of objects’ problem (Barberà, Bossert, and Pattanaik, 2004).

This note presents an early result of our work on applying formal methods to auction theory. As a side-effect, we hope to introduce formal methods to economists in a very simple context, a proof of Vickrey’s well-known theorem. The paper is related to Lange et al. (2013), which presented implementations of Vickrey’s theorem and proof in four different systems, chosen to compare a range of important mechanized reasoning systems. Much of the code used here is more general than necessary as it has been developed for an Auction Theory Toolbox (Lange et al., 2013), which we are currently using to address a central problem in auction design and practice, namely the provision of assurances that an auction design is well specified and implemented (Caminati et al., 2013).

Section 2 presents a standard statement of Vickrey’s theorem and its proof. Section 3 introduces formal proofs and mechanized reasoning in general; Section 4 describes a formal implementation of Vickrey’s theorem and its proof.

2 Vickrey’s theorem

Vickrey’s theorem concerns second price auctions, in which winners pay the highest bid after theirs and losers pay nothing. The notation, statement and proof in Maskin (2004) – which reviewed Milgrom (2004) – are not unusual:

Theorem 1 (Vickrey 1961; Milgrom 2.1). *In a second-price auction, it is (weakly) dominant for each buyer i to bid his valuation v_i (i.e. regardless of how other buyers bid, it is optimal for buyer i to set a bid of $b_i = v_i$). Furthermore, the auction is efficient.*

Proof #1. Suppose that buyer i bids $b_i < v_i$. The only circumstance in which the outcome for i is changed by its bidding b_i rather than v_i is when the highest bid b by other bidders satisfies $v_i > b > b_i$. In that event, buyer i loses by bidding b_i (for which its net payoff is 0) but wins by bidding v_i (for which its net payoff is $v_i - b$). Thus, it is *worse* off bidding $b_i < v_i$. By symmetric argument, it can only be worse off bidding $b_i > v_i$. We conclude that bidding its valuation (truthful bidding) is weakly dominant. Because it is optimal for buyers to bid truthfully and the high bidder wins, the second-price auction is efficient. $\square$

While perfectly intelligible – and even elegant – to humans, Maskin’s proof is too stylized for computers: the assertion that there is only one circumstance in which changing bids changes the outcome is not fully established; the “symmetric argument” is not explicitly elaborated. Less obviously, the case distinctions on the basis of participants’ bids (e.g. $b_i < b < v_i \dots$) yield nine leaf cases.

As a precursor to formalizing Vickrey’s theorem, we therefore first elaborated a more detailed paper proof. We also restructured the proof to involve only four cases, rather than the original nine:

Proof #2. Let N be the set of bidders, and suppose bidder i bids $b_i = v_i$, whatever b_j each other bidder $j \neq i$ bids. There are two cases:

1. i wins. This implies $b_i = v_i = \max_{j \in N} \{b_j\}$, $p_i = \max_{j \in N \setminus \{i\}} \{b_j\}$, and $u_i(\mathbf{b}) = v_i - p_i \geq 0$. Now consider i submitting an arbitrary bid $\hat{b}_i \neq b_i$ so that the bid vector is $(b_1, \dots, b_{i-1}, \hat{b}_i, b_{i+1}, \dots, b_n)$. This has two sub-cases:
 - (a) i wins with this new bid, so that $u_i(b_1, \dots, b_{i-1}, \hat{b}_i, b_{i+1}, \dots, b_n) = u_i(\mathbf{b})$: i receives the same utility from winning the item, and pays the same price as the second highest bid has not changed.
 - (b) i loses with this new bid, so that $u_i(b_1, \dots, b_{i-1}, \hat{b}_i, b_{i+1}, \dots, b_n) = 0 \leq u_i(\mathbf{b})$.
2. i loses. This implies $p_i = 0$, $u_i(\mathbf{b}) = 0$, and $b_i \leq \max_{j \in N \setminus \{i\}} \{b_j\}$ as, otherwise, i would have won. This yields again two cases for i ’s alternative bid $\hat{b}_i$:
 - (a) i wins, so that $u_i(b_1, \dots, b_{i-1}, \hat{b}_i, b_{i+1}, \dots, b_n) = v_i - \max_{j \in N \setminus \{i\}} \{b_j\} = b_i - \max_{j \in N \setminus \{i\}} \{b_j\} \leq 0 = u_i(\mathbf{b})$.
 - (b) i loses, so that $u_i(b_1, \dots, b_{i-1}, \hat{b}_i, b_{i+1}, \dots, b_n) = 0 = u_i(\mathbf{b})$.

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

3 Formal proofs and mechanized reasoning

While formal proofs can be presented – without reference to computers – as the next step in mathematics’ march towards greater rigor (Wiedijk, 2008), they are most often seen as inputs to mechanized reasoners. Mechanized reasoners are computer programs that perform logical operations, in the same way that computer algebra systems (CAS) like Mathematica, Maple or Sage perform algebraic operations.

The idea of mechanizing reasoning dates back at least to Leibniz (1686), who envisaged a machine which could compute the validity of arguments and the truth of mathematical statements. The development of formal logic from 1850 to 1930, the advent of the computer, and the inception of *artificial intelligence* (AI) as a research field at the Dartmouth Conference in 1956 all paved the way for the first mechanized reasoners in the 1960s, most notably de Bruijn’s Automath.

Since then, the project has been both more and less successful than anticipated. In pure maths, mechanized reasoning has only helped prove a few high-profile theorems: in addition to Kepler’s conjecture, these include Robbins’ conjecture about bases for Boolean algebras (McCune, 1997) and the four-color map theorem. Furthermore, the first proofs of both Kepler’s conjecture and the four-color map theorem used computers to carry out computational exercises (q.v. Appel and Haken, 1977; Appel, Haken, and Koch, 1977), rather than logical ones; only later have mechanized proof checkers been used to confirm these results (q.v. Gonthier, 2008).

However, mechanized reasoning has been more successfully applied to translating existing human proofs into formal proofs that can be verified by a proof assistant: as of January 2014, 88 of the ‘top 100’ mathematical theorems on a list maintained by Wiedijk (2013) had been formalized. Buchberger (2006) has suggested that this apparent disparity between discovering new proofs and formalizing existing ones can be explained by the fact that automated reasoning has so far focused on proving individual theorems, whereas mathematicians in practice have rather built up and explored entire theories and thus invented new notions, proved theorems about them, and developed new proof techniques as needed. In practice, the contrast is not as stark as suggested by Buchberger. For example, the comprehensive SSReflect library for the Coq proof assistant was initially developed to support the proof of the four-color map theorem (Whiteside, Aspinall, and Grov, 2012) but has also been widely used in other formalization efforts. Similarly, we had to develop general-purpose extensions to the Isabelle library to support our proof effort (cf. figure 4).

Perhaps surprisingly – although consistent with the greater success of applied AI over ‘pure’ AI – mechanized reasoning and formal methods have enjoyed greater success in industrial applications. In software engineering, any computer program defines a logical universe within which certain statements may or may not be true. Proof assistants can seek to prove or disprove these statements as theorems. Perhaps the best known examples of this come from transport and finance: in code controlling automated commuter rail systems, theorems that no two trains occupy the same location at the same time have been proved; within financial transactions software, theorems that transactions do not create or destroy value, but merely transfer it, have also been proved (Woodcock et al., 2009). In 2013, Facebook acquired Monoidics, a start-up firm applying theorem proving to software code analysis, marking a possible breakthrough of these techniques into the mainstream.

In hardware engineering, Intel has used theorem provers since the mid-1990s, following an embarrassing and costly recall after a chip was discovered not to properly implement the IEEE floating point division standard. By viewing digital chips as a set of Boolean statements, a prover may ask whether particular theorems (e.g. “this chip implements the IEEE division standard”) hold within the world defined by those statements (Harrison, 2006).

The differences between these uses of mechanized reasoning extend beyond their applications. Most importantly, systems differ in their *logics* which, in turn, are comprised of a *syntax* (which defines well-formed formulae), a *semantics* (which assigns those formulae a meaning) and a *calculus* (a set of inference rules which allow derivation of formulae, called theorems, from other formulae, called premises). Together, the syntax and semantics determine a logic’s expressiveness; the calculus may or may not be sound, complete and decidable. To explain, denote a set of premises by Γ , and a formula by φ ; then $\Gamma \models \varphi$ means “ φ follows from Γ ” (semantically) while $\Gamma \vdash \varphi$ means “ φ can be derived from Γ ” (in the calculus). A calculus is *sound* if a property that can be derived by it actually follows from its premises: $(\Gamma \vdash \varphi) \rightarrow (\Gamma \models \varphi)$. A calculus is *complete* if a property that follows from its premises can be derived by it: $(\Gamma \models \varphi) \rightarrow (\Gamma \vdash \varphi)$. Finally, a calculus is *decidable* if there is a procedure that, for any Γ and φ , either produces a derivation of φ from Γ or proves that no such derivation exists; a calculus is *semi-decidable* if, for any Γ and φ such that $\Gamma \models \varphi$, there exists a procedure that produces a derivation of φ from Γ .

Intel can express basic aspects of its chip designs and the theorems that it seeks to prove on them in *propositional* or *Boolean logic*, the simplest classical logic. This allows only the use of propositional variables – which may be either true or false – and *connectives* such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\Rightarrow$ (implies) (e.g. *first_bidder_bids_highest* $\wedge$ *second_bidder_bids_lowest*). While lacking in expressiveness due to its restriction to concrete, finite statements, proposi-

tional logic has a sound, complete and decidable calculus.

First-order logic (FOL) enriches the expressiveness of propositional logic in two ways. First, it is possible to speak about objects (e.g. bidder “ i ”) and their properties (e.g. “bidder i wins auction”, $wins(i)$). Second, $\exists$ and $\forall$ allow quantification over objects (e.g. “every losing bidder pays nothing”, $\forall i. \neg wins(i) \Rightarrow pay(i) = 0$, or “there exists a winner”, $\exists x. wins(x)$). Expressions such as $wins$ are called *predicate symbols*, Boolean functions which – when applied to their arguments – evaluate to either true or false. Gödel’s completeness theorem proved that FOL has a sound and complete calculus, but the price of FOL’s greater expressiveness is that no such calculus is decidable; FOL, however, has semi-decidable calculi.

Finally, *higher-order logic* (HOL) enriches the expressiveness of FOL by extending quantification to predicates and functions. Further, unlike FOL, it allows predicates and functions to take other predicates and functions as arguments (e.g. truthful bidding is a weakly dominant strategy in auction A for bidders N : $equilibrium_weakly_dominant_strategy\ N\ v\ b\ A$). Against this, HOL’s calculi are not decidable, and they are – by Gödel’s incompleteness theorem – incomplete.

Thus, the choice of calculus has implications for the degree of automation possible. However, decidable calculi generally facilitate automated theorem proving, computationally hard problems in propositional logic may still be intractable; similarly, simple problems expressed in HOL may be proved fully automatically. Typically, difficult theorems will require user interaction.

This paper uses Isabelle, a major HOL proof assistant: while bid vectors could easily be expressed in FOL as functions from bidders to their bids, quantifying over “all bids” requires HOL if the formal representation is to remain close to manual mathematics. (For example, quantification over all bids within FOL would require modelling them as variables and employing an auxiliary ternary predicate that evaluates to true whenever “bid vector b has value x for bidder i ”).

4 Blueprint of a formal proof of Vickrey’s theorem

The blueprint of a formal proof is not the same as a formal proof, which is a fleeting pattern of bits in a computer. Hales, 2012

This section guides the reader through the formal proof written for Isabelle. The formal proof itself is available as *Vickrey.thy*, a 9 KB Isabelle file containing 185 lines of code that draws on five ancillary bodies of theory written for this project, each stored in a separate file.² The latter files also contain code needed for proving different properties of different auctions; the overall code needed for proving Vickrey’s theorem (including *Vickrey.thy*) amounts to 17 KB and 404 lines. Such figures are usually distorted by the usage of whitespace, comments, documentation and long, self-explaining identifiers (such as *equilibrium_weakly_dominant_strategy* vs. WDS) in the interest of readability. The “de Bruijn factor” (Wiedijk, 2012) gives a more reliable estimate of the effort of writing a formal proof. It is computed by dividing the size of the code by the size of an informal T_EX source, measured after stripping comments and compression with a state-of-the-art general-purpose file compression tool. In our case, the T_EX source of Proof #2 and of the definitions it requires³ amounts to 2.6 KB after cleanup and compression, whereas the Isabelle code amounts to 2.9 KB, leading to a de Bruijn factor of 1.1.

²See <https://github.com/formare/auctions/tree/master/isabelle/Auction> for the code.

³A self-contained document is available on our Auction Theory Toolbox homepage (Lange et al., 2013).

Typically, factors of around 4 have been observed for formal proof efforts (Wiedijk, 2012), but recall that our $\text{\TeX}$ source is already more elaborate than usual paper definitions or proofs: it both defines terms like `max`, and eschews the shorthand appeals of Proof #1.

Figure 4 provides a high-level depiction of our Isabelle files. Ellipses denote files already existing within Isabelle’s library. Dotted ellipses denote files containing general purpose definitions and lemmas that we wrote when Isabelle’s library did not contain them. Rectangles denote files specific to auctions developed for this paper. Directed edges denote dependence among the files, with the source code being imported into the target code.

Figure 1: High level theory graph for the formal proof of Vickrey’s theorem

The formalization in *Vickrey.thy* first breaks Vickrey’s theorem into two parts, *vickreyA*, which proves that truthtelling is a weakly dominant strategy in second-price auctions, and *vickreyB*, which proves that this is efficient. In Isabelle, *vickreyA* reads:

```

theorem vickreyA :
  fixes  $N :: \text{"participant set"}$  and  $v :: \text{valuations}$  and  $A :: \text{single\_good\_auction}$ 
  assumes  $val : \text{"valuations } N \ v"$ 
  defines  $b \equiv v$ 
  assumes  $spa : \text{"second\_price\_auction } A"$  and  $card\_N : \text{"card } N > 1"$ 
  shows  $\text{"equilibrium\_weakly\_dominant\_strategy } N \ v \ b \ A"$ 

```

The **theorem** statement names the theorem for later reference, while the **fixes** statement defines it to apply to any N , v and A of the given types, where $::$ separates an expression from its type. In Isabelle’s *typed* language, subsequent uses of variables must respect their types. (These

type annotations are optional as long as Isabelle can infer them from the context; we make most of them explicit for clarity.) The **defines** declaration makes our interest in truthful bidding explicit, setting bids to be equal to valuations within the scope of the theorem. The **assumes** keyword introduces assumptions, which may be labeled: the second and third assumptions, for example, assume A to be a second-price (Vickrey) auction, and there to be more than one bidder. Finally, the **shows** keyword states the theorem: for a set N of agents participating in an auction A , with valuations v , setting bids b (equated to valuations) yields an equilibrium in weakly dominant strategies. The latter is defined as follows in *SingleGoodAuctionProperties.thy*:

definition *equilibrium_weakly_dominant_strategy* ::

“*participant set* $\Rightarrow$ *valuations* $\Rightarrow$ *bids* $\Rightarrow$ *single_good_auction* $\Rightarrow$ *bool*” **where**

“*equilibrium_weakly_dominant_strategy* $N \ v \ b \ A \longleftrightarrow$

$\text{valuations } N \ v \wedge \text{bids } N \ b \wedge \text{single_good_auction } A \wedge$

$(\forall i \in N .$

$(\forall \text{whatever_bid} . \text{bids } N \ \text{whatever_bid} \longrightarrow$

$(\text{let } b' = \text{whatever_bid}(i := b \ i)$

$\text{in } (\forall x \ p \ x' \ p' . ((N, \text{whatever_bid}), (x, p)) \in A \wedge ((N, b'), (x', p')) \in A$

$\longrightarrow \text{payoff } (v \ i) \ (x' \ i) \ (p' \ i) \geq \text{payoff } (v \ i) \ (x \ i) \ (p \ i))))$ ”

The definition’s optional second line specifies the types of the objects involved in the concept by indicating that we are defining a predicate whose arguments are a set of participants, a valuation and a bid vector, and an auction. (Function types, rendered as $A \Rightarrow B \Rightarrow C$ in Isabelle, would more commonly be written $A \times B \rightarrow C$ on paper.) The body of the definition then states that the predicate *equilibrium_weakly_dominant_strategy* given arguments N , v , b and A evaluates to true if and only if the remaining expression does. The *whatever_bid*($i := b \ i$) notation takes the arbitrary but fixed bid vector *whatever_bid* and replaces its i th component with i ’s original bid $b \ i$; in the proof below the latter will be the truthful bid. The **let** keyword has its natural meaning. We model an auction as an (*input*, *outcome*) relation, where the input consists of the set of bidders and their bids and the outcome comprises a Boolean allocation vector and a vector of transfers. As the theorem only requires varying i ’s bid, given an arbitrary vector of bids by other agents, we denote the outcome arising from everyone placing an arbitrary bid (*whatever_bid*) by (x, p) , while (x', p') denotes the outcome arising from i placing its original bid while the others still bid arbitrarily.

The definition therefore combines elements from propositional, first order and higher order logics. Propositional logic lends its connectives (e.g. $\wedge$), FOL its quantification over variables (e.g. $\forall i \in N$), and HOL its quantification over functions (e.g. $\forall \text{whatever_bid}$).

Formal proofs have tended to be written in one of two *proof styles*. *Procedural* proofs state, in the proof assistant’s formal language, a sequence of *tactics* to apply to the *proof obligations* or *goals* that remain on the *goal stack* which, when empty, indicate that the proof is complete. Procedural proofs therefore prescribe how to search for a proof, and may thus resemble the search process used by a human. By contrast, *declarative* proofs resemble the final, written version of human proofs, using the proof assistant’s formal language to bridge the gap between assumptions and a conclusion by deriving implications of the former and sufficient conditions for the latter, typically with smaller steps than those taken by humans. Given the additional difficulty associated with writing and reading procedural proofs, they are falling out of fashion relative to declarative proofs. While Isabelle allows both declarative and procedural proofs, our proof is written in the declarative style, with occasional calls to built-in automated proof methods. The snippet below concentrates on case 2b (reorganised for expositional clarity) of

the second paper proof:⁴

```

1 proof –
2   (*...*)
3   {
4     fix i :: participant
5     (*...*)
6     let ?b = “whatever_bid(i := b i)”
7     (*...*)
8     have weak_dominance : “payoff (v i) (x' i) (p' i) ≥ payoff (v i) (x i) (p i)”
9     proof cases
10      assume non_alloc : “x' i ≠ 1”
11      with spa_pred' i_range have “x' i = 0” using spa_allocates_binary by blast
12      with spa_pred' i_range have loser_payoff : “payoff (v i) (x' i) (p' i) = 0”
13      by (rule second_price_auction_loser_payoff)
14      have i_bid_at_most_second : “?b i ≤ ?b_max”
15      proof (rule ccontr)
16        assume “¬?thesis”
17        then have “?b i > ?b_max” by simp
18        with defined spa_pred' i_range have “second_price_auction_winner N ?b x' p' i”
19        by (simp add : only_max_bidder_wins)
20        with non_alloc show False
21        unfolding second_price_auction_winner_def
22        second_price_auction_winner_outcome_def by blast
23      qed
24      show ?thesis
25      proof cases
26        assume “x i ≠ 1”
27        then have “x i = 0” by (rule spa_allocates_binary')
28        with spa_pred i_range have “payoff (v i) (x i) (p i) = 0”
29        by (rule second_price_auction_loser_payoff)
30        also have “... = payoff (v i) (x' i) (p' i)” using loser_payoff ..
31        finally show ?thesis by (rule eq_refl)
32      next
33        (*...*)
34      qed
35    next
36      (*...*)
37    qed
38  }
39  (*...*)
40 qed

```

The proof begins with the **proof** keyword. Invoked alone, Isabelle would automatically select matching inference rules to apply.⁵ We write **proof** – to suppress this automatic feature in favour of supplying our own declarative steps. In lines 9 and 25, on the one hand, and 15, on

⁴We use “...” to distinguish the standard use of ellipses from “...” in the Isabelle code, whose meaning we introduce when explaining **also have** in line 30 of the code snippet.

⁵The calculus, i.e. the set of inference rules, is extensible, allowing new lemmas to be added as inference rules.

the other, we explicitly name the inference rules to apply:

- **proof** *cases* makes a case distinction; analysis of a case concludes by **showing** that, in this case, the desired thesis holds;⁶ **qed** clears the goal stack, and **next** begins consideration of the next case.
- **proof** (*rule ccontr*) seeks to prove a result by contradiction, and therefore culminates in **show False**.

The proof’s body is composed of a sequence of **have** statements, which assert local proof obligations; the ensuing **proof** statement – indented for readability – seeks to prove the obligation. Here, the case distinction proof sets out to establish that $u_i(\dots, v_i, \dots) \geq u_i(\dots, b_i, \dots)$: given i ’s valuation, i ’s payoff from outcome (x', p') (which arises from i bidding truthfully), exceeds that from the outcome (x, p) resulting from an arbitrary bid by i .

The **have** statements often appear in sentences like **from** $\dots$ **have** $\dots$ **by** $\dots$ or **have** $\dots$ **using** $\dots$ **by** $\dots$. Both **from** and **using** introduce knowledge to be applied in discharging the **have** proof obligation. Our proof actually uses the shorthands **then** (“**from** the previously established statement”) and **with** $\dots$ (“**from** the previously established statement and the further facts provided”). The **by** keyword invokes an automated proof method (an example of *automated theorem proving*), an alternative to discharging simple proof obligations by explicit declarative proofs (an example of *interactive theorem proving*). The automated proof methods used here include:

1. *simp* (lines 17 and 19) automatically applies simplification rules (e.g. $x \wedge x = x$), to rewrite the statement to be proved. The user may supply additional simplification rules to be used, such as *only_max_bidder_wins* in line 19.
2. *blast* (lines 11 and 22) “is (in principle) a complete proof procedure for first-order formulas” (Nipkow, 2013), with particular strength in “logic, sets and relations” such as $x = y$. In practice, *blast* has an upper bound for the number of unsafe inference rules to be applied in a proof (20 unless specified explicitly). Unsafe rules may reduce a provable goal to an unprovable set of subgoals; for example, the rule that from P one can derive $P \vee Q$ is unsafe, as its application to the goal $P \vee Q$ would introduce the subgoal P , which might be unprovable, whereas Q might still be provable (cf. Wenzel et al., 2013, sec. 9.4.2). Unlike *simp*, *blast* does not rewrite statements, nor does it provide feedback unless certain tracing options for expert users are enabled; in practice, *blast* either succeeds, fails, or – as an example of what semi-decidability means in practice – runs for a very long time until the user cancels it.
3. *rule* (lines 13, 15, 27, 29 and 31) applies the given lemma as an inference rule. For example, line 13’s lemma *second_price_auction_loser_payoff* reads **assumes** “*spa_pred* N b x p ” **and** “ $i \in N$ ” **and** “ $x\ i = 0$ ” **shows** “*payoff* ($v\ i$) ($x\ i$) ($p\ i$) = 0”, where the predicate *spa_pred* N b x p is defined to hold when x and p are the outcome of a second price auction with inputs N and b . Towards the end of the listing we have established the facts *spa_pred* N *whatever_bid* x p (abbreviated by the label *spa_pred*) and $i \in N$ (abbreviated *i_range*). Together with $x\ i = 0$ from the proof step in line 27, we have established the complete set of assumptions of the lemma, which we apply in line 29 as a rule to establish the local goal *payoff* ($v\ i$) ($x\ i$) ($p\ i$) = 0. Finally, “ $\dots$ ” in line 30 is an abbreviation for **by rule**, which automatically applies a matching inference rule, same as stated above for **proof**.

⁶More generally, **show** restates a proof obligation at its final step, when it can finally be discharged.

In developing this proof, we have used the **try** and **try0** keywords, which apply a range of methods to the problem at hand. We then used the feedback provided, which included the length of time taken, to select the most appropriate method.

The automated calls can be seen as akin to high level appeals to “the only circumstance” or “symmetric arguments” in human proofs. In all cases, they could be replaced by manually provided declarative steps. In some cases, Isabelle’s Sledgehammer tool (Blanchette and Paulson, 2013) can automatically provide the declarative steps. Note, however, that arguments on a level as high as “the only circumstance” or “symmetry” are still rarely supported by automated reasoners.

The **assume** ... **then have** constructions (lines 16 and 17, and 26 and 27) list assumptions **then** state the proof obligations. Line 16’s identifier *?thesis* refers to the proof obligation at the current level of reasoning in the proof; $\neg ?thesis$ is its negation. Isabelle syntactically substitutes identifiers starting with *?* – including *?b* in our listing – by other, usually more complex expressions before checking a proof step, akin to macros in programming languages such as C: before the code is compiled, a preprocessor expands occurrences of macros by their full definitions. This is distinct from, for example, the (semantic) equation of two variables, as in “ $b \equiv v$ ”.

The **unfolding** keyword applied in lines 21 to 22 also performs substitutions, replacing the names of the two concepts stated with the bodies of their definitions. Unlike abbreviations with *?*, the latter need to be semantic definitions, which the reasoner can access (e.g. *second_price_auction_winner_def* is restated in terms of $i \in N, i \in \arg \max b$, etc.).

Finally, lines 28–31’s **have** ... **also have** ... **finally show** construction allows chains of reasoning with equality before finally discharging a proof obligation: the “...” following the **also have** are replaced by the right hand side of the previous **have** statement. In line 30, this establishes that bidder *i*’s payoff is zero given valuation v_i and either the outcome (x, p) , or (x', p') .

In closing, Figure 2 shows how the Isabelle/jEdit proof development environment communicates with the user when checking a proof. The upper half shows lines 26–29 with the fact *i_range* omitted in the third line. The lower half shows that – without this fact – the assumptions under which the rule *second_price_auction_loser_payoff* applies are not completely satisfied, producing an error message. The absence of such error messages means that Isabelle confirms that a proof is correct.

Figure 2: Feedback about a wrong use of the *rule* automated proof method

Acknowledgements We are grateful to Makarius Wenzel for help refining our Isabelle code, to Peter Cramton, Peter Postl, Rajiv Sarin and Ron Smith for comments, and to the EPSRC for funding (EP/J007498/1). Rowat thanks Birkbeck for its hospitality.

References

- Appel, K. and W. Haken (1977). “Every Planar Map is Four Colorable Part I: Discharging”. In: *Ill. J. of Math.* 21.3, pp. 429–490.
- Appel, K., W. Haken, and J. Koch (1977). “Every Planar Map is Four Colorable Part II: Reducibility”. In: *Ill. J. of Math.* 21.3, pp. 491–567.
- Barberà, S., W. Bossert, and P. K. Pattanaik (2004). “Ranking sets of objects”. In: *Handbook of Utility Theory*. Ed. by S. Barberà, P. J. Hammond, and C. Seidl. Vol. II. Dordrecht: Kluwer, pp. 893–977.
- Blanchette, J. C. and L. C. Paulson (Dec. 5, 2013). *Hammering Away. A User’s Guide to Sledgehammer for Isabelle/HOL*. URL: <http://isabelle.in.tum.de/dist/Isabelle2013-2/doc/sledgehammer.pdf>.
- Buchberger, B. (2006). “Mathematical Theory Exploration”. In: *Automated Reasoning*. Ed. by U. Furbach and N. Shankar. LNCS 4130. Springer, pp. 1–2. doi: 10.1007/11814771_1.
- Caminati, M. B., M. Kerber, C. Lange, and C. Rowat (2013). *Proving soundness of combinatorial Vickrey auctions and generating verified executable code*. arXiv:1308.1779 [cs.GT].
- Geanakoplos, J. D. (July 2005). “Three brief proofs of Arrow’s impossibility theorem”. In: *Econ. Theory* 26.1, pp. 211–215.
- Geist, C. and U. Endriss (2011). “Automated search for impossibility theorems in social choice theory: ranking sets of objects”. In: *J. of Artif. Intell. Res.* 40 (January–April), pp. 143–174.
- Gonthier, G. (Dec. 2008). “Formal proof – the four color theorem”. In: *Not. of the AMS* 55.11, pp. 1382–1393.
- Hales, T. (Sept. 6, 2012). *Dense Sphere Packings. A Blueprint for Formal Proofs*. London Mathematical Society Lecture Note Series 400. Cambridge University Press.
- Hales, T. C. (Nov. 2005). “A proof of the Kepler conjecture”. In: *Annals of Mathematics* 162.3, pp. 1063–1185.
- Harrison, J. (May 2006). “Floating-Point Verification using Theorem Proving”. In: *Formal Methods for Hardware Verification*. Ed. by M. Bernardo and A. Cimatti. LNCS 3965. Springer, pp. 211–242.
- Lange, C., C. Rowat, M. Kerber, M. Wenzel, M. Caminati, T. Mossakowski, and W. Windsteiger (Jan. 13, 2013). *Auction Theory Toolbox*. URL: <http://cs.bham.ac.uk/research/projects/formare/code/auction-theory/> (visited on 01/13/2013).
- Lange, C., M. B. Caminati, M. Kerber, T. Mossakowski, C. Rowat, M. Wenzel, and W. Windsteiger (2013). “A Qualitative Comparison of the Suitability of Four Theorem Provers for Basic Auction Theory”. In: *Intelligent Computer Mathematics*. Ed. by J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger. LNCS 7961. Springer, pp. 200–215. ISBN: 978-3-642-39319-8. doi: 10.1007/978-3-642-39320-4. arXiv:1303.4193 [cs.LG].
- Leibniz, G. W. (1686). “Projet et Essais pour arriver à quelque certitude pour finir une bonne partie des disputes et pour avancer l’art d’inventer”. In: *Logik-Texte: Kommentierte Auswahl zur Geschichte der modernen Logik*. Ed. by K. Berka and L. Kreisler. Deutsche Übersetzung aus G. W. Leibniz, *Fragmente zur Logik*, Akademie-Verlag, Berlin, 1960. Berlin, Deutschland: Akademie-Verlag. Chap. I.1, pp. 15–17.

- Maskin, E. (Dec. 2004). “The unity of auction theory: Milgrom’s master class”. In: *Journal of Economic Literature* 42.4, pp. 1102–1115. URL: http://scholar.harvard.edu/files/maskin/files/unity_of_auction_theory.pdf.
- McCune, W. (Dec. 1997). “Solution of the Robbins problem”. In: *J. of Autom. Reason.* 19.3, pp. 263–276.
- Milgrom, P. (2004). *Putting auction theory to work*. Churchill lectures in economics. Cambridge University Press.
- Nipkow, T. (2009). “Social choice theory in HOL: Arrow and Gibbard-Satterthwaite”. In: *J. of Autom. Reason.* 43.3, pp. 289–304.
- (Dec. 5, 2013). *Programming and Proving in Isabelle/HOL*. URL: <http://isabelle.in.tum.de/dist/Isabelle2013-2/doc/prog-prove.pdf>.
- Reny, P. J. (Jan. 2001). “Arrow’s theorem and the Gibbard-Satterthwaite theorem: a unified approach”. In: *Econ. Lett.* 70.1, pp. 99–105.
- Suzumura, K. (Mar. 2000). “Welfare economics beyond welfarist-consequentialism”. In: *Jpn. Econ. Rev.* 51.1, pp. 1–32.
- Tang, P. and F. Lin (July 2009). “Computer-aided proofs of Arrow’s and other impossibility theorems”. In: *Artif. Intell.* 173.11, pp. 1041–1053.
- (Sept. 2011). “Discovering theorems in game theory: two-person games with unique pure Nash equilibrium payoffs”. In: *Artif. Intell.* 175.14–15, pp. 2010–2020.
- Wenzel, M., C. Ballarin, S. Berghofer, J. Blanchette, T. Bourke, L. Bulwahn, A. Chaieb, L. Dixon, F. Haftmann, B. Huffman, G. Klein, A. Krauss, O. Kunčar, T. Nipkow, L. Noschinski, D. von Oheimb, L. Paulson, S. Skalsberg, and C. Sternagel (Dec. 5, 2013). *The Isabelle/Isar Reference Manual*. Ed. by M. Wenzel. URL: <http://isabelle.in.tum.de/dist/Isabelle2013-2/doc/isar-ref.pdf>.
- Whiteside, I., D. Aspinall, and G. Grov (2012). “An Essence of SSReflect”. In: *Intelligent Computer Mathematics*. Ed. by J. Jeuring, J. A. Campbell, J. Carette, G. Dos Reis, P. Sojka, M. Wenzel, and V. Sorge. LNAI 7362. Springer, pp. 186–201. ISBN: 978-3-642-31373-8.
- Wiedijk, F. (2007). “Arrow’s impossibility theorem”. In: *J. of Formaliz. Math.* 15.4, pp. 171–174.
- (2008). “Formal proof – Getting Started”. In: *Not. of the AMS* 55.11, pp. 1408–1414.
- (Feb. 2009). “Formalizing Arrow’s theorem”. In: *Sādhanā* 34.1, pp. 193–220.
- (Mar. 1, 2012). *The “de Bruijn factor”*. URL: <http://www.cs.ru.nl/~freek/factor/> (visited on 03/07/2013).
- (2013). *Formalizing 100 Theorems*. URL: <http://www.cs.ru.nl/~freek/100/>.
- Woodcock, J., P. G. Larsen, J. Bicarregui, and J. Fitzgerald (Oct. 2009). “Formal method: practice and experience”. In: *ACM Comput. Surv.* 41.4, pp. 1–40.