Set Theory or Higher Order Logic to Represent Auction Concepts in Isabelle?

Marco B. Caminati\textsuperscript{1}, Manfred Kerber\textsuperscript{1}, Christoph Lange\textsuperscript{1,2}, and Colin Rowat\textsuperscript{3}

\textsuperscript{1} Computer Science, University of Birmingham, UK
\textsuperscript{2} Fraunhofer IAIS and University of Bonn, Germany
\textsuperscript{3} Economics, University of Birmingham, UK

\url{http://cs.bham.ac.uk/research/projects/formare/}

Abstract. When faced with the question of how to represent properties in a formal proof system any user has to make design decisions. We have proved three of the theorems from Maskin’s 2004 survey article on Auction Theory using the Isabelle/HOL system, and we have verified software code that implements combinatorial Vickrey auctions. A fundamental question in this was how to represent some basic concepts: since set theory is available inside Isabelle/HOL, when introducing new definitions there is often the issue of balancing the amount of set-theoretical objects and of objects expressed using entities which are more typical of higher order logic such as functions or lists. Likewise, a user has often to answer the question whether to use a constructive or a non-constructive definition. Such decisions have consequences for the proof development and the usability of the formalization. For instance, sets are usually closer to the representation that economists would use and recognize, while the other objects are closer to the extraction of computational content. We have studied the advantages and disadvantages of these approaches, and their relationship, in the concrete application setting of auction theory. In addition, we present the corresponding Isabelle library of definitions and theorems, most prominently those dealing with relations and quotients.

1 Introduction

When representing mathematics in formal proof systems, alternative foundations can be used, with two important examples being set theory (e.g., Mizar takes this approach) and higher order logic (e.g., as in Isabelle/HOL). Another dimension in the representation is the difference between classical and constructive approaches. Again, there are systems which are predominantly classical (as most first order automated theorem provers) and constructive (e.g., Coq). Isabelle/HOL is flexible enough to enable the user to take these different approaches in the same system (e.g., although it is built on higher order logic, it contains a library for set theory, \texttt{Set.thy}). For instance, participants in an auction, i.e.

* This work has been supported by EPSRC grant EP/J007498/1 and an LMS Computer Science Small Grant.
bidders, can be represented by a predicate \texttt{Bidder x} or alternatively as a list of bidders \{b1, b2, b3\}.

The difference between a classical and a constructive definition can be demonstrated by the example of the argument of the maximum for a function (which we need for determining the winner of an auction). Classically we can define it, e.g., as \texttt{arg_max f A = \{x \in A. f x = Max(f\{A\})\}}. This definition is easy to understand but, unlike a constructive one, it does not tell how to compute \texttt{arg_max}. The constructive definition is more complicated (and has to consider different cases). It corresponds to a recursive function recurring on the elements of the domain. From a programming perspective, the two kinds of definitions just illustrated (classical versus constructive) can be seen respectively as specification versus implementation.

The approaches coexist in Isabelle/HOL. For example, for a set \texttt{X} one can apply the higher order function \texttt{f} by the construction \texttt{f' X} to yield the image of the set. As a result, an author does not have to make a global decision of whether to use sets or higher order functions, but has fine grained control on what to use for a new mathematical object (e.g., sets vs. lists or lambda functions). Such a choice will typically depend on many factors. One factor is the task at hand (e.g., whether one needs to prove a theorem about an object or needs to compute its value). Another factor is naturalness of the constructions. This will typically depend on the authors and their expected audiences. Pragmatically, users also have to consider which of the possible approaches is more viable given the support provided by existing libraries for the system being used.

The ForMaRE project \cite{8} applies formal methods to economics. One of the branches of economics the project focuses on is auction theory, which deals with the problem of allocating a set of resources among a set of participants while maximizing one or more parameters (e.g., revenue, or social welfare) in the process. ForMaRE has produced the Auction Theory Toolbox (ATT), containing Isabelle code for a range of auctions and theorems about them\footnote{See \url{https://github.com/formare/auctions/}; the state at the time of this writing is archived at \url{https://github.com/formare/auctions/tree/1f1e7035da2543a0645b9c44a5276229a0aeb478}.} Therefore, there is a good opportunity to practically test the feasibility of the different approaches, as introduced above, in this concrete setting. We adopted a pragmatic attitude: we typically took a set theoretic approach, since firstly we felt most familiar with it and secondly we knew from our ongoing interaction with economists that it would look more natural to them. However, when we needed to generate code, this generally excluded set-theoretical constructions such as the set comprehension notation. As a consequence most of our work was done in the set-theoretical realm, but some done constructively which allowed us to produce the code we wanted. The disadvantage of this approach is that we had to provide supplementary ‘compatibility’ proofs to show the equivalence of the set theoretical and the computable definitions when needed. This means that, as a byproduct of our efforts, we also generated a good amount of generic set-theoretical mater-