Research Proposal: Relating Formal Theories

Ramana Kumar

March 28, 2011

Mechanically formalising mathematics is an activity with decades of history and some established tools. At a high level, mechanical formalisation means proving theorems in a formal system as implemented by a piece of software, a theorem prover or proof assistant. But there are many theorem provers, all different. Diversity has advantages, but so would effective communication between systems. I propose to research how such communication is best achieved, and to create tools for doing so.

Why relate formalisations?

Pragmatically

- To make duplicate work unnecessary.

  Theories of the same content are found duplicated in the libraries of various proof assistants. Basic definitions, of say numbers or relations, perhaps do have different natural forms in different logics. But higher-level theorems, like the fundamental theorem of algebra, shouldn't depend so much on the exact details of the underlying logic, and should only have to be proved once or twice. There are reasons to maintain multiple proofs of the same theorem (see [8]). But often someone who is familiar with one system wants to prove something that first requires some supporting formalisation, and the support only exists in another system.A translation mechanism enabling library reuse would be helpful in these situations.

- To enable collaboration.

  The more complex the mathematics, the more practical it is to formalise it in pieces. But these pieces might be done by different teams, working with different systems. The Flyspeck project [9] is one example of a substantial formalisation that is currently across three proof assistants. Aside from reducing the number of systems one needs to trust to believe a formalised result, translating between systems helps teams on the same project share work.

- Since some things are more easily expressed in certain systems.

  For example, natural formalisations of abstract algebra would arguably use dependently typed records, which are only available in some proof assistants. Also, most proof assistants are based on type theory, but sometimes the expressivity of untyped set theory is desirable. There is literature [5, 6, 13] on set theory versus type theory and the advantages of translating between them.

- Since some systems provide better tools.

Theorem provers vary on level of automation, number of users, quality of documentation, user interface, suitability for software verification versus traditional mathematics, code generation facilities, and size or contents of the library of formalised results. These, along with familiarity, are all good reasons for someone to choose one system over another, and someone else to choose differently. Good translations would reduce the cost of this choice, by making the existing library less of an issue, and by making it easier to switch systems during development.

Theoretically

- To find out how to make efficient translations.

  Interpreting one logic in another usually results in a blowup in proof/term size and a slowdown in proof checking [22]. Developing good translations between systems means finding ways to mitigate or avoid this problem. The mechanism of an efficient translation would be theoretically interesting.

- To find out how to make usable translations.

  This is another engineering problem whose solutions would be interesting. A translation accomplished by, say, embedding the source logic in the target logic, would look unnatural to a user of the target system, who would have formalised the results directly in the target logic. But it might be possible to make the equivalence of isomorphic structures transparent to the user.

- To explore a philosophical question.

  Can two formalisations (or two formal models) be about the same thing, if they are in different logics, or even in the same logic but with different definitions? What makes them formalisations of the same theorem (or models of the same system)? Whatever it is ultimately grounds any hope of a translation between them, so writing the translation can help us see it.

- To investigate language and foundations.

  Ideas used in translating between theorem provers may also be useful for making mechanical formalisations more appealing to mainstream mathematicians. Ganesalingam [4] has done work in the other direction, extracting the formal content from mathematical language, which may help.

  The needs of translation suggest using weak formal systems as container languages, perhaps in the style of Metamath [15], or like Primitive Recursive Arithmetic suggested in the QED Manifesto [1], and the interplay with stronger systems is interesting.

Existing translations between systems

There are several implementations of translations between different proof assistants, for example [7, 12, 14, 17, 19, 21, 22]. In each case, we may ask three questions:

- Are proofs translated?

  Finding a good translation between statements in different logics can be hard enough, but a complete translation will also deal with proofs. Advantages of translating proofs include: the source system does not become part of the trusted code base, and the soundness of the translator doesn't need to be checked. Naumov [16], for example, discusses this issue.

- Is the translation usable by a native of the target system?

  The results of a translation might fill a specific gap in some formalisation, but the details might make the translation too cumbersome to extend in the target system. An ideal translation would port not just precise results but also reusable definitions for continued development in another system.

- Is the translation maintained over time?

  A particular formalisation can be ported manually on a one-off basis. Sometimes a more general translator from one prover to another will be written. But it is rare for a translator to be maintained alongside new versions of the source and target proof assistants. A related issue here is coverage of the source language, in other words, restrictions on what can be translated.

We would like all three answers to be "yes". Achieving that for many pairs of systems is a good goal.

There has also been some work on "theory management", akin to package management, in particular Hurd's OpenTheory project [10, 11]. One main point here is to make the dependencies (required results) and provisions of a formal theory explicit, so that theories can be composed, sometimes with one standing in for another. The second main point is to design theories so that they fit nicely into this environment of composable theories. Theory management, especially if it is theorem prover-neutral, would make translations between systems much easier, and would shape the design of and requirements for such translations. Related ideas include a "standard library" of formalised results [23, 24], and the notion of "high-level theories" [2].

My aim is to tackle the problem of translation between proof assistants directly. Rather than implementing a separate translation for each pair of systems, I hope to find a workable "hub" language to which each system needs only one two-way link. Lessons both from existing translations and from theory management are relevant to the design of such a language.

First year goals

- Deliverable: A map of existing translations between theorem provers, indicating what they accomplish and how.

  The goal here is to see what works well out of the approaches taken so far, so that good ideas can be reused in a more comprehensive translation scheme. A secondary goal is to assess the state of interoperability, in other words, how many pairs of systems can already share results. The map would extend the second part of this research proposal, and thus is within my capacity.

- Deliverable: A design of a language for capturing the content of formal theories and emitting it back in particular logics.

  Goals for the language include the ability to capture any formal content, and the support for making isomorphic structures easily interchangeable. It is likely that this would need to be a "formalist's language", like Metamath's, in order to interface with a wide variety of logics. I would start by studying Metamath, the logical framework of Isabelle [20], and discourse representation structures as used in [4].

- Deliverable: A working prototype of the OpenTheory project tools.

  The aim is to get experience with theory engineering, and tools to experiment with theory design. A working prototype is arguably already available, or close to it, so this item should

not involve as much work as it would seem. My experience as a developer of the HOL4 proof assistant [18] will be of use here.

References

[1] Anonymous. The QED manifesto. In Alan Bundy, editor, *CADE*, volume 814 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 1994.

[2] Jacques Carette and William M. Farmer. High-level theories. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *AISC/MKM/Calculemus*, volume 5144 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2008.

[3] Ulrich Furbach and Natarajan Shankar, editors. *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.

[4] Mohan Ganesalingam. A language for mathematics. `http://people.pwf.cam.ac.uk/mg262/`, 2009.

[5] Michael J. C. Gordon. Merging HOL with set theory: preliminary experiments. Technical Report 353, University of Cambridge Computer Laboratory, 1994.

[6] Michael J. C. Gordon. Set theory, higher order logic or both? In Joakim von Wright, Jim Grundy, and John Harrison, editors, *TPHOLs*, volume 1125 of *Lecture Notes in Computer Science*, pages 191–201. Springer, 1996.

[7] Michael J. C. Gordon, James Reynolds, Warren A. Hunt Jr., and Matt Kaufmann. An integration of HOL and ACL2. In *FMCAD*, pages 153–160. IEEE Computer Society, 2006.

[8] Adam Grabowski and Christoph Schwarzweller. On duplication in mathematical repositories. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2010.

[9] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete & Computational Geometry*, 44(1):1–34, 2010.

[10] Joe Hurd. OpenTheory: Package management for higher order logic theories. In Gabriel Dos Reis and Laurent Théry, editors, *PLMMS '09: Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems*, pages 31–37. ACM, August 2009.

[11] Joe Hurd. Composable packages for higher order logic theories. In M. Aderhold, S. Autexier, and H. Mantel, editors, *Proceedings of the 6th International Verification Workshop (VERIFY 2010)*, July 2010.

[12] Alexander Krauss and Andreas Schropp. A mechanized translation from higher-order logic to set theory. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2010.

[13] Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed? *ACM Trans. Program. Lang. Syst.*, 21(3):502–526, 1999.

[14] Sean McLaughlin. An interpretation of Isabelle/HOL in HOL Light. In Furbach and Shankar [3], pages 192–204.

[15] Norman D. Megill. Metamath: A computer language for pure mathematics. `http://metamath.org`, 1997.

[16] Pavel Naumov. Importing Isabelle formal mathematics into NuPRL. Technical report, Cornell University, Ithaca, NY, USA, 1999.

[17] Pavel Naumov, Mark-Oliver Stehr, and José Meseguer. The HOL/NuPRL proof translator (a practical approach to formal interoperability). In Richard J. Boulton and Paul B. Jackson, editors, *TPHOLs*, volume 2152 of *Lecture Notes in Computer Science*, pages 329–345. Springer, 2001.

[18] Michael Norrish and Konrad Slind. HOL4 manuals. `http://hol.sourceforge.net`, 1998.

[19] Steven Obua and Sebastian Skalberg. Importing HOL into Isabelle/HOL. In Furbach and Shankar [3], pages 298–302.

[20] Larry Paulson, Tobias Nipkow, and Makarius Wenzel. Isabelle manuals. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html`, 2009.

[21] Carsten Schürmann and Mark-Oliver Stehr. An executable formalization of the HOL/Nuprl connection in the metalogical framework Twelf. In Miki Hermann and Andrei Voronkov, editors, *LPAR*, volume 4246 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2006.

[22] Freek Wiedijk. Encoding the HOL Light logic in Coq. `http://www.cs.ru.nl/~freek/notes/holl2coq.pdf`.

[23] Freek Wiedijk. Estimating the cost of a standard library for a mathematical proof checker. `http://www.cs.ru.nl/~freek/notes/mathstdlib2.pdf`.

[24] Freek Wiedijk. Selecting the domain of a standard library for a mathematical proof checker. `http://www.cs.ru.nl/~freek/notes/mathstdlib.pdf`.