Importing HOL into Isabelle/HOL

Steven Obua [⋆] and Sebastian Skalberg

Technische Universität München
D-85748 Garching, Boltzmannstr. 3, Germany

Abstract. We developed an importer from both HOL 4 and HOL-light into Isabelle/HOL. The importer works by replaying proofs within Isabelle/HOL that have been recorded in HOL 4 or HOL-light and is therefore completely safe. Concepts in the source HOL system, that is types and constants, can be mapped to concepts in Isabelle/HOL; this facilitates a true integration of imported theorems and theorems that are already available in Isabelle/HOL. The importer is part of the standard Isabelle distribution.

1 Introduction

The idea of sharing theorems between different proof-assistants is not new; there has been previous work on translating from HOL to NuPRL [1–3], from Isabelle to NuPRL [4] and from HOL to COQ [5, 6]. Only [1, 3–5] provide implementations; of these implementations only [3, 5] translate *proofs* instead of just theorems. Both implementations can deal only with a subset of the HOL inference rules and have not been used for large developments.

Our translator from HOL 4 [10] and HOL-light [11] to Isabelle/HOL [9] is therefore the first one that fulfills both of the following two criteria:

– The translation process is safe relative to the correctness of the destination system, in this case Isabelle/HOL. This is achieved by replaying *proofs* that have been recorded in the source system (HOL 4 or HOL-light) in the destination system (Isabelle/HOL).
– Large developments have been translated with our translator, in fact almost all of the entire HOL 4 distribution and all of base HOL-light.

In contrast to previous work is also that our translation is basically *between systems, not between logics.* Although some complications result from the fact that Isabelle/HOL is an object logic instance of the Isabelle framework, and HOL 4 and HOL-light directly implement higher-order logic, all these systems still share the same logic: classical simply-typed polymorphic higher-order logic (HOL). The HOL community has always profited from the fact that while the implementations of their systems has been relatively stable, a large database of proven theorems has been developed in each of the systems. Now the time has come to go a step further and to make theorems in one HOL system available in

⋆ Supported by the Ph.D. program "Logik in der Informatik" of the "Deutsche Forschungsgemeinschaft."

the other systems, thus unifying these large databases. Other work sharing this vision is that of McLaughlin who translates Isabelle/HOL to HOL-light [7].

Our translator consists of two components. First there is the proof-recording component which resides in HOL 4 and HOL-light and which records the theorems together with their proofs so that they can be saved as a collection of XML files. Second there is the importer component which takes this collection as input and uses it to reprove the exported theorems in Isabelle/HOL. The importer component can be configured to map imported concepts to concepts that are already available in Isabelle/HOL. This makes it for example possible to map the type of real numbers of HOL 4 or HOL-light to the type of real numbers of Isabelle, because they represent the same abstract concept independent of their specific construction. Another example of this versatility is that the importer can also be configured to map HOL 4's `LET` constant to Isabelle's `Let` constant although the constants differ in their names (different capitalization) and the order in which they take their arguments.

The importer component is part of the standard Isabelle distribution since 2004; the proof-recording component for HOL-light is part of the HOL-light distribution since release 2.0; the proof-recording component for HOL 4 can currently only be obtained by contacting the authors of this paper. There is also a simple importer implementation available which has been written in Java; although this implementation misses some features it can be used to check all of HOL-light; it can be downloaded from [12].

2 The Proof-Recording Component

Each HOL system is based on the central idea of an abstract datatype `thm`. Instances of `thm` can be created and manipulated only according to the rules of the logic. The part of the HOL system that implements this abstract datatype, together with theory extension mechanisms like constant and type definition, is called the *kernel* of the system. The rest of the system is built on top of it.

Thanks to the concept of a kernel, adding proof-recording to an HOL system is relatively easy: first one adds a new component `proof` to the internal representation of `thm`, then one modifies the functions that manipulate `thm` to record these manipulations in the new component. Basically, each constructor of the `proof` component corresponds to an inference rule of the kernel. In HOL 4, which is implemented in Standard ML, these changes were transparent to the rest of the system; in HOL-light, whose implementation language is OCaml, unexpected problems arose from the fact that the built-in equality on theorems leaked through the abstractness of the datatype; this equality had changed, of course, because now for two theorems to be equal they have to have the same proof, too! Therefore all places in the HOL-light system had to be found and modified that made use of equality on theorems.

A `proof` can be considered a tree consisting of other `proof`s, `term`s and `type`s. Saving this tree to disk naively is not feasible in practice; it is simply just to big. This might come as a surprise: after all, the proof has to fit into the main memory of the computer! The solution to this puzzle is that a proof in main memory is not a tree, but a DAG; unfolding this DAG into a tree when saving can lead to exponential blow-up. Therefore we go through the following steps during saving a collection of proofs:

- Apply $\alpha\beta\eta$-normalization to all terms; therefore we need no proof constructors for the β and η inference rules, which degenerate to reflexivity.
- Simplify proofs that involve reflexivity; the proof `TRANS (REFL t) p` for example can be simplified to just `p`.
- Identify all proofs that are shared in main memory; each shared proof is saved into a separate XML file. When saving a proof A that has a shared proof B as its child, instead of B only a link to the XML file of B is saved in the XML file of A.
- For each proof that is saved into a separate XML file, share all the terms and types within that proof, using a DAG representation.

These simple measures yield manageable proof-on-disk sizes: The base HOL 4 distribution results in 80,000 files, taking up 350MB disk space (13MB when gzipped). The base HOL-light system results in 130,000 files, taking up 229MB disk space (21MB when gzipped).

One extreme case is the proof of the Jordan Curve theorem in HOL-light by Thomas Hales. It produces about 1,000,000 files; unix commands like `ls` broke down when used naively. Therefore it would be better not to shift the proof sharing to the file system, but design an own file format for this purpose. This format could also deal with gzip-like compression issues.

Note that we use a first-order representation of proofs with sharing as our compression technique; another approach can be found in [8] where proofs are represented as higher-order terms. It is not clear which approach is superior, or whether a combination of both approaches would be beneficial.

Adding proof-recording to the kernel of an HOL system does not change the runtime of this HOL system significantly. Saving the recorded proofs to disk is a time-consuming task, though: the base HOL 4 system needs 50 minutes, the base HOL-light system about 30 minutes. Saving the Jordan Curve theorem took a couple of hours.

The semantics of the proof constructors establishes sort of an abstract HOL kernel [1] which is the union of the HOL 4 and HOL-light kernel, but operates not only modulo α-, but also modulo β- and η-equivalence. One inefficiency of this kernel is its current storage format, as mentioned before. Another improvement of it would be to incorporate higher inference rules such as rewriting which would certainly reduce the size of proofs considerably. Taking this one step further, a "golden" kernel should also provide the possibility of *coding higher rules and storing this code along with the proof objects.* Such a kernel could then serve as a standard kernel for exchanging theorems between HOL systems, and find also its applications in areas such as proof-carrying code, where the size of proof objects is regarded an important factor.

[1] for a description of this kernel, see [12]

3 The Importer Component

The import of the recorded proofs is done in two phases:

1. A *configuration Isabelle theory* transforms a set of XML proof files into a set of Isabelle/HOL theories.
2. The generated Isabelle/HOL theories can now be used just as other Isabelle theories; because the statements in these theories are proven with the help of the recorded proofs, the XML proof files must still be present when using these generated theories.

The critical phase is the first one; here it is decided via the configuration file how the imported concepts are mapped onto already existing ones. The importer does not accept axioms; this means that all constants that are specified via axioms in the source HOL system must be mapped onto existing Isabelle constants, and statements must be proven in Isabelle about these existing constants that correspond to the imported axioms. Further mappings are desirable for a better integration of the imported theories with already existing theories, but these additional mappings are not required.

There are mainly three mapping constructs: the command *const-maps* which maps constants, the command *type-maps* which maps types, and the attribute *hol4rew* which can be attached to Isabelle theorems. Furthermore there is the additional command *ignore-thms* which is currently also essential for the mapping process; its purpose is to ignore certain theorems and to not import them. Unfortunately one currently has to make explicit use of *ignore-thms* when mapping constants or types: the defining theorems/proofs for these constants and types have to be ignored. This is bound to confuse particularly the novice user, and is still annoying also the experienced one.

The importing process can be described as follows:

– A list is fetched from the collection of XML proof files that describes all *named* theorems in that collection; theorems that are not named occur also among the proof files, they have been created because of proof sharing.
– All entries of the list are imported one after another. If the entry corresponds to a theorem flagged by *ignore-thms*, it is skipped. Otherwise first the statement S of the theorem is fetched from the file. Then the types and constants of that statement are mapped according to *const-maps* and *type-maps*, yielding a statement $S' = \mathrm{map}\,(S)$.
– After this, the *shuffler* is applied to S', yielding a statement $S'' = \mathrm{shuffle}\,(S')$. The shuffler makes a statement more "Isabelle-like", converting for example quantified variables into unquantified schematic variables. The shuffler also applies all rewriting rules to the statement that have been defined via the *hol4rew* attribute.
– If S'' can be looked up in the Isabelle theorem database, then the import of the theorem has been successful. For mapped constants, this is very often the case. Otherwise the proof P is fetched from the XML proof file; mapping yields $P' = \mathrm{map}\,(P)$. Then P' is replayed in Isabelle, mimicking the inference

rules of the abstract HOL kernel, yielding a theorem T. The theorem T is stored so that other proofs referencing this proof can access it. Furthermore $T' = \text{shuffle}\,(T)$ is stored and dumped to the generated theories.
– Thus, all shared proofs are replayed at most once. While this is desirable for obvious performance reasons, there is also a functional aspect to it. Certain proofs do have *side-effects*, because definition of constants and types are also encoded as proofs. The side-effect of replaying a constant definition is that this constant is now defined in the generated Isabelle theory; the same holds for type definitions. Replaying a proof at most once ensures that a side-effect is executed at most once.

4 Conclusion

We have described an importer/translator from HOL 4 and HOL-light to Isabelle/HOL; other source systems can be supported given that they adhere to the contract of the abstract kernel. The translator has been used to import large developments like the real analysis libraries of HOL 4 and base HOL-light and facilitates an integration of imported and already existing theories; thus it sets new standards for safe interoperability between HOL systems, though its user-friendliness could be improved.
Acknowledgment. Thanks to John Harrison for including the proof-recorder in his HOL-light distribution and for putting time and effort into making that inclusion as smooth as possible; also thanks to Virgile Prevosto for OCaml-related help.

References

1. D. J. Howe. Importing Mathematics from HOL into Nuprl. *TPHOLs'96*, LNCS 1125, Springer 1996.
2. C. Schürmann, M. Stehr. An Executable Formalization of the HOL/Nuprl Connection in the Metalogical Framework Twelf. *LPAR 2004*, to appear.
3. P. Naumov, M. Stehr, J. Meseguer. The HOL/NuPRL Proof Translator: A Practical Approach to Formal Interoperability. *TPHOLs'01*, LNCS 2152, Springer 2001.
4. P. Naumov. Importing Isabelle Formal Mathematics into NuPRL. *TPHOLs'99*, LNCS 1690, Springer 1999.
5. E. Denney. A Prototype Proof Translator from HOL to Coq.*TPHOLs'00*, LNCS 1869, Springer 2000.
6. F. Wiedijk. Encoding the HOL Light logic in Coq. Unpublished notes.
7. S. McLaughlin. An interpretation of Isabelle/HOL in HOL Light, submitted.
8. S. Berghofer, T. Nipkow. Proof terms for simply typed higher order logic.*TPHOLs'00*, LNCS 1869, Springer 2000.
9. T. Nipkow, L. C. Paulson, M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, Springer 2002
10. The HOL System Description. `http://hol.sourceforge.net`
11. J. Harrison. The HOL Light manual. `http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.pdf`
12. S. Obua. `http://www4.in.tum.de/~obua/importer`