PRocH: Proof Reconstruction for HOL Light

Cezary Kaliszyk and Josef Urban

[1] University of Innsbruck, Austria
[2] Radboud University Nijmegen

Abstract. PRocH[3] is a proof reconstruction tool that imports in HOL Light proofs produced by ATPs on the recently developed translation of HOL Light and Flyspeck problems to ATP formats. PRocH combines several reconstruction methods in parallel, but the core improvement over previous methods is obtained by re-playing in the HOL logic the detailed inference steps recorded in the ATP (TPTP) proofs, using several internal HOL Light inference methods. These methods range from fast variable matching and more involved rewriting, to full first-order theorem proving using the `MESON` tactic. The system is described and its performance is evaluated here on a large set of Flyspeck problems.

1 Introduction, Motivation, and Related Work

Independent verification of proofs found by Automated Theorem Provers (ATPs) is not an uncommon topic in automated reasoning research. Systems like IVY [4] rely on very detailed proof output from ATPs (Otter, Prover9), which is then independently replayed and checked by a trusted system (ACL2). This technique has been used several times, e.g., for replaying the `MESON` and Otter/Prover9 proofs in HOL Light, replaying the Otter/Prover9 proofs in Mizar, and replaying the Metis proofs in HOL and Isabelle. The GDV [7] tool is even parametric: any ATP system understanding TPTP can be used for independent verification. The Metis/Isabelle combination has been used also as a part of the Sledgehammer [5] tool that uses arbitrary ATPs to discharge Isabelle proof obligations. If an ATP proof is found, and Metis can reconstruct the proof $NeededLemmas \vdash Conjecture$, then `metis(NeededLemmas)` is a valid Isabelle tactic, that is in practice used as an ATP proof importer. In HOL Light, `MESON` can be used in the same way for importing the proofs found by ATPs on FOL problems produced by the recently developed "HOL(y)Hammer" (HH) tool [3].

However, Metis and `MESON` are on average weaker than state-of-the art ATPs like Vampire and E. As ATPs and premise-selection tools get stronger, the ability of Metis and `MESON` to reconstruct (in short time usable in large ITP libraries) the ATP proof just from the proof premises decreases. Also, proofs in ITP systems like Isabelle, HOL Light and Mizar should (eventually) strive for human readability. Even if the strength of Metis and `MESON` grew, a single call to them would get hard to understand, requiring further explanation. These reasons motivate our work on a general tool that reconstructs TPTP proofs in HOL Light, using not just the proof premises, but also the steps recorded in the TPTP proof format.

[3] Proch (pronounced as "prokh") means "dust/powder" in Polish. The proof is (also) reconstructed from fine-grained inference dust.

2 Using Existing Approaches on HOL Light Problems

In total, the experiments with ATP-proving of HOL Light and Flyspeck theorems described in [3] have produced 7247 proofs when using Vampire, E (run under the Epar [9] scheduler) and Z3, sometimes with high timelimits (900s). Only Vampire and E produce full TPTP proofs, but Z3 also prints the necessary premises (unsat core). These proofs are pseudo/cross-minimized, i.e., each proof was re-run by all ATPs using the proof premises only, while the number of proof premises was decreasing. Using the resulting sets of premises, Epar can find 6318 proofs in 30s. This set is used for further evaluations here. Table 1 shows the performance of the potential proof-importing tools, i.e., `MESON`, Metis, and Prover9 run with 300s time limit. Note that (unlike `MESON`) both Metis and Prover9 are just run externally, i.e., not reconstructing a valid HOL proof. As mentioned above,

Table 1: `MESON`, Metis, and Prover9 with 300s on the 6318 Epar proofs

method	`MESON`	Metis	Prover9
replayed	5255	4595	4672
replayed (%)	83.1	72.7	73.9

low proof times are important for working with large ITP libraries containing (tens of) thousands of theorems, each typically proved using several of the low-level (`MESON`, Metis, Mizar "by", etc.) "atomic" calls. Table 2 therefore shows the performance of the above methods when using only 1 second for reconstruction.

Table 2: `MESON`, Metis, and Prover9 with 1s on the 6318 Epar proofs.

method	`MESON`	Metis	Prover9
replayed	5014	2803	4111
replayed (%)	79.3	44.3	65.0

Particularly the numbers obtained for Metis are considerably worse than the numbers obtained so far with Metis-based proof reconstruction in Sledgehammer [1], where only 10% of ATP proofs are lost by Metis. One possible reason is that Metis has been well-integrated with Sledgehammer, e.g., by using customized Sledgehammer-generated term orderings. Another part of explanation could be that the proofs found by HH on Flyspeck are on average harder than the proofs found by Sledgehammer on the Judgement Day benchmark. The reasons can be that the Judgement Day benchmark consists of goals that are on average easier, the HH premise selection might be more precise (allowing more involved ATP proofs), and also ATP systems like Vampire and E have been strengthened since the time of the Judgement Day evaluation.

3 PRocH System Description

The HH tool runs in parallel several (now 14) AI/ATP combinations on a given HOL problem, and if a proof is found, it is pseudo/cross-minimized by further parallel running of the ATPs (and their strategies). PRocH then follows by trying in parallel several (old and new) proof reconstruction methods. Unlike the above methods that can only use the ATP proof premises to find their own detailed proof, the most complicated of the PRocH's methods (`hh_recon`) also tries to reconstruct in HOL Light the TPTP proofs created by the ATP systems. In this its closest relative is the `isar_proof` Sledgehammer function described in [5], from which it probably differs by complete reliance on type annotations. In some sense, PRocH's `hh_recon` is so far less ambitious than `isar_proof`, because it does not yet attempt to write a HOL proof script. This also allows to treat some constructs (e.g., higher-order application) differently from `isar_proof` during the reconstruction. PRocH's use is now similar to HOL's `MESON` tactic, i.e., a call to `hh_recon[HOLPremises]` will try to justify a given HOL conjecture by going through the following stages (described more in the following subsections):

1. *Translation to FOL:* A HOL Light problem in the form $HOLPremises \vdash HOLConjecture$ is translated to an untyped FOF TPTP problem, where part of the FOF encoding of terms are annotations encoding their HOL type.
2. *Running ATPs:* An external ATP is run on the first-order problem producing a TPTP proof.
3. *Parsing:* The untyped FOF and CNF formulas in the TPTP proof are parsed back into typed HOL terms (making use of the encoded type annotations). This part also has to handle skolemization.
4. *Replaying:* The justification structure of the TPTP proof is replayed on the parsed HOL Light terms, resulting in a valid HOL Light proof.

3.1 Translation to FOL and Producing FOL Proofs

The translation to FOL is described in [3], but we show a brief example here. The translation has to encode higher-order features like lambda abstraction, currying, quantification over function variables and their application. As a leading example, consider the following higher-order theorem `FORALL_ALL`[4]

$$\forall \texttt{P l. } (\forall \texttt{x. ALL (P x) l}) \iff \texttt{ALL } (\lambda \texttt{s. } \forall \texttt{x. P x s}) \texttt{ l}$$

saying that each `x`-image of a binary relation (predicate) `P(x,y)` contains (is true for) all elements of a list `l` iff for all elements `s` of `l` the unary predicate "`P(x,s) is true for all x`" is true. To express this in FOL, first the lambda function is lifted from the context (its definition is created and used as an antecedent) and the higher-order applications are made explicit as follows:

$$\begin{aligned}\forall \texttt{l P F. } &(\forall \texttt{s. happ F s} \iff (\forall \texttt{x. happ (happ P x) s})) \\ &\implies ((\forall \texttt{x. ALL (happ P x) l}) \iff \texttt{ALL F l})\end{aligned}$$

[4] `http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/lists.html#FORALL_ALL`

The formulas before and after the lambda-lifting and `happ`-introduction conversions are logically equivalent in the HOL logic,[5] and such conversions are used to change the initial HOL proof state into a form $ConvHOLPremises \vdash ConvHOLConjecture$ that will later correspond to the formulas reconstructed from the ATP proof. After these conversions, the implicit polymorphic HOL type domains (`A,B`) are explicitly introduced as variables and quantified over, and type annotations are added for all HOL terms using the `s` and `p` wrappers (and `happ` is shortened to `i`), resulting in the following FOF formula:

```
![A,B,L,P,F]:
 (![S]:(p(s(bool,i(s(fun(B,bool),F),s(B,S))))
   <=> ![X]:p(s(bool,i(s(fun(B,bool),i(s(fun(A,fun(B,bool)),P),s(A,X))),
                       s(B,S)))))
 =>
 (![X]:p(s(bool,all(s(fun(B,bool),i(s(fun(A,fun(B,bool)),P),s(A,X))),
                    s(list(B),L))))
   <=> p(s(bool,all(s(fun(B,bool),F),s(list(B),L))))))
```

This way the HOL problem $ConvHOLPremises \vdash ConvHOLConjecture$ is translated to an untyped FOF TPTP problem $FOLPremises \vdash FOLConjecture$, on which ATPs like E and Vampire are run, producing derivations in the TPTP format [8]. Unlike the fixed and very detailed Otter/Prover9 IVY format, the TPTP proof steps may be justified by arbitrary inference method, and thus may in theory be arbitrarily hard. In practice, for E and Vampire the (overwhelming number of) proof steps are detailed and easy to check with weak ATPs. Several interesting steps from E's proof of `FORALL_ALL` are as follows:

```
fof(2,axiom,p(s(bool,t)), file('f1', aTRUTH)).
fof(4,axiom, (~(p(s(bool,f)))<=>p(s(bool,t))), file('f1', aBOOL_CASES_AX)).
fof(5,conjecture, (![A,B,L,P,F]: ... ), file('f1', cFORALL_ALL)).
fof(6,negated_conjecture, ~(![A,B,L,P,F]: ... ),
    inference(assume_negation,[status(cth)],[5])).
...
fof(25,negated_conjecture,?[X10]:?[X11]:?[X12]:?[X13]:?[X14]: ...,
    inference(variable_rename,[status(thm)],[24])).
fof(26,negated_conjecture,![X15]:(~(p(s(bool,i(s(fun(esk3_0,bool),esk6_0)...,
    inference(skolemize,[status(esa)],[25])).
...
cnf(33,plain,(~p(s(bool,f))),inference(cn,[status(thm)],[32,theory(equality)])).
...
cnf(6393,negated_conjecture,(p(s(bool,f))),
    inference(spm,[status(thm)],[6320,5512,theory(equality)])).
cnf(6404,negated_conjecture,($false),
    inference(sr,[status(thm)],[6393,33,theory(equality)])).
cnf(6405,negated_conjecture,($false),6404,['proof']).
```

[5] For HOL speakers, e.g., the `happ` functor is just the HOL identity, i.e., we use:

```
happ_def = new_definition `(happ : ((A -> B) -> A -> B)) = I`;;
happ_conv_th = prove (`!(f:A->B) x. f x = happ f x`, ... );;
happ_conv = REWR_CONV happ_conv_th;;
```

Such TPTP proofs produced by ATPs on the type-annotated input are the starting point for the HOL proof reconstruction. This is done in two stages: reconstruction of HOL terms/formulas, and reconstruction of the justification structure (HOL proof).

3.2 Reconstructing Terms and Formulas

The TPTP proof format is first parsed into suitable ML data structures using a lexer/parser combination created with `ocamllex/ocamlyacc`. For terms/formulas, parts of the HOL Light parsing mechanisms are re-used. In particular we gradually construct the intermediate HOL Light `preterm` structure, on whose final form the HOL Light `retypeckeck` function is called to obtain a HOL term. The preterm is constructed using variable/constant constructors (`Varp`), binary applications (`Combp`), abstractions (`Absp`), and type annotations (`Typing`).

Initially, the preterm just mirrors the FOL term structure, and in several passes the HOL structure is recovered from the type annotations. The recovery process might fail if the ATPs did proof-relevant operations that break the type annotation, however, at least with the resolution/paramodulation inferences done by E this practically does not happen. The first step is discovery of HOL type variables in formulas. For every type annotation `s(type, term)` all variables (and skolem constants) that appear in the left argument are considered to be type variables. In the next step, quantifications over such type variables are removed (they are implicitly universal in the HOL logic). During skolemization, type variables might have become arguments to newly introduced skolem functors. Such type arguments are removed, they are implicit in the HOL logic.

After that the `s` and `p` annotations are changed into `Typing` constructors with the appropriate types, and HOL Light's `retypeckeck` is called on the transformed preterm to obtain a HOL term.

3.3 Replaying ATP Proofs in HOL Light

As mentioned above, the problem $HOLPremises \vdash HOLConjecture$ is in HOL Light first converted (packaging the conversions in `HH_TAC`) to the equivalent $ConvHOLPremises \vdash ConvHOLConjecture$ problem. This problem (proof state) is then further transformed using the HOL formulas reconstructed from the ATP proof, and using mechanisms implemented by the HOL Light subgoal package to handle the ATP proof steps. Given the topologically sorted list of proof steps, for every proof step a HOL tactic is applied, depending on the type of the step. Axioms are looked up among the HOL goal assumptions (using their name) and proved using these assumptions. The negated conjecture is introduced by transforming the goal using HOL's `REFUTE_TAC` ("R") (proof by contradiction). Skolemization steps are justified using HOL's `CHOOSE_TAC` ("C"), and for plain inference steps (SZS status THM) we gradually try three increasingly complex methods: matching (`MATCH_ACCEPT_TAC` - "m"), rewriting (`REWRITE_TAC` - "r"), and HOL's full first-order ATP (`MESON_TAC` - "1" or "2" depending on the number of premises). The final contradiction concludes the HOL proof using HOL's `ACCEPT_TAC` ("A"). For the reconstruction of the proof of `FORALL_ALL`, the sequence of this steps is as follows: `mR1rrC1111111mC1111122222221222A`.

4 Evaluation

Table 4 shows 1s evaluation of the reconstruction methods tried on the 6318 Epar proofs. The methods (tactics) are described in Table 3.[6] PRocH tries (after the HH conversion) three methods in parallel, i.e., its total CPU time can be 3s. This is not very significant for the comparison, see the 300s results of MESON and Prover9 in Table 1. The methods in Table 4 are ordered from top to bottom by a greedy covering sequence (the last column), where the next method always adds most to the previous methods. The unique number of solutions, SOTAC and Σ−SOTAC [3] are metrics that show the usefulness in the whole population.

Table 3: Names and descriptions of the tactics tried for proof reconstruction.

Method	Description
PRocH	HH conversion, then parallel replay with HH_RECON, MESON, and Prover9.
MESON	Standard MESON_TAC conversion then MESON and its replay.
SIMP	SIMP_TAC: Simplification by repeated conditional contextual rewriting.
Prover9	Standard Prover9 conversion then Prover9 and its proof replay.
REWRITE	REWRITE_TAC: goal simplification by repeated unconditional rewriting.
INT_ARITH	Basic algebra and linear arithmetic over the integers.
COMPLEX_FIELD	Basic "field" facts over the complex numbers.

Table 4: Performance of reconstruction tactics run in 1s on 6318 Epar proofs.

Prover	Theorem (%)	Unique	SOTAC	Σ−SOTAC	Greedy (%)
PRocH	5687 (90.0)	418	0.404	2298.50	5687 (90.0)
MESON	5014 (79.3)	118	0.367	1839.30	5862 (92.7)
SIMP	2384 (37.7)	54	0.290	692.30	5968 (94.4)
INT_ARITH	407 (6.4)	4	0.236	95.95	5972 (94.5)
REWRITE	1540 (24.3)	3	0.249	382.87	5975 (94.5)
COMPLEX_FIELD	84 (1.3)	2	0.270	22.68	5977 (94.6)
Prover9	2208 (34.9)	1	0.293	646.40	5978 (94.6)

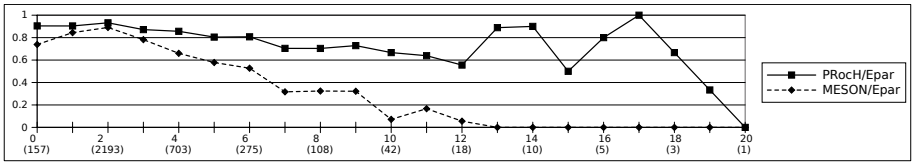
The performance of the three submethods used by PRocH are shown in Table 5. They are again ordered by their greedy covering sequence. The HH preprocessing significantly improves the Prover9-based replay, but more important for the overall performance gain is the large number (406, i.e., 6.4% of 6318) of unique solutions contributed by HH_RECON. Finally, the performance of PRocH and MESON is compared in Figure 1 depending on the count of premises in the reconstructed proof. As the number of premises goes up (ATP proofs get more involved), PRocH becomes more and more necessary.

Table 5: Performance of the three submethods used by PRocH.

Prover	Theorem (%)	Unique	SOTAC	Σ−SOTAC	Greedy (%)
HH + Prover9	4737 (74.9)	253	0.412	1954.00	4737 (74.9)
HH + HH_RECON	4299 (68.0)	406	0.421	1811.50	5499 (87.0)
HH + MESON	4737 (74.9)	188	0.406	1921.50	5687 (90.0)

[6] We tried more tactics, but they did not find more solutions. Higher times help very little, see: http://cl-informatik.uibk.ac.at/users/cek/recon_stats.html

Fig. 1: PRocH and MESON dependence on premise nr. (Epar proof nr. in brackets).



5 Conclusion and Future Work

96.4% of the 6318 Epar proofs are reconstructed in 300s by some of the methods. To a great extent this validates the AI/ATP proof methods developed in [3], which were so far only verified by manual checking that the most striking (much shorter) AI/ATP proofs are really valid. 94.4% of the 6318 Epar proofs are reconstructed in 1s by one of the five (sub)methods run in parallel, i.e., the top three methods from Table 4, where PRocH consists of the three parallel submethods from Table 5. This makes the replay of more involved proofs fast and practical.

It would be good to postprocess the verbose ATP proofs into more compact, structured [11], and human-readable proofs that would be stored directly as HOL Light code. Running proof-shortening tools in a loop is a simple method that already helps a lot, e.g., when importing Otter/Prover9 proofs into Mizar. Tools for lemma and concept introduction [6,10] can be experimented with, and with stronger AI/ATP assistance are becoming more and more important.

References

1. S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.
2. S. Hetzl, A. Leitsch, and D. Weller. Towards algorithmic cut-introduction. In N. Bjørner and A. Voronkov, editors, *LPAR*, volume 7180 of *LNCS*, pages 228–242. Springer, 2012.
3. C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *CoRR*, abs/1211.7012, 2012.
4. W. McCune and O. S. Matlin. Ivy: A Preprocessor and Proof Checker for First-Order Logic. In *Computer-Aided Reasoning: ACL2 Case Studies*, number 4 in Advances in Formal Methods, pages 265–282. Kluwer, 2000.
5. L. C. Paulson and K. W. Susanto. Source-level proof reconstruction for interactive theorem proving. In *TPHOLs*, pages 232–245, 2007.
6. K. Pąk. Methods of lemma extraction in natural deduction proofs. *Journal of Automated Reasoning*, 50:217–228, 2013.
7. G. Sutcliffe. Semantic derivation verification: Techniques and implementation. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.
8. G. Sutcliffe, S. Schulz, K. Claessen, and A. V. Gelder. Using the TPTP language for writing derivations and finite interpretations. In *IJCAR*, pages 67–81, 2006.
9. J. Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2013.
10. J. Vyskočil, D. Stanovský, and J. Urban. Automated Proof Compression by Invention of New Definitions. In E. M. Clarke and A. Voronkov, editors, *LPAR (Dakar)*, volume 6355 of *LNCS*, pages 447–462. Springer, 2010.
11. F. Wiedijk. A synthesis of the procedural and declarative styles of interactive theorem proving. *Logical Methods in Computer Science*, 8(1), 2012.