Exploring Metamath Proof Structures

Christoph Wernhard[1] and Zsolt Zombori[2,3]

[1] University of Potsdam, Germany info@christophwernhard.com
[2] Alfréd Rényi Institute of Mathematics, Hungary zombori@renyi.hu
[3] Eötvös Loránd University, Budapest, Hungary

Introduction. Our work focuses on proofs as structured objects and we explore different ways in which proofs can be structured. Proof structuring plays a crucial role both in human and automated mathematics. Human proofs are highly structured via the extensive use of lemmas, typically resulting in short proofs because a lemma hides its subproof, which would possibly have even multiple occurrences in the expanded overall proof. The basic principle underlying many "non-resolution" ATP techniques [11, 1, 2, 7, 16, 15] is enumerating proof structures in combination with formula unification. Structure-based techniques have great influence on their capabilities for proof search. We are interested in contrasting human-formalized mathematical proofs, as exemplified by the *set.mm* database of *Metamath* [9], with those structured via automated methods, exemplified by the TreeRePair [8] algorithm.

The *Metamath* language with the so-called *Metamath Proof Explorer*, that is, the database *set.mm* of more than 40.000 proven theorems [9], provides our source of human-formalized mathematics. *Simple by design*, all *Metamath* proofs are ultimately built up from only two primitive proof constructors: Meredith's condensed detachment [12, 18], D with two arguments, representing modus ponens with unification applied to major and minor premise, and G with a single argument, which represents embedding the premise proven by the argument within a universal quantifier upon a fresh individual variable [10]. Within *set.mm* these appear as axioms `ax-mp`[1] and `ax-gen`. This very same proof structure format (so far just with the D constructor) underlies our previous work on analyzing proofs, eliminating redundancies in proofs, proof search, and lemma learning, also with the practical environment *CD Tools*[2] embedded in *SWI-Prolog* [17, 18, 16, 15, 13, 14].

TreeRePair [8] is an adaptation of the *RePair* string compression algorithm to trees, originally designed for XML documents. It is an eager algorithm aiming to produce maximally compressed tree descriptions. It represents the compressed tree by a tree grammar where nonterminals can have parameters such that not only duplicate subtrees but also duplicate "tree patterns" are factorized. For example, the pattern $\mathsf{f}(\mathsf{a}, \_)$ in the tree $\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{b}),\mathsf{f}(\mathsf{a},\mathsf{c}))$, which can be compressed to $\{\mathit{Start} \to \mathsf{f}(A(\mathsf{b}),\mathsf{f}(A(\mathsf{c}))),\ A(x) \to \mathsf{f}(\mathsf{a},x)\}$. The correspondence between tree grammars and proof structures was observed in [15] and realized there with the implementation of [8].[3] For this project, we created our own adapted Prolog implementation of TreeRePair.

The *set.mm Metamath* database provides a large set of proof trees, one for each theorem, that can be viewed both as human structured (using lemmas) and as fully expanded (with the proof trees of lemmas inserted). We try to understand ways in which human proof structuring differs from automated methods, hoping that this will contribute to improving search in ATP through knowledge transfer from human formalizations, potentially relying on machine learning.

[1] With argument order compared to D reversed.
[2] http://cs.christophwernhard.com/cdtools/.
[3] Lemma synthesis by *cut introduction* on the basis of resolution proofs, e.g., [19, 5], is another ATP-based approach to proof compression, where also tree grammar compression is considered [6]. Tree compression is applied there to *formulas*, while we apply it to proof structures. Relating and comparing the approaches remains future work.

Prolog Representation of *Metamath* Databases and Proof Terms. Since Prolog is our basic implementation language for symbolic computations, we translate a *Metamath* database to a set of Prolog facts. This *Metamath* interface was implemented from scratch in *SWI-Prolog* and is integrated in *CD Tools*.[4] Our translation yields the same first-order formulas as the *Metamath* hammer tool [3].[5] Verification so far works throughout *set.mm*, with three exceptions.[6] The Prolog fact base is generated in about 2 minutes from *set.mm*. Once generated and pre-compiled by *SWI-Prolog*, it can be loaded in half a second. We also translate proofs to Prolog terms, where the space saving by *Metamath*'s proof compression [10, App. B] is preserved through factorized terms. Based on this generic and lossless Prolog representation of *Metamath* proofs, we support various conversions. For our experiments, we removed the *variable-type* or `$f` hypotheses and represent proofs as in the following example.

```
mpd(X,Y) -> 'ax-mp'(X,a2i(Y))                    (1)
```

If we replace `ax-mp` with `d` (corresponding to Meredith's D), the example becomes

```
mpd(X,Y) -> d(a2i(Y),X)                          (2)
```

We may read this as a rewrite rule that rewrites a lemma application `mpd(X,Y)` in a proof term. Exhaustive rewriting yields a proof term with only axiom designators as constants and only `d`, `g` (or `ax-mp`, `ax-gen`) as function symbols. For our example, we then obtain – depending on our preference of `ax-mp` or D – either

```
mpd(X,Y) -> 'ax-mp'(X,'ax-mp'(Y,'ax-2'))    or   (3)
mpd(X,Y) -> d(d('ax-2',Y),X)                     (4)
```

A particular formula can be associated with such a proof term: the most general theorem proven by it. It is a Horn clause with a body atom for each variable (`X` and `Y` in our example). Based on the definition of `ax-2` we obtain

$$\mathsf{P}(x \Rightarrow y) \leftarrow \mathsf{P}(x \Rightarrow z) \wedge \mathsf{P}(x \Rightarrow (z \Rightarrow y)) \tag{5}$$

Here P (suggesting "provable") is the predicate to express condensed detachment on object-level formulas with first-order logic at the meta-level. The function symbol $\Rightarrow$ represents implication on the object level (in *set.mm* this is `wi`, displayed as `->`). As a λ-term, the (expanded) definition of mpd can be read as $\mathsf{mpd} \stackrel{\text{def}}{=} \lambda x.\lambda y.\mathsf{ax\text{-}2}yx$, and as a variable-free combinator-term [4] this would be $\mathsf{mpd} \stackrel{\text{def}}{=} \mathbf{C}\mathsf{ax\text{-}2}$. Besides rewriting a given poof, proof term definitions such as (1)–(4) can be used to control proof search by restricting proof term enumeration to terms constructed from a given set of left-hand sides, considered as *proof schemas* [15]. At preprocessing, the most general theorems of the corresponding right-hand sides are determined. During the actual proof search – structure enumeration in combination with formula unification – these theorem formulas participate in the unification. Proof term definitions can also be exploited by any prover by just adding their most general theorems as additional axioms.

Proofs Created by Humans vs. Extracted via TreeRePair. We experimented with taking a large set of fully expanded proofs from *set.mm* and applying TreeRePair to compute a structuring that corresponds to lemma definitions in the form of (3) or (4). Comparing the found lemmas with those present in *Metamath* allows to address interesting questions such as

[4] http://cs.christophwernhard.com/cdtools/index.html#metamath.

[5] Differently from the *Metamath* hammer tool, it generates first-order representations directly, not via a higher-order representation. For more information see Web documents at the URL referenced in footnote 4.

[6] Equality, element relationship, and theorem `bj-0` stating well-formedness, seem to require special treatment.

- To what degree can the selection of the human-chosen lemmas in *Metamath* be explained as based on their strong compressing effect?
- Can TreeRePair find lemmas not in *Metamath* that provide a good compressing effect and seem useful to humans?
- Variations of ATP calculi – for example binary resolution or hyperresolution – can be modeled by means of proof structure lemmas [15]. Can these be identified among the human-provided lemmas? Can these be found via TreeRePair?
- Can we identify abstract categories of lemmas, e.g., general inference rule or specific for certain theorems, based on compressing effects and occurrences in given sets of proofs?
- Do mechanically observed redundancies in human-made proofs (e.g., failure of S- and C-regularity [18]) have a beneficial purpose?

The human-created *Metamath* proofs are typically quite short, due to the intensive use of lemmas. Fully expanding all their proofs, however, yields really large binary trees (with `d` and `g` as function symbols),[7] resulting in scalability issues. We address this by an iterative process, where we start with running TreeRePair on the fully expanded proofs of only a small subset of problems. In the resulting lemma set, we mark those that are existing lemmas in *set.mm* (i.e., have the same proof structure) as *rediscovered*. We move on to the next iteration, selecting a larger set of problems, but this time the rediscovered lemmas are not expanded in the proof trees. With this approach, we can process the first 1500 theorems of *set.mm* in about 1 minute. We obtain 766 lemmas, of which 715 (48% of the processed theorems) are rediscovered and 46 are novel (see Appendix A).[8]

Acknowledgments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495, by the North-German Supercomputing Alliance (HLRN), by the ERC grant CoG ARTIST 101002685, by the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008, the Hungarian Artificial Intelligence National Laboratory Program (RRF-2.3.1-21-2022-00004), the ELTE TKP 2021-NKTA-62 funding scheme and the COST action CA20111.

References

[1] Bibel, W.: Automated Theorem Proving. Vieweg (1987), first edition 1982

[2] Bibel, W., Otten, J.: From Schütte's formal systems to modern automated deduction. In: Kahle, R., Rathjen, M. (eds.) The Legacy of Kurt Schütte, chap. 13, pp. 215–249. Springer (2020). https://doi.org/10.1007/978-3-030-49424-7_13

[3] Carneiro, M., Brown, C.E., Urban, J.: Automated theorem proving for Metamath. In: Naumowicz, A., Thiemann, R. (eds.) ITP 2023. LIPIcs, vol. 268, pp. 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ITP.2023.9

[4] Curry, H., Feys, R.: Combinatory Logic, vol. I. North-Holland (1958)

[5] Ebner, G., Hetzl, S., Leitsch, A., Reis, G., Weller, D.: On the generation of quantified lemmas. J. Autom. Reasoning **63**(1), 95–126 (2019). https://doi.org/10.1007/s10817-018-9462-8

[6] Hetzl, S.: Applying tree languages in proof theory. In: Dediu, A.H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 301–312 (2012). https://doi.org/10.1007/978-3-642-28332-1_26

[7] For example, `peano3` from *set.mm* when fully expanded has a tree size (number of inner nodes) of 7.42×10^{22}. However, the DAG size (number of distinct compound subterms) is only 3,555 (*set.mm* commit `7d05f95`, Aug 5, 2024).

[8] For this experiment we used *set.mm* (https://github.com/metamath/set.mm/) in the version of commit `c10a7e5`, Oct 15, 2023. Our code for the experiments will be made available as free software. Since currently under active development, it is for now only available upon request from the authors.

[7] Letz, R.: Tableau and Connection Calculi. Structure, Complexity, Implementation. Habilitationsschrift, TU München (1999), available from http://www2.tcs.ifi.lmu.de/~letz/habil.ps, accessed Jun 30, 2022

[8] Lohrey, M., Maneth, S., Mennicke, R.: XML tree structure compression using RePair. Inf. Syst. **38**(8), 1150–1167 (2013). https://doi.org/10.1016/j.is.2013.06.006, system available from https://github.com/dc0d32/TreeRePair, accessed Jun 30, 2022

[9] Megill, N., Wheeler, D.A.: Metamath: A Computer Language for Mathematical Proofs. lulu.com, second edn. (2019), online https://us.metamath.org/downloads/metamath.pdf

[10] Megill, N.D.: A finitely axiomatized formalization of predicate calculus with equality. Notre Dame J. of Formal Logic **36**(3), 435–453 (1995). https://doi.org/10.1305/ndjfl/1040149359

[11] Prawitz, D.: An improved proof procedure. Theoria **26**, 102–139 (1960)

[12] Prior, A.N.: Logicians at play; or Syll, Simp and Hilbert. Australasian Journal of Philosophy **34**(3), 182–192 (1956). https://doi.org/10.1080/00048405685200181

[13] Rawson, M., Wernhard, C., Zombori, Z., Bibel, W.: Lemmas: Generation, selection, application. In: Ramanayake, R., Urban, J. (eds.) TABLEAUX 2023. LNAI, vol. 14278, pp. 153–174 (2023). https://doi.org/10.1007/978-3-031-43513-3_9

[14] Wernhard, C.: CD Tools – Condensed detachment and structure generating theorem proving (system description). Tech. rep. (2022). https://doi.org/10.48550/arXiv.2207.08453

[15] Wernhard, C.: Generating compressed combinatory proof structures — an approach to automated first-order theorem proving. In: Konev, B., Schon, C., Steen, A. (eds.) PAAR 2022. CEUR Workshop Proc., vol. 3201. CEUR-WS.org (2022), https://arxiv.org/abs/2209.12592

[16] Wernhard, C.: Structure-generating first-order theorem proving. In: Otten, J., Bibel, W. (eds.) AReCCa 2023. CEUR Workshop Proc., vol. 3613, pp. 64–83. CEUR-WS.org (2024), https://ceur-ws.org/Vol-3613/AReCCa2023_paper5.pdf

[17] Wernhard, C., Bibel, W.: Learning from Łukasiewicz and Meredith: Investigations into proof structures. In: Platzer, A., Sutcliffe, G. (eds.) CADE 28. LNCS (LNAI), vol. 12699, pp. 58–75. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_4

[18] Wernhard, C., Bibel, W.: Investigations into proof structures. J. Autom. Reasoning (2024), to appear, preprint https://arxiv.org/abs/2304.12827

[19] Woltzenlogel Paleo, B.: Atomic cut introduction by resolution: Proof structuring and compression. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16. LNCS, vol. 6355, pp. 463–480. Springer (2010). https://doi.org/10.1007/978-3-642-17511-4_26

A Extracting New Lemmas

Here we list the 46 lemmas that were identified by TreeRePair after processing the proofs of the first 1500 theorems in *set.mm*. For each lemma we show its proof term definition and its most general theorem in a notation that mimics that of *set.mm*, but uses Prolog variables (uppercase letters). The role of the `P` predicate in (5) is taken there by `|-`. The `$e` statements form the body of the Horn clause, and the `$p` statement its head. For 3 of these lemmas the most general theorem already appears as a theorem in *set.mm*, but with a proof that is different from our proof obtained with tree compression. They are annotated with a line *Same MGT as* that indicates the respective theorem in *set.mm*. This suggests the possibility to simplify some proofs in *set.mm* by using lemmas already present. That the discovered lemmas are about propositional formulas is since we processed the first 1500 theorems of *set.mm*, which are on propositional calculus. Analyzing the identified lemmas further is left for future work.

```
lemma26(X,Y) -> bitri(X,anbi1i(Y)).
$e |- ( A <-> ( B /\ C ) ) $.
$e |- ( B <-> D ) $.
$p |- ( A <-> ( D /\ C ) ) $.

lemma45(X) -> '3bitr4i'(X,'df-xor','df-xor').
$e |- ( -. ( A <-> B ) <-> -. ( C <-> D ) ) $.
$p |- ( ( A \/_ B ) <-> ( C \/_ D ) ) $.

lemma56(X,Y) -> syland(X,ancomsd(Y)).
$e |- ( A -> ( B -> C ) ) $.
$e |- ( A -> ( ( D /\ C ) -> E ) ) $.
$p |- ( A -> ( ( B /\ D ) -> E ) ) $.

lemma60(X) -> syl(ifptru,X).
$e |- ( ( if- ( A , B , C ) <-> B ) -> D ) $.
$p |- ( A -> D ) $.

lemma65(X) -> '3ad2ant2'('3ad2ant2'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ ( D /\ A /\ E ) /\ F ) -> B ) $.

lemma66(X) -> '3ad2ant1'('3ad2ant2'(X)).
$e |- ( A -> B ) $.
$p |- ( ( ( C /\ A /\ D ) /\ E /\ F ) -> B ) $.

lemma73(X) -> '3ad2ant1'('3ad2ant1'(X)).
$e |- ( A -> B ) $.
$p |- ( ( ( A /\ C /\ D ) /\ E /\ F ) -> B ) $.

lemma74(X) -> '3ad2ant1'('3ad2ant3'(X)).
$e |- ( A -> B ) $.
$p |- ( ( ( C /\ D /\ A ) /\ E /\ F ) -> B ) $.

lemma76(X) -> 'ax-mp'('df-bi',X).
$e |- ( -. ( ( ( A <-> B ) -> -. ( ( A -> B ) -> -. ( B -> A ) ) ) ->
            -. ( -. ( ( A -> B ) -> -. ( B -> A ) ) -> ( A <-> B ) ) ) -> C ) $.
$p |- C $.
```

```
lemma77(X) -> '3ad2ant3'('3ad2ant3'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ D /\ ( E /\ F /\ A ) ) -> B ) $.

lemma81(X) -> '3ad2ant3'('3ad2ant2'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ D /\ ( E /\ A /\ F ) ) -> B ) $.

lemma83(X) -> '3ad2ant2'('3ad2ant3'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ ( D /\ E /\ A ) /\ F ) -> B ) $.

lemma87(X) -> '3ad2ant3'('3ad2ant1'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ D /\ ( A /\ E /\ F ) ) -> B ) $.

lemma88(X) -> '3ad2ant2'('3ad2ant1'(X)).
$e |- ( A -> B ) $.
$p |- ( ( C /\ ( A /\ D /\ E ) /\ F ) -> B ) $.

lemma115(X) -> syl5(notnotr,X).
$e |- ( A -> ( B -> C ) ) $.
$p |- ( A -> ( -. -. B -> C ) ) $.

lemma129 -> bicomd(ibar).
$p |- ( A -> ( ( A /\ B ) <-> B ) ) $.

lemma143(X) -> ja(X,id).
$e |- ( -. A -> B ) $.
$p |- ( ( A -> B ) -> B ) $.

lemma145(X,Y) -> 'ax-mp'('ax-1','ax-mp'(X,Y)).
$e |- A $.
$e |- ( A -> ( ( B -> ( C -> B ) ) -> D ) ) $.
$p |- D $.

lemma154(X,Y,Z) -> '3bitr4i'(anbi2i(X),Y,Z).
$e |- ( A <-> B ) $.
$e |- ( C <-> ( D /\ A ) ) $.
$e |- ( E <-> ( D /\ B ) ) $.
$p |- ( C <-> E ) $.

lemma157(X,Y) -> syland(a1i(X),ancomsd(Y)).
$e |- ( A -> B ) $.
$e |- ( C -> ( ( D /\ B ) -> E ) ) $.
$p |- ( C -> ( ( A /\ D ) -> E ) ) $.

lemma163 -> bicomd(iba).
$p |- ( A -> ( ( B /\ A ) <-> B ) ) $.

lemma164 -> '3anbi1d'(biidd).
$p |- ( A -> ( ( B /\ C /\ D ) <-> ( B /\ C /\ D ) ) ) $.
```

```
lemma173(X) -> ancoms(imp(X)).
$e |- ( A -> ( B -> C ) ) $.
$p |- ( ( B /\ A ) -> C ) $.
```
Same MGT as `impcom`.

```
lemma174(X) -> ex(ancoms(X)).
$e |- ( ( A /\ B ) -> C ) $.
$p |- ( B -> ( A -> C ) ) $.
```
Same MGT as `expcom`.

```
lemma180(X,Y) -> '3bitr4i'(X,Y,orbi12i(Y,Y)).
$e |- ( A <-> ( A \/ A ) ) $.
$e |- ( B <-> A ) $.
$p |- ( B <-> ( B \/ B ) ) $.

lemma192(X) -> com23(ex(X)).
$e |- ( ( A /\ B ) -> ( C -> D ) ) $.
$p |- ( A -> ( C -> ( B -> D ) ) ) $.

lemma198 -> imbi1i(notnotb).
$p |- ( ( A -> B ) <-> ( -. -. A -> B ) ) $.

lemma210 -> notbid(id).
$p |- ( ( A <-> B ) -> ( -. A <-> -. B ) ) $.

lemma212(X,Y) -> '3bitr4i'(anbi1i(X),Y,'df-3an').
$e |- ( A <-> ( B /\ C ) ) $.
$e |- ( D <-> ( A /\ E ) ) $.
$p |- ( D <-> ( B /\ C /\ E ) ) $.

lemma259 -> imbi2d(bicom1).
$p |- ( ( A <-> B ) -> ( ( C -> B ) <-> ( C -> A ) ) ) $.

lemma263 -> 'ax-mp'('df-bi',simplim).
$p |- ( ( A <-> B ) -> -. ( ( A -> B ) -> -. ( B -> A ) ) ) $.

lemma275(X) -> imp(imim2d(X)).
$e |- ( A -> ( B -> C ) ) $.
$p |- ( ( A /\ ( D -> B ) ) -> ( D -> C ) ) $.

lemma284 -> a1i(simpl).
$p |- ( A -> ( ( B /\ C ) -> B ) ) $.

lemma285 -> imbi2i(notnotb).
$p |- ( ( A -> B ) <-> ( A -> -. -. B ) ) $.

lemma292(X) -> 'pm5.32ri'(syl6rbb(X,orcom)).
$e |- ( A -> ( B <-> ( C \/ D ) ) ) $.
$p |- ( ( ( D \/ C ) /\ A ) <-> ( B /\ A ) ) $.

lemma310 -> 'pm5.74ri'(ibibr).
$p |- ( A -> ( B <-> ( B <-> A ) ) ) $.
```

```
lemma317(X) -> bitri(anbi1i(X),anass).
$e |- ( A <-> ( B /\ C ) ) $.
$p |- ( ( A /\ D ) <-> ( B /\ ( C /\ D ) ) ) $.

lemma322(X) -> orri(biimpi(X)).
$e |- ( -. A <-> B ) $.
$p |- ( A \/ B ) $.

lemma328 -> con3i('ax-1').
$p |- ( -. ( A -> B ) -> -. B ) $.

lemma338 -> bicomd(biorf).
$p |- ( -. A -> ( ( A \/ B ) <-> B ) ) $.

lemma346(X,Y,Z) -> impbid(expcom(X),jaod(Y,Z)).
$e |- ( ( A /\ B ) -> ( C \/ D ) ) $.
$e |- ( B -> ( C -> A ) ) $.
$e |- ( B -> ( D -> A ) ) $.
$p |- ( B -> ( A <-> ( C \/ D ) ) ) $.

lemma349(X,Y) -> 'pm2.61ian'(adantrr(X),adantrl(Y)).
$e |- ( ( A /\ B ) -> C ) $.
$e |- ( ( -. A /\ D ) -> C ) $.
$p |- ( ( B /\ D ) -> C ) $.
```

Same MGT as `exmid2`.

```
lemma351 -> imdistani('pm2.27').
$p |- ( ( A /\ ( A -> B ) ) -> ( A /\ B ) ) $.

lemma362(X,Y) -> syl6c(con3d(X),con3d(Y),'pm5.21im').
$e |- ( A -> ( B -> C ) ) $.
$e |- ( A -> ( D -> C ) ) $.
$p |- ( A -> ( -. C -> ( B <-> D ) ) ) $.
```