Automated Theorem Proving for Metamath

Mario Carneiro
Carnegie Mellon University, Pittsburgh, PA, USA

Chad E. Brown
Czech Technical University in Prague, Czech Republic

Josef Urban
Czech Technical University in Prague, Czech Republic

Abstract

Metamath is a proof assistant that keeps surprising outsiders by its combination of a very minimalist design with a large library of advanced results, ranking high on the Freek Wiedijk's 100 list. In this work, we develop several translations of the Metamath logic and its large set-theoretical library into higher-order and first-order TPTP formats for automated theorem provers (ATPs). We show that state-of-the-art ATPs can prove 68% of the Metamath problems automatically when using the premises that were used in the human-written Metamath proofs. Finally, we add proof reconstruction and premise selection methods and combine the components into the first hammer system for Metamath.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Higher order logic; Theory of computation → Logic and verification

Keywords and phrases Metamath, Automated theorem proving, Interactive theorem proving, Formal proof assistants, proof discovery

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.9

Supplementary Material *Software*: `https://github.com/ai4reason/mm-atp-benchmark`
Software: `https://github.com/digama0/mm-hammer`

Funding This work was partially supported by the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/0000466 (CB, JU), Amazon Research Awards (CB, JU), the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902 (CB), and the EU ICT-48 2020 project TAILOR no. 952215 (JU).
Mario Carneiro: Supported by the Hoskinson Center for Formal Mathematics at CMU.

1 Introduction

Metamath [20] is a formal system developed by Norman Megill in 1990. Its largest database, `set.mm`[1], has 40338 theorems in ZFC set theory, including a diverse range of topics including analysis, topology, graph theory, number theory, Hilbert spaces, and it continues to grow steadily due to its small but active community. In the space of theorem prover languages, it is one of the simplest, by design.

Metamath is one of the last formal proof systems with a large mathematical library that have not yet been translated to today's automated theorem provers (ATPs) [23]. Such translations between ITPs and ATPs are one of the main parts of *hammer systems* [4], which have become popular in the recent years, especially in the Isabelle community [21, 22, 3, 18, 8]. Hammer systems today exist also for the Coq [7, 9], HOL [10, 15], and Mizar [27, 16, 14] proof assistants. The goal of this work is to provide the first such ATP translation for Metamath, and to do the first evaluation of the potential of state-of-the-art ATP systems on

[1] `https://github.com/metamath/set.mm`

the translated Metamath library. This also results in a new large mathematical benchmark for ATP systems. We also build other components of the first full Metamath hammer here, such as proof reconstruction and premise selection [1].

The rest of the paper is structured as follows. Section 2 provides a brief summary of Metamath. In Section 3, we describe a translation of Metamath to HOL via the Metamath Zero (MM0) system. This is then used as an input to several versions of translation to the higher-order TPTP (TH0) format [11] in Section 4. We then describe an alternative translation to first-order class theory in Section 5. Section 6 describes the resulting large new benchmark of higher-order and first-order ATP problems obtained by the translation, and Section 7 provides the first evaluation of higher-order and first-order ATPs on the benchmark. We show that with higher time limits, the ATP systems can jointly prove 68% of the problems when using the premises provided in the human written Metamath proofs. This is a very encouraging result, both for the development of hammers for Metamath, and for the developers of ATP systems. In section 8 we discuss how IVY proofs are reconstructed to Metamath, and section 9 has some interesting examples of theorems the provers were able to get.

2 Metamath

Development in Metamath is generally facilitated by a proof assistant, and unlike many theorem provers there is no proof assistant that has a monopoly on the job; the most common proof assistants in use are mmj2 by Mel O'Cat and MM-PA which is bundled with the original `metamath.exe` verifier by Norman Megill. A `.mm` Metamath file does not contain proof scripts, but rather it is a textual format for complete and fully explicit proof objects. This makes it very attractive as a data source, because very little is required to parse and validate the theorems from the file.

The name "Metamath" comes from "metavariable mathematics," because the core concept is the pervasive use of metavariables over an object logic. For example, theorem ax6e[2] asserts $\vdash \exists x\ x = y$, but x and y are metavariables ranging over variables in FOL. Depending on whether x and y are taken to be the same or different FOL variables, we get two α-equivalence classes of FOL theorems from this single Metamath theorem: $\vdash \exists u\ u = v$ asserts that there exists an element equal to some fixed free variable v, and $\vdash \exists u\ u = u$ asserts that there exists an element equal to itself. These are both true statements, and the original Metamath theorem includes both of these as substitution instances.

This ability for a Metamath theorem to encode multiple α-equivalence classes of FOL theorems is known in the Metamath community as "bundling," and it poses a problem for translation to plain FOL or HOL.

3 Translating Metamath to HOL via MM0

Metamath Zero [5] is a formal system developed by M. Carneiro with a logic somewhat intermediate between Metamath and HOL. It positions itself as an interchange language between other proof systems, and in particular the MM0 toolchain[3] implements a translation from Metamath to MM0 that addresses exactly this bundling issue. MM0 requires that all

[2] `https://us.metamath.org/mpeuni/ax6e.html`
[3] `https://github.com/digama0/mm0`

theorems are fully unbundled, meaning that a theorem like ax6e has to be split into two theorems:

- ax6e: $\vdash \exists x\ x = y$
- ax6e_b: $\vdash \exists x\ x = x$

Each theorem now has a straightforward rendering as a theorem in FOL.

Another problem that the translator addresses is parsing of math expressions. In metamath, the native representation of statements is as sequences of constant and variable tokens, so a verifier does not truly need to know how to break the statement into formula constructors – this is encoded as part of the proof itself. In MM0, the native representation instead uses trees of expression constructors, which is a better fit for traditional ATPs. So ax6e would actually be translated as $(\mathsf{wex}\ x\ (\mathsf{wceq}\ (\mathsf{cv}\ x)\ (\mathsf{cv}\ y)))$ which now encodes the parse tree of $\exists x\ x = y$ (including the invisible cv coercion from set variables to class expressions).

The MM0 toolchain also has a translator from MM0 to HOL, in a lisp-based format. The main mismatch at this level is that MM0 variables represent open terms, and so they have to be transformed into higher order variables to match HOL semantics. For example, the Metamath theorem axi4:[4] $\vdash (\forall x\ \varphi \to \varphi)$ is translated to the MM0 theorem:

$$\mathsf{axi4}\ \{x : \mathsf{setvar}\}\ (\varphi : \mathsf{wff}\ x) \colon (\mathsf{wi}\,(\mathsf{wal}\ x\ \varphi)\ \varphi)$$

where the binders encode that x is a first order set variable and φ is a second order wff variable that is allowed to depend on x.[5] It is translated to the following HOL theorem:[6]

$$\mathsf{axi4} : \forall\,(\varphi : \mathsf{setvar} \to \mathsf{wff})\ (x : \mathsf{setvar}).\quad (\mathsf{wi}\,(\mathsf{wal}\,(\lambda x : \mathsf{setvar}.\ \varphi\ x))\,(\varphi\ x))$$

This is written in lisp concrete syntax as:

```
(theorem "axi4"
  (for ("ph" ("setvar" "wff")))
  (for)
  (for ("x3" "setvar"))
  ("wi"
    ("wal" (fn ("x3" "setvar") ("ph" "x3")))
    ("ph" "x3"))
  ("sp" (fn ("x3" "setvar") ("ph" "x3")) "x3"))
```

The three `(for)` blocks are for introducing binders for the second order variables, then the hypotheses (of which there are none in this example), and finally the first order variables, and then the type of the theorem as above. The last expression, $(\mathsf{sp}\ (\lambda x : \mathsf{setvar}.\ \varphi\ x)\ x)$, is the proof, encoded as a lambda calculus term. (The proof in this case is short because axi4 is just an alias of theorem sp, but usually this will include many theorem applications.)

By chaining all these translations on `set.mm` we obtain a `set.lisp` file containing statements and complete proofs for every theorem in `set.mm`.

[4] `https://us.metamath.org/mpeuni/axi4.html`
[5] Note that "wff" is a common abbreviation for "well-formed formula."
[6] Names of costructors that return wffs begin with the letter w. For example, wi is the wff constructor for implication and wal is the wff constructor for universal quantification (over sets).

4 Translating Metamath in HOL to TH0

From the higher-order representation of set.mm we have several options for how to create TH0 problems for automated theorem provers. We consider specifically three versions (denoted as *v1*, *v2* and *v3* below) of this last part of the translation. To describe the difference between these three versions, we consider a few example theorems from set.mm.

4.1 Translation v1

Each set.mm theorem consists of a finite list of premises[7] (wffs) and a conclusion (a wff). The premises and conclusion may depend on certain variables ranging over wffs, classes and sets. Wff variables and class variables may depend on sets and these dependencies are made explicit in the higher-order translation of set.mm. Set variables may also be locally bound (while variables representing wffs and classes are global to the theorem). In the source higher-order logic (after translating set.mm as above and before translating to TH0) there are three base types: wff, class and setvar. We also have function types $\alpha \to \beta$ as usual. There are a variety of constructors for wffs and classes. For the examples we only need these few, given here with their source higher-order types:

- wi : wff → wff → wff (implication)
- wa : wff → wff → wff (conjunction)
- wb : wff → wff → wff (equivalence)
- w3a : wff → wff → wff → wff (tertiary conjunction)
- wceq : class → class → wff (equality on classes)
- wcel : class → class → wff (membership on classes)
- wal : (setvar → wff) → wff (universal quantification)
- wsb : (setvar → wff) → (setvar → wff) (substitution)
- cab : (setvar → wff) → class (class abstraction)[8]

In set.mm a set variable can be used as a class. After translating to higher-order, we need a corresponding way to coerce a set variable to be a class. This is given by sv of type setvar → class.

When translating to TH0, we can use the builtin type o (`$o` in ASCII) for wff and the builtin type ι (`$i` in ASCII) for setvar. Also, we can use the type $\iota \to o$ for class. So the translated types (for all versions of the translations) are as follows:

- wi : $o \to o \to o$
- wa : $o \to o \to o$
- wb : $o \to o \to o$
- w3a : $o \to o \to o \to o$
- wceq : $(\iota \to o) \to (\iota \to o) \to o$
- wcel : $(\iota \to o) \to (\iota \to o) \to o$
- wal : $(\iota \to o) \to o$
- wsb : $(\iota \to o) \to \iota \to o$
- cab : $(\iota \to o) \to \iota \to o$
- sv : $\iota \to \iota \to o$

[7] Here the words *premise* and *conclusion* are used in the meaning of *antecedent* and *succedent* of a sequent. In particular, *premise* does not mean here *"another toplevel fact used in the proof"* (premise selection terminology).

[8] This is a constructor that returns a class. Names of such constructors usually begin with c.

There are over 1200 other constructors we will not explicitly mention here. There are no constructors (even among the unmentioned ones) returning ι, so that when translating we can always assume terms of type ι are variables.

Consider the theorem sylan9eq.[9] The theorem depends on two wff variables φ and ψ and three class variables A, B and C. The theorem has two premises: $\mathsf{wi}\ \varphi\ (\mathsf{wceq}\ A\ B)$ and $\mathsf{wi}\ \psi\ (\mathsf{wceq}\ B\ C)$. The conclusion of the theorem is $\mathsf{wi}\ (\mathsf{wa}\ \varphi\ \psi)\ (\mathsf{wceq}\ A\ C)$. In the v1 translation this theorem is translated into the closed proposition

$$\forall (\varphi\, \psi : o)\, (A\, B\, C : \iota \to o).\ \mathsf{wi}\, \varphi\, (\mathsf{wceq}\, A\, B) \to \mathsf{wi}\, \psi\, (\mathsf{wceq}\, B\, C) \to \mathsf{wi}\, (\mathsf{wa}\, \varphi\, \psi)\, (\mathsf{wceq}\, A\, C).$$

This closed proposition is, of course, not provable in isolation. When creating the TH0 problem we examine the set.mm proof to determine the axioms and previous theorems used in the proof. In the case of sylan9eq two previous results are used: syl2an[10] and eqtr.[11] These two previous results can be assumed to have been translated earlier to yield the following two closed propositions:

- syl2an : $\forall (\varphi\, \psi\, \xi\, \theta\, \tau : o).\ \mathsf{wi}\ \varphi\ \psi \to \mathsf{wi}\ \tau\ \xi \to \mathsf{wi}\ (\mathsf{wa}\ \psi\ \xi)\ \theta \to \mathsf{wi}\ (\mathsf{wa}\ \varphi\ \tau)\ \theta$
- eqtr : $\forall (A\, B\, C : \iota \to o).\ \mathsf{wi}\ (\mathsf{wa}\ (\mathsf{wceq}\ A\ B)\ (\mathsf{wceq}\ B\ C))\ (\mathsf{wceq}\ A\ C)$

The TH0 problem given by v1 translation of sylan9eq declares syl2an and eqtr (in the form shown above) as axioms and declares the v1 translation of the theorem (as shown above) as the conjecture. (The reader can easily check that the conjecture follows from the two axioms.)

We next consider the theorem axi4. This example has no premises and one wff variable φ and one set variable x. In this case, the wff φ has a dependence on a set variable and so has type $\iota \to o$ (as opposed to o in the previous example). The conclusion of the theorem is $\mathsf{wi}\ (\mathsf{wal}\ (\lambda x.\varphi\ x))\ (\varphi\ x)$. The v1 translation quantifies over φ and x to create the closed proposition $\forall \varphi : \iota \to o. \forall x : \iota.\mathsf{wi}\ (\mathsf{wal}\ (\lambda x.\varphi\ x))\ (\varphi\ x)$. The set.mm proof uses one previous result, called sp,[12] which the v1 translation of is precisely the same as the closed proposition above, making the TH0 theorem proving problem trivial (the single axiom is the same as the conjecture).

4.2 Translation v2

Since wi, wa and wceq are intended to correspond to implication, conjunction and equality (on classes), we also created translations that make use of this intention. Translation v2 behaves as v1 except that for 10 constructors (corresponding to true, false, implication, conjunction, equivalence, negation, disjunction, equality, universal quantification and existential quantification) are translated using their intended meaning. In particular $\mathsf{wi}\ \varphi\ \psi$ translate as $\varphi' \to \psi'$ (where φ' is the v2 translation of φ and ψ' is the v2 translation of ψ). Likewise, $\mathsf{wa}\ \varphi\ \psi$ and $\mathsf{wb}\ \varphi\ \psi$ translate as $\varphi' \wedge \psi'$ and $\varphi' \leftrightarrow \psi'$. Similarly, $\mathsf{wceq}\ A\ B$ translates to $A' = B'$ where A' and B' are the v2 translation of A and B. For the universal quantifier, $\mathsf{wal}\ (\lambda x.\ \varphi)$ is translated to $\forall x.\ \varphi'$ where φ' is the v2 translation of φ.[13]

[9] `https://us.metamath.org/mpeuni/sylan9eq.html`
[10] `https://us.metamath.org/mpeuni/syl2an.html`
[11] `https://us.metamath.org/mpeuni/eqtr.html`
[12] `https://us.metamath.org/mpeuni/sp.html`
[13] We can assume the argument is of the form $\lambda x.\varphi$ by η-expansion.

4.3 Translation v3

Translation v3 behaves as v2 except that 11 more constructors are translated using their intended meaning, including w3a, wsb and cab. The v3 translation of w3a $\varphi\ \psi\ \xi$ is $\varphi' \wedge \psi' \wedge \xi'$ where φ', ψ' and ξ' are the v3 translations of φ, ψ and ξ. The v3 translation of wsb $(\lambda x.\ \varphi)\ y$ is $(\lambda x.\ \varphi')\ y$ which β-reduces to φ'^{x}_{y}.[14] This corresponds to substituting y for x in the (translation of the) formula φ. A term of the form cab $(\lambda x.\ \varphi)$ is meant to return the class $\{x \mid \varphi\}$. Since classes are predicates of type $\iota \to o$ and membership of a set in a class corresponds to application of the predicate to the set, the v3 translation of cab $(\lambda x.\ \varphi)\ y$ is simply taken to be φ'^{x}_{y}, treating it essentially the same way as wsb. Using η-reduction, we can also say wsb $(\lambda x.\ \varphi)$ and cab $(\lambda x.\ \varphi)$ v3 translate to $\lambda x.\ \varphi'$.

The v2 and v3 translations of the conjecture for sylan9eq are both

$$\forall\,(\varphi\ \psi : o)\ (A\ B : \iota \to o).\ \ (\varphi \to A = B) \to (\psi \to B = C) \to \varphi \wedge \psi \to A = C.$$

The two TH0 problems (for v2 and v3) also include the translation of the two dependencies syl2an and eqtr as axioms, though these are no longer needed for the proof.

4.4 More Examples

The v2 and v3 translations of the conjecture for axi4 are both

$$\forall\,(\varphi : \iota \to o)\ (x : \iota).\ \ (\forall x.\ \varphi\ x) \to \varphi\ x.$$

Again, the dependency is also translated and included in as an axiom in the TH0 problem, though the axiom is no longer needed to prove the conjecture.

To see the distinction between the v2 and v3 translations, we briefly consider three more small examples: rp_simp2,[15] sbt[16] and abbi2i.[17] The three translations of the conjecture for rp_simp2 are as follows:

- v1: $\forall\,(\varphi\ \psi\ \xi : o).$ wi (w3a $\varphi\ \psi\ \xi$) ψ
- v2: $\forall\,(\varphi\ \psi\ \xi : o).$ w3a $\varphi\ \psi\ \xi\ \to\ \psi$
- v3: $\forall\,(\varphi\ \psi\ \xi : o).\ \ \varphi\ \wedge\ \psi\ \wedge\ \xi\ \to\ \psi$

The three translations of the conjecture for sbt are as follows:

- v1 and v2: $\forall\,(\varphi : \iota \to \iota \to o).\ \ (\forall\,x\ y : \iota.\ \varphi\ x\ y) \to \forall\,y : \iota.$ wsb $(\lambda z.\varphi\ z\ y)\ y$
- v3: $\forall\,(\varphi : \iota \to \iota \to o).\ \ (\forall\,x\ y : \iota.\ \varphi\ x\ y) \to \forall y : \iota.\ \varphi\ y\ y$

The three translations of the conjecture for abbi2i are as follows:

- v1: $\forall\,(\varphi : \iota \to o)\ (A : \iota \to o).\ \ (\forall x : \iota.$ wb (wcel (cv x) A) $(\varphi\ x)) \to$ wceq A (cab $(\lambda x.\varphi x)$)
- v2: $\forall\,(\varphi : \iota \to o)\ (A : \iota \to o).\ \ (\forall x : \iota.$ (wcel (cv x) A) $\leftrightarrow\ (\varphi\ x)) \to A\ =$ cab $(\lambda x.\varphi x)$
- v3: $\forall\,(\varphi : \iota \to o)\ (A : \iota \to o).\ \ (\forall x : \iota.$ (wcel (cv x) A) $\leftrightarrow\ (\varphi\ x)) \to A\ =\ (\lambda x.\varphi x)$

4.5 Why three translations?

The translations v1-v3 represent increasingly "deep" interpretation of the Metamath formulas. One might wonder why we are considering all of them, instead of just using the best one – clearly giving the prover more information is a good idea. (And as 7 will show, this is largely correct.) However, there are three complicating factors that make it not a completely one sided tradeoff:

[14] We can assume the second argument is a variable y since it has type ι.
[15] https://us.metamath.org/mpeuni/rp-simp2.html
[16] https://us.metamath.org/mpeuni/sbt.html
[17] https://us.metamath.org/mpeuni/abbi2i.html

- If the prover is given more information, it has more options, and this can cause it to run away and prove the wrong things.
- Also following from the previous point, the prover will be more "creative" with the more deeply embedded proofs, performing more normalization and often resulting in longer proofs than if it is forced to play by Metamath rules.
- Most pertinently for our hammer system, the deeper translations require more of the prover's mechanisms to be translatable back to Metamath proofs, and since only the simpler mechanisms have reconstruction implemented for them, using the deeper translations can cause the reconstruction rate to decrease, even though the ATP has a higher success rate, since it will come up with proofs outside the translatable fragment.

5 Translating Metamath in HOL to First-Order Class Theory

We additionally translated the higher-order representation of set.mm into first-order theorem proving problems by interpreting propositions and terms as classes. This allows us to compare the performance of first-order and higher-order ATPs on problems coming from a common source. In addition we use the first-order prover Prover9 [19] to obtain IVY proof objects which we use to reconstruct Metamath proofs.

Recall that the type of cab in the HOL representation is $(\mathsf{setvar} \to \mathsf{wff}) \to \mathsf{class}$. In the resulting TH0 for the v1 and v2 translations terms with cab at the head could always be assumed to be of the form $\mathsf{cab}\ (\lambda x.\ \varphi)$ (by η-expansion if necessary). We do not have such term level binders in first-order terms, so we must find an alternative method to handle such binders. Every occurrence of a wff or class will be under n setvar binders, binding $x_0, \ldots, x_{n-1} \in V$, where V is the class of all sets. Instead of making the binders explicit in the logic, we translate a wff φ in context $x_0, \ldots, x_{n-1}$ as the class $\{((x_0, \ldots, x_{n-1}), \emptyset) \mid \varphi\}$ (i.e., the class of all n-tuples of sets for which φ holds, with an associated dummy value of $\emptyset$). Likewise we translate a class A in context $x_0, \ldots, x_{n-1}$ as the class $\{((x_0, \ldots, x_{n-1}), y) \mid y \in A\}$. The operator cab takes a wff in context $x_0, \ldots, x_{n-1}, x_n$ to a class in context $x_0, \ldots, x_{n-1}$, where n need not be known in advance. In particular we take $\mathsf{cab}(B)$ to be the class

$$\{((x_0, \ldots, x_{n-1}), x_n) \mid n \in \omega,\ ((x_0, \ldots, x_{n-1}, x_n), \emptyset) \in B\}.$$

For example, suppose we have a HOL version of a Metamath term of the form $\mathsf{cab}\ (\lambda x_n.\ \varphi)$ in a context $x_0, \ldots, x_{n-1}$. We can translate φ in context $x_0, \ldots, x_{n-1}, x_n$ to obtain a first-order term φ' representing the class $\{((x_0, \ldots, x_{n-1}, x_n), \emptyset) \mid \varphi'\}$. We then translate $\mathsf{cab}\ (\lambda x_n.\ \varphi)$ simply to be the first-order term $\mathsf{cab}(\varphi')$, corresponding to the class

$$\{((x_0, \ldots, x_{n-1}), x_n) \mid ((x_0, \ldots, x_{n-1}, x_n), \emptyset) \in \varphi'\}.$$

The distinction between sets and proper classes is not useful as we will generally be concerned with sets in a context of a certain length. For example, the proper class $\{((x_0, x_1), y) \mid y \in x_1\}$ can be considered a set in a context of length 2 since if $x_0, x_1 \in V$ are fixed, then $\{y \mid y \in x_1\}$ is a set. We say a class A is a set in a context of length n if $\{y \mid ((x_0, \ldots, x_{n-1}), y) \in A\}$ is a set for every $x_0, \ldots, x_{n-1} \in V$. Note that a class can be a set in a context of different lengths. For example, the empty class is a set in a context of length n for every $n \in \omega$. Consider the Metamath wff wtru (corresponding to the true wff). This will be translated to a first-order constant wtru intended to be the class $\{((x_0, \ldots, x_{n-1}), \emptyset) \mid x_0, \ldots, x_{n-1} \in V\}$. Note that this is both a proper class and a set in a context of length n for every $n \in \omega$.

In general Metamath constructors with functional arguments may change the length of the context, as cab does. However, most operations do not change the length of the context. For example, $\mathsf{wa}(\varphi, \psi)$ in context $x_0, \ldots, x_{n-1}$ is simply $\mathsf{wa}(\varphi', \psi')$ where φ and ψ are the translations of φ and ψ in context $x_0, \ldots, x_{n-1}$. The intended semantics of $\mathsf{wa}(B, C)$ is

$$\{((x_0, \ldots, x_{n-1}), \emptyset) \mid ((x_0, \ldots, x_{n-1}), \emptyset) \in B \wedge ((x_0, \ldots, x_{n-1}), \emptyset) \in C\}.$$

Note that if φ' is a class $\{((x_0, \ldots, x_{n-1}), \emptyset) \mid \varphi\}$, ψ' is a class $\{((x_0, \ldots, x_{n-1}), \emptyset) \mid \psi\}$ and $\mathsf{wa}(\varphi', \psi')$ is the class $\{((x_0, \ldots, x_{n-1}), \emptyset) \mid \varphi \wedge \psi\}$. As with cab, wa does not depend on the length of the context n.

As a consequence of treating setvar binders in this way, we must decide how to translate the (now implicitly) bound variables as first-order terms. We do this by simply having a constant $\mathtt{var}_n^i$ for each $i < n$, intended to correspond to the variable x_i in the context $x_0, \ldots, x_{n-1}$. Semantically, $\mathtt{var}_n^i$ is the class $\{((x_0, \ldots, x_{n-1}), y) \mid y \in x_i\}$.

Since many wff and class variables depend on a context of set variables, we also include functions $\mathtt{eval}_n^m$ of arity $m + 1$. The intention is that the first argument of $\mathtt{eval}_n^m$ is a class (intended to be in context $x_0, \ldots, x_{m-1}$) and the next m arguments are sets in a context of length n (intended to be in context $x_0, \ldots, x_{n-1}$). Since no Metamath constructor yields a set, the only sets available in context $x_0, \ldots, x_{n-1}$ are each x_i (represented in first-order by the constant $\mathtt{var}_n^i$ and possibly some universal first-order variables intended to range over sets. Each first-order variable Y intended to range over sets will occur as $\mathsf{cv}(Y)$ (the coercion sending sets to classes). We consider a slightly different semantics of cv in the first-order class theory translation than the higher-order case. For a set Y, we take $\mathsf{cv}(Y)$ to be the class

$$\{((x_0, \ldots, x_{n-1}), y) \mid n \in \omega, x_0, \ldots, x_{n-1} \in V, y \in Y\}.$$

That is, cv lifts a set to a class in an arbitrary context (with no dependence on the set variables in the context).

The result of applying $\mathtt{eval}_n^m$ to $m + 1$ arguments is a class (intended to be in context $x_0, \ldots, x_{n-1}$) obtained by composition. In particular, we define $\mathtt{eval}_n^m$ as

$$\begin{aligned}
&\mathtt{eval}_n^m(B, A_0, \ldots, A_{m-1}) = \\
&\qquad \{(x_0, \ldots, x_{n-1}, z) \mid \exists y_0, \ldots, y_{m-1} \in V.\ (y_0, \ldots, y_{m-1}, z) \in B \\
&\qquad\qquad \wedge (\forall y.\ y \in y_0 \Leftrightarrow (x_0, \ldots, x_{n-1}, y) \in A_0) \\
&\qquad\qquad \wedge \cdots \\
&\qquad\qquad \wedge (\forall y.\ y \in y_{m-1} \Leftrightarrow (x_0, \ldots, x_{n-1}, y) \in A_{m-1})\}.
\end{aligned}$$

Note that if some A_i were not a set in a context of length n, then $\mathtt{eval}_n^m(B, A_0, \ldots, A_{m-1}) = \emptyset$ since a corresponding $y_i \in V$ would not exist. This is never relevant in practice as the only arguments to $\mathtt{eval}_n^m$ after the first argument are of the form $\mathtt{var}_n^i$ or $\mathsf{cv}(Y)$, as stated above.

In case we have a Metamath wff variable φ that depends on m set variables, then occurrences of φ in the higher-order representation will be applied to m arguments. To make this first-order we simply choose n to be the length of the context in which the wff occurs and translate as $\mathtt{eval}_n^m(\varphi', A_0, \ldots, A_{m-1})$ where φ' is a first-order variable corresponding to φ and A_i is the first-order term obtained by translating the arguments of φ.

Some special identities are easy to verify given the semantics described above, e.g.,

$$\mathtt{eval}_n^m(\mathtt{var}_m^i, A_0, \ldots, A_{m-1}) = A_i$$

and

$$\mathtt{eval}_n^m(\mathsf{wa}(B, C), A_0, \ldots, A_{m-1}) = \mathsf{wa}(\mathtt{eval}_n^m(B, A_0, \ldots, A_{m-1}), \mathtt{eval}_n^m(C, A_0, \ldots, A_{m-1}))$$

where we assume each A_i is a set in a context of length n. Some first order problems resulting from the translation only become provable if such identities are included. However, for now we have primarily focused on the problems solvable without such identities included.

As a final step to obtain a first-order atomic proposition, we apply a unary predicate `p` to a first-order term φ' (the translation of a Metamath wff φ in an empty context). The intention is that $\mathtt{p}(A)$ should be true precisely if $((), \emptyset) \in A$.

The hypotheses and conclusion of a Metamath axiom or theorem may have universally bound set variables. To translate these to terms we use a unary function $\mathtt{all}_n$. The intended semantics of $\mathtt{all}_n(A)$ is

$$\{((x_0, \dots, x_{n-1}), \emptyset) \mid \forall x_n \in V.\ ((x_0, \dots, x_{n-1}, x_n), \emptyset) \in A\}.$$

Again for some translated problems to be theorems we would need to include certain properties of $\mathtt{all}_n$, e.g.,

$$\forall Y.\ \mathtt{p}(Y) \Leftrightarrow \forall X.\ \mathtt{p}(\mathtt{eval}^1_0(Y, \mathsf{cv}(X))).$$

In practice we omit these extra properties for now.

We again consider the example theorem sylan9eq. In the first-order version the two wff variables φ and ψ and the two class variables A and B all range over classes (and hence are represented simply by first-order variables). The two premises translate to first-order terms terms $\mathsf{wi}(\varphi, \mathsf{wceq}(A, B))$ and $\mathsf{wi}(\psi, \mathsf{wceq}(B, C))$. which can then be used as arguments to `p` to obtain atomic propositions. The conclusion translates to the first order term $\mathsf{wi}(\mathsf{wa}(\varphi, \psi), \mathsf{wceq}(A, C))$. Combining the premises with the conclusion and quantifying over the variables yields the first-order sentence

$$\forall \varphi\ \psi\ A\ B\ C.\ \mathtt{p}(\mathsf{wi}(\varphi, \mathsf{wceq}(A, B))) \to \mathtt{p}(\mathsf{wi}(\psi, \mathsf{wceq}(B, C))) \to \mathtt{p}(\mathsf{wi}(\mathsf{wa}(\varphi, \psi), \mathsf{wceq}(A, C))).$$

As before the sentence is a consequence of syl2an and eqtr which translate to the first-order sentences

$$\forall\ \varphi\ \psi\ \xi\ \theta\ \tau.\ \ \mathtt{p}(\mathsf{wi}(\varphi, \psi)) \to \mathtt{p}(\mathsf{wi}(\tau, \xi)) \to \mathtt{p}(\mathsf{wi}(\mathsf{wa}(\psi, \xi), \theta)) \to \mathtt{p}(\mathsf{wi}(\mathsf{wa}(\varphi, \tau), \theta))$$

and

$$\forall\ A\ B\ C.\ \ \mathtt{p}(\mathsf{wi}(\mathsf{wa}(\mathsf{wceq}(A, B), \mathsf{wceq}(B, C)), \mathsf{wceq}(A, C))).$$

We also reconsider the theorem axi4. This translates to the first-order statement

$$\forall\ \varphi\ x.\ \ \mathtt{p}(\mathsf{wi}(\mathsf{wal}(\mathtt{eval}^1_1(\varphi, \mathtt{var}^0_1)), \mathtt{eval}^1_0(\varphi, x))).$$

Again the proof uses sp which translates to the same first-order statement (making the theorem proving problem trivial).

6 Benchmark

We use the translations to TPTP described in Sections 4 and 5 to create higher-order and first-order ATP problems. This is implemented in Lisp, as a program that reads the HOL Lisp representation of the MM0 (Section 3) version of set.mm as its input, and produces the corresponding ATP problem for each Metamath theorem proved in set.mm .

The version of set.mm we used corresponds to the git repo with a commit from June 24, 2022.[18] There are 40338 theorems with proofs in this version of set.mm. The translation to MM0 increases the number of theorems to 40556 (218 extra theorems) since some theorems also have α-degenerate versions that are used in their degenerated form later in the library. For example, the Metamath set.mm theorem `ax7v` is $x = y \to x = z \to y = z$ where x and y must be distinct variables. The MM0 version expands this into three versions:

- `ax7v`: $x = y \to x = z \to y = z$
- `ax7v_b` (an α-degenerate): $x = y \to x = x \to y = x$
- `ax7v_b1` (another α-degenerate): $x = y \to x = y \to y = y$

There are also three corresponding TH0 problems (in each of v1, v2 and v3).

The axioms of each TH0 problem are determined by the named facts (proof-external facts, premises) used in the MM0 proof. Note that when generating the problem for a given theorem we already have the TH0 formulas corresponding to the previous facts. Note also that the translation from Metamath to MM0 already distinguished between α-degenerates. While a Metamath proof may have depended on `ax7v`, its MM0 proof may depend on `ax7v` and one of its α-degenerates, say `ax7v_b`. In that case the TH0 problem would include axioms corresponding to `ax7v` and `ax7v_b` (but not `ax7v_b1`).

In the end we obtain 40556 TH0 problems for each of v1, v2 and v3.[19] This corresponds precisely to the 40556 MM0 theorems obtained by translating set.mm. To this we also add the first-order version produced by the translation described in Section 5.

7 Initial ATP Evaluation

7.1 Higher-order Evaluation

We first evaluate three top-performing[20] higher-order ATPs on the three higher-order versions of the problems using several values for timeout. Only one full evaluation is done on v1 of the problems, since we consider the encoding suboptimal. Indeed, E-HO 2.6 solves 77.65% (20352 vs 11456) more problems on v2 with v1 adding only 0.23% (46) more solutions to v2. All our experiments are performed on a server with 36 hyperthreading Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz cores and 256 GB of RAM. We use the following ATP systems:

- E prover version 2.6 [25, 24], run both in its default portfolio (auto-schedule) mode and with several strategies developed previously by strategy invention systems [26, 13] targeting ITP libraries [16, 12].
- Vampire version 5980 [17, 2], using mainly default (casc2020) higher-order portfolio. We have also briefly tested some Vampire strategies in a standalone mode.
- Zipperposition version CASC20 [6, 28], using its default CASC'20 portfolio. We have also tried several other Zipperposition settings.

For most of the experiments we have used a time limit of 60s. We have also used initially a lower 10s time limit, and later also higher time limits, especially to see the performance of the portfolio-based systems. The highest time used for an evaluation on the full dataset was 280s, and we have increased it to 600s and 1200s in several cases to see the improvement on unsolved problems by E, Vampire and Zipperposition. Of the 40556 problems the ATPs can in total solve 27436, i.e., 67.65%. The detailed results of the complete runs of the systems are shown in Table 1. Table 2 shows the top-5 greedy cover including incomplete runs done with high time limits on unsolved problems only.

[18] Specifically the commit d75c0dbe.
[19] The benchmark is publicly available at `https://github.com/ai4reason/mm-atp-benchmark` .
[20] `https://www.tptp.org/CASC/J11/WWWFiles/DivisionSummary1.html`

The highest performance is achieved by Zipperposition which in 280s solves 62.68% (25420) of the v3 problems, and 61.53% (24959) of the v2 problems. This (v2) drops to 57.99% (23518) when using half of the time only, i.e. 140s. Vampire solves 58.01% (23555) of the v3 problems in 280s which is a surprisingly good performance given the 2022 CASC results,[21] where Vampire loses more than 25% on Zipperposition (367 vs 460 problems solved in the CASC THF category). Note that none of the ATP developers have yet seen our benchmark and thus could not develop targeted ATP strategies on the problems, as is typically done for TPTP and CASC. Vampire also gains considerably by going from v2 to v3 and by increasing the time limit (the latter likely thanks to its large portfolio mode). It solves only 45.57% (18482) of the v2 problems in 60s, which increases to 52.08% (21123) of the v3 problems in 60s, and to 56.65% (22976) v3 problems in 120s. Furthermore, Vampire gains from running its strongest strategies with higher time limits: 7 of the strategies run separately for 60s on v3 add together 443 problems, raising Vampire's performance to 23998 problems (in general in 240s + 7*60s = 660s).

E prover outperforms Vampire on v2 in 60s (21001 solved by E vs 18482 by Vampire), and even more so in 10s (20352 vs 17160). Surprisingly, E's performance is lower on v3 compared to v2 (20799 vs 21001 in 60s). The performance however does not increase much with higher time limits as in the case of Vampire, indicating that Vampire makes better use of multiple strategies. We have however only evaluated the official E version 2.6, which seems to have been improved a lot very recently by E version 3.0 in the latest 2022 CASC results. We plan to evaluate E 3.0 when it is officially released.

Table 1 The complete runs of the systems on the benchmark, ordered by performance.

System	mode	version	time (s)	solved
Z	portfolio	v3	280	25420
Z	portfolio	v2	280	24959
V	portfolio	v3	280	23555
Z	portfolio	v2	140	23518
V	portfolio	v3	120	22976
V	portfolio	v3	60	21123
E	portfolio	v2	60	21001
E	portfolio	v3	60	20799
E	portfolio	v2	10	20352
E	strat. f17	v3	120	19782
E	strat. f17	v2	10	19624
V	portfolio	v2	60	18482
Z	fo-complete-basic	v2	10	17295
V	portfolio	v2	10	17160
Z	ho-pragmatic	v2	10	16115
E	portfolio	v1	10	11456

7.2 First-order runs

We also evaluate the first order translation, by running Vampire, E and Prover9 on these problems. Vampire proves 15938 of them, while E and Prover9 solve 15136 and 14693 respectively. The Vampire performance can be compared to its 60s performance on the v2 higher-order problems (18482). This likely again demonstrates the efficiency of the v2 and v3 higher-order translations, because practically none of the standard logical connectives are

[21] See footnote 20.

Table 2 The top 5 methods in the greedy sequence. Note that we use different (and also high) time limits and that the high-time runs are only done on previously unsolved problems.

System	mode	version	time (s)	added	sum
Z	portfolio	v3	280	25420	25420
V	portfolio	v3	600	960	26380
V	portfolio	v3	1200	415	26795
E	portfolio	v3	600	279	27074
Z	portfolio	v2	280	124	27198

mapped in a shallow way to their first-order logical counterparts in this first-order translation. Because of that, all these problems are also Horn, and they are also quite small due to the minimized premises. This is likely the reason why all the three ATPs perform similarly here, with Vampire and E unable to benefit from their strategies for large problems with large non-Horn clauses. The systems are also not very complementary, with E adding 148 and Prover9 adding 110 problems to Vampire. Prover9 adds 505 problems to E.

7.3 Premise Selection Experiments

7.3.1 Higher-order runs

We evaluate Vampire also on the large (*chainy*) versions of the higher-order v3 problems, using a 60s time limit. This emulates the hammer-style reasoning with the whole library that exists before a particular theorem was stated. Using its LTB (large-theory batch division of the CASC competition) portfolio, Vampire solves 8509 v3 problems, while its hol mode solves only 4013 v3 problems. This is relatively few compared to the performance on the benchmark with preselected premises.

Vampire uses SInE as its default premise selector. A number of learning-based premise selection methods typically improve on SInE if enough data about previous proofs are available to train on. For simplicity of use and comparison with other standard hammers we have decided to use here a fast implementation of the distance weighted k-nearest neighbor (k-NN), parameterized by several values of k (10, 20, 40, 80, 120, 160, 240) that typically work well with current ATPs. The first evaluation is done chronologically, by training k-NN incrementally on the human-written proofs as the library grows in time. This means that we always allow k-NN to see all the facts and proofs that precede the fact for which it is predicting the premises, but none of the facts and proofs that follow after that.

We use again the standard higher-order portfolio of Vampire and 60s time limit for each of the values of k. The results are shown in Table 3. The performance peaks at around 12k proved problems for the values of $k = 120, 160, 80$. This is more than 40% better than the best SInE result above (8509). All seven k-NN predictions solve together 14787 problems (in general in 7 minutes), with the top three most orthogonal slices (120, 240, and 20) solving 14113 (in general in 3 minutes).

Table 3 Vampire on k-NN premise selection slices.

Premises	10	20	40	80	120	160	240
V-thf v3	9112	10078	11060	11863	12043	11997	11582
V-fof v1	2600	4239	6294	8366	9416	9875	10352

7.3.2 First-order runs

To get a version that can be reconstructed in Metamath, we also evaluate Vampire on the premise selection slices of the large (chainy) first-order versions of the problems. The results for the 7 standard slices are again shown in Table 3. Unlike in the thf-v3 version, Vampires benefits here from increasingly large slices, so we add also slices with 480 and 960 premises to the fof-v1 premise selection. These slices solve 10726 and 10593 problems respectively, adding many new solutions. In total, the 9 first-order slices solve 12373 problems, with the top 4 most complementary slices (480, 960, 80, and 240) solving 12089 problems. This implies a 30% performance in the first-order v1 hammering setting, which we currently use for the first version of the proof reconstruction.

8 Proof Reconstruction

While Vampire has impressive solving capabilities, we were not able to get it to produce a proof object which was sufficiently detailed for our purposes, so we instead turned to the IVY proof format used by Prover9. Prover9 is not as powerful, but we can still use Vampire as a more precise relevance filter by using the lemmas from the proof it produces as input to Prover9, and process the resulting IVY proof.

IVY is a resolution-style proof format for doing classical reasoning, so it is not a priori obvious how to reconstruct these terms into a Metamath proof without a deep embedding. However, our input clauses and the conjecture are all Horn clauses (that is, they have at most one non-negated literal), and this makes all the difference.

IVY proofs consist of the following kind of proof steps:

- `input` steps refer to one of the hypotheses, except that instead of using $\forall\vec{x}.\ \overrightarrow{A_i(\vec{x})} \to B(\vec{x})$, the quantifiers are removed and the clauses are turned into disjunctions, as in $B(\vec{v}) \vee \bigvee_i \neg A_i(\vec{v})$, with the literals possibly reordered.
  - Because these inputs appear in the same order as they were given to the checker, they are easy to identify.
  - The conjecture is negated, so it turns into multiple inputs and the variables are skolemized: $\forall\vec{x}.\ \overrightarrow{A'_i(\vec{x})} \to B'$ becomes $A'_i(\vec{c})$ for each i, plus $\neg B'(\vec{c})$.
- `instantiate` steps refer to a previous step $p : \bigvee_i C_i$ plus a substitution mapping $\{v_i \mapsto t_i\}$ and results in a proof of $\bigvee_i(C_i[v_i \mapsto t_i])$.
- `resolve` steps specify $p : \bigvee_i C_i \vee P$ and $q : \bigvee_i D_i \vee \neg P$ (where P may appear in the middle of the disjunction but is identified by a path), and results in a proof of $\bigvee_i C_i \vee \bigvee_i D_i$.
- `propositional` steps prove an arbitrary clause Q from previous step $p : P$ where $P \to Q$ is a propositional tautology.
- IVY also supports `new_symbol`, `flip`, and `paramod` steps but these never appear in reconstructed proofs.

The key observation is that IVY never leaves the realm of Horn clauses in the proof. This is not syntactically a requirement – proofs can in principle involve arbitrary propositions – but we can see why it might happen with this kind of input:

- All the inputs are Horn clauses.
- `instantiate` or `resolve` on Horn clauses yield more Horn clauses.
- While `propositional` steps can yield non-Horn clauses in principle, this is mainly used for clause simplification, and there is a unique best clause that the solver will want to generate here, namely the input clause with duplicate hypotheses removed. That is, this is used only for simplifying $\neg A \vee \neg A \vee B$ to $\neg A \vee B$.
- The fact that the clauses have this restricted form is likely the reason why we do not observe the more advanced kinds of steps.

So our strategy for reconstruction is essentially to interpret these as proofs in minimal logic or terms in the simply typed lambda calculus, where $(\bigvee_i \neg A_i) \vee B$ is interpreted as $(\bigwedge_i A_i) \to B$ and $\bigvee_i \neg A_i$ is interpreted as $(\bigwedge_i A_i) \to \mathbf{F}$. The proof steps all have associated lambda terms:

- Hypothesis inputs are $H_i(\vec{v}) : (\bigwedge_i A_i(\vec{v})) \to B(\vec{v})$
- The conjecture is $\mathbf{thm} : B'(\vec{c}) \to \mathbf{F}$
- `instantiate`$(p, \{v_i \mapsto t_i\})$ is just $p[v_i \mapsto t_i]$
- For `resolve`(p, q): if $p : \bigwedge_i C_i \to P$ and $q : (\bigwedge_i D_i) \wedge P \to A$ or $q : (\bigwedge_i D_i) \wedge P \to \mathbf{F}$, then `resolve`$(p, q) := \lambda \vec{c_i}, \vec{d_i}.\ q(\vec{d_i}, p(\vec{c_i}))$.
- For `propositional` steps we spot the duplicates and generate a term like $\lambda x\, y.\ p(x, x, y)$.

Because the final result is a proof of `false`, we get a closed term of type $\mathbf{F}$ after translation, and we can normalize it to eliminate all the lambdas. Since the only constructor for $\mathbf{F}$ in this grammar is **thm**, the result will be of the form $\mathbf{thm}(p)$, where p is a proof structured out of applications of H_i to a substitution and a list of subproofs, which is exactly the form expected by Metamath proofs. So we strip the **thm** node and the result is a well formed proof.

8.1 Proof Objects

Table 4 shows the longest IVY and Metamath proof objects obtained in the experiments. This is for IVY measured by the number of proof steps, while for Metamath these are lines of the reconstructed proof.

Table 4 Length of the longest proof objects in IVY steps and Metamath lines.

Problem	mercolem6	tgbtwnconn1lem1	hdmap14lem9	isoas	lclkrlem2a
IVY	674	480	392	375	316
Problem	mercolem6	mercolem2	merlem5	mercolem7	minimp_sylsimp
Metamath	5660830	849	77	50	45

An outlier here is `mercolem6`, which is one lemma in the proof that Meredith's axiom

$$((\varphi \to \psi) \to (\bot \to \chi) \to \theta) \to (\theta \to \varphi) \to \tau \to \eta \to \varphi$$

is complete for propositional logic. Prover9 is able to return a proof with only 674 lines, but it balloons to a massive 5 660 830 lines after Metamath reconstruction, over 7 times the size of `set.mm`. The reason for this due to the normalization process described in section 8. Each Metamath proof step is exactly and only an application of a previous theorem, with substitutions for the variables, and then proofs for the hypotheses. That is, in IVY terminology we are structurally required to perform `instantiate` steps only on the leaves of the proof.

What happened in this proof is that Prover9 found a useful *lemma*, which has a long proof, and then applied it many times with different instantiations, and the Metamath proof is forced to replicate the subproof many times in order to push the instantiations to the leaves. This is essentially an artificial restriction caused by our implicit requirement that the hammer should generate *one proof*, rather than a sequence of lemmas leading up to the proof. In actual practice a user would split the proof at this useful lemma and refer to it. In fact, the name `mercolem6` indicates that this is lemma 6 of something, so this technique is already being used here.

Future versions of the hammer may include this kind of lemma generation, but we decided not to pursue it since it is extremely rare. Most of the time the cost of extracting these narrow lemmas is higher than the proof savings for applying them.

9 Examples

Three similar examples Zipperposition and E can prove in the *v3* representations are the set.mm theorems amgm2d[22], amgm3d[23] and amgm4d[24] comparing arithmetic and geometric means. The first theorem states that for positive reals A and B,

$$(A \cdot B)^{\frac{1}{2}} \leq \frac{A+B}{2}.$$

The next two theorems state

$$(A \cdot B \cdot C)^{\frac{1}{3}} \leq \frac{A+B+C}{3}$$

and

$$(A \cdot B \cdot C \cdot D)^{\frac{1}{4}} \leq \frac{A+B+C+D}{4}$$

for positive reals A, B, C and (in the last case) D. All three theorems are proven by making use of a lemma amgmlem[25] giving the property

$$(\Sigma^{M} F)^{\frac{1}{|A|}} \leq \frac{\Sigma^{\mathbb{C}} F}{|A|}$$

where A is a finite set, F is a function from A to positive reals, and Σ^{Q} is performs a binary operation from a given monoid Q to the images of F. In this case $\mathbb{C}$ is the complex field (thought of as its additive group here) and so $\Sigma^{\mathbb{C}}$ is ordinary summation. However, M is the multiplicative group of $\mathbb{C}$ and so Σ^{M} in the usual Π operator performing finitely many multiplications. In order for an ATP to prove the examples above, it must instantiate with appropriate values of A and F in the assumption amgmlem, essentially giving the appropriate n-tuple (for $n \in \{2, 3, 4\}$). In this case the n-tuples are represented as words and special theorems gsumws2[26], gsumws3[27] and gsumws4[28] give equations between applying Σ^{Q} to an appropriate length word and the summation of the "characters." Applying these theorems when proving amgm2d, amgm3d and amgm4d leads to the generation of the appropriate n-tuple (word) being constructed by the ATP via unification.

It is worth noting Zipperposition, E and Vampire could also prove the *v3* problems corresponding to gsumws2, gsumws3 and gsumws4. By contrast, none of the ATPs could prove the vital lemma amgmlem. Also, none of the ATPs could prove amgmw2d[29], a generalized version of amgm2d stating

$$A^{P} \cdot B^{Q} \leq A \cdot P + B \cdot Q$$

for positive reals A, B, P and Q such that $P + Q = 1$.

[22] https://us.metamath.org/mpeuni/amgm2d.html
[23] https://us.metamath.org/mpeuni/amgm3d.html
[24] https://us.metamath.org/mpeuni/amgm4d.html
[25] https://us.metamath.org/mpeuni/amgmw2d.html
[26] https://us.metamath.org/mpeuni/gsumws2.html
[27] https://us.metamath.org/mpeuni/gsumws3.html
[28] https://us.metamath.org/mpeuni/gsumws4.html
[29] https://us.metamath.org/mpeuni/amgmw2d.html

A different example Zipperposition and E can prove is zringunit[30]. This states A is a unit of the ring of integers if and only if A is an integer with norm 1 (i.e., A is -1 or 1). A previous result used in the proof is gzrngunit[31] which states the units of the ring of Gaussian integers is precisely those with norm 1. None of the ATPs were able to prove gzrngunit.

Several other interesting ATP proofs are available on our web page.[32] This includes E's higher-order proof of theorem `xmulneg1`[33] which has 127 steps in Metamath and takes 18131 given clause loops in 30 seconds to E.[34] It proves for extended reals that a product with a negative is the negative of the product:

```
xmulneg1 $p |- ( ( A e. RR* /\ B e. RR* ) -> ( -e A *e B ) = -e ( A *e B ) )
```

E also proves the `matinv` theorem in 12 seconds and 13052 given clause loops, which takes a 73-step proof in Metamath.[35] The theorem states that the inverse of a matrix is the adjunct of the matrix multiplied with the inverse of the determinant of the matrix if the determinant is a unit in the underlying ring:

```
matinv.a $e |- A = ( N Mat R ) $.
matinv.j $e |- J = ( N maAdju R ) $.
matinv.d $e |- D = ( N maDet R ) $.
matinv.b $e |- B = ( Base ‘ A ) $.
matinv.u $e |- U = ( Unit ‘ A ) $.
matinv.v $e |- V = ( Unit ‘ R ) $.
matinv.h $e |- H = ( invr ‘ R ) $.
matinv.i $e |- I = ( invr ‘ A ) $.
matinv.t $e |- .xb = ( .s ‘ A ) $.
matinv $p |- ( ( R e. CRing /\ M e. B /\ ( D ‘ M ) e. V ) ->
    ( M e. U /\ ( I ‘ M ) = ( ( H ‘ ( D ‘ M ) ) .xb ( J ‘ M ) ) ) )
```

Further impressive ATP proofs collected by us include theorems about integrals,[36] triangle inequality,[37] measure,[38] sums of vector spaces,[39] etc. These proofs typically take over one hundred steps in Metamath.

10 Hammer Tool

The mm-hammer tool is publicly available from our GitHub repository[40]. It packages the theorem proving, proof reconstruction and premise selection methods described above for the Metamath users. We provide there also an installer script that installs all the prerequisites (including Prover9 and Vampire).

11 Conclusion

We have developed the first translations of the Metamath set.mm library to the formats used by state-of-the-art higher-order and first-order automated theorem provers. Based on them, we have constructed several versions of a large new benchmark of 40556 mathematical ATP

[30] https://us.metamath.org/mpeuni/zringunit.html
[31] https://us.metamath.org/mpeuni/gzrngunit.html
[32] http://grid01.ciirc.cvut.cz/~mptp/mm_prf/
[33] https://us.metamath.org/mpeuni/xmulneg1.html
[34] http://grid01.ciirc.cvut.cz/~mptp/mm_prf/mmset12407_xmulneg1.p
[35] https://us.metamath.org/mpeuni/matinv.html
[36] https://us.metamath.org/mpeuni/ditgsplit.html
[37] https://us.metamath.org/mpeuni/isxmet2d.html
[38] https://us.metamath.org/mpeuni/sibfinima.html
[39] https://us.metamath.org/mpeuni/mapdlsm.html
[40] https://github.com/digama0/mm-hammer

problems based on set.mm . The initial evaluation of the ATPs is very encouraging. The strongest higher-order system (Zipperposition) proves 62.68% of the problems in 280s, and 57.99% of the problems in 140s. Even when using low (hammer-friendly) time limits, the higher-order ATPs are very useful, with E proving 50.18% of the problems in 10s. These are very encouraging results for providing ATP-based automation for the Metamath authors.

We have also developed the first version of a full hammer tool for Metamath and made it publicly available to the Metamath community. This includes mainly a proof reconstruction tool that imports the Prover9/IVY proof objects into Metamath. The tool already replays all 15k proofs that Prover9 can find when using human-based premises extracted from Metamath. Another component of the hammer is a real-time pipeline that translates Metamath user problems into first-order formats, and runs premise selectors and a portfolio of large-theory Vampires on the problems, followed by running Prover9/IVY on the Vampire-minimized problems when successful. The first version of the tool proves 30% of the Metamath theorems when running the ATPs on four premise selections in parallel for 60 seconds.

References

1 Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014. `doi:10.1007/s10817-013-9286-5`.

2 Ahmed Bhayat and Giles Reger. A combinator-based superposition calculus for higher-order logic. In *IJCAR (1)*, volume 12166 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2020.

3 Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 116–130. Springer, 2011. `doi:10.1007/978-3-642-22438-6_11`.

4 Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016. `doi:10.6092/issn.1972-5787/4593`.

5 Mario Carneiro. Metamath zero: Designing a theorem prover prover. In *Intelligent Computer Mathematics: 13th International Conference, CICM 2020, Bertinoro, Italy, July 26–31, 2020, Proceedings*, pages 71–88, Berlin, Heidelberg, 2020. Springer-Verlag. `doi:10.1007/978-3-030-53518-6_5`.

6 Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond.* Theses, École polytechnique, September 2015. URL: `https://hal.archives-ouvertes.fr/tel-01223502`.

7 Lukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *J. Autom. Reason.*, 61(1-4):423–453, 2018.

8 Martin Desharnais, Petar Vukmirović, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer, 2022. URL: `https://matryoshka-project.github.io/pubs/seventeen.pdf`.

9 Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark W. Barrett. Smtcoq: A plug-in for integrating SMT solvers into coq. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017. `doi:10.1007/978-3-319-63390-9_7`.

10 Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In *Certified Programs and Proofs (CPP'15)*, LNCS. Springer, 2015. `doi:10.1145/2676724.2693173`.

11 Allen Van Gelder and Geoff Sutcliffe. Extending the TPTP language to higher-order logic with automated parser generation. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *LNCS*, pages 156–161. Springer, 2006. doi:10.1007/11814771_15.

12 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. doi:10.1007/s00454-009-9148-4.

13 Jan Jakubův and Josef Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017. doi:10.1145/3018610.3018619.

14 Jan Jakubuv, Karel Chvalovský, Zarathustra Amadeus Goertzel, Cezary Kaliszyk, Mirek Olsák, Bartosz Piotrowski, Stephan Schulz, Martin Suda, and Josef Urban. Mizar 60 for mizar 50. *CoRR*, abs/2303.06686, 2023. doi:10.48550/arXiv.2303.06686.

15 Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015.

16 Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *Journal of Automated Reasoning*, 55(3):245–256, 2015. doi:10.1007/s10817-015-9330-8.

17 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013. doi:10.1007/978-3-642-39799-8_1.

18 Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine learning for Sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP 2013*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013. doi:10.1007/978-3-642-39634-2_6.

19 William McCune. Prover9 and Mace4, 2005–2010. URL: http://www.cs.unm.edu/~mccune/prover9/.

20 Norman D. Megill and David A. Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu Press, Morrisville, North Carolina, 2019. http://us.metamath.org/downloads/metamath.pdf.

21 Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *J. Autom. Reasoning*, 40(1):35–60, 2008. doi:10.1007/s10817-007-9085-y.

22 Lawrence C. Paulson and Jasmin C. Blanchette. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *Workshop on the Implementation of Logics (IWIL)*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2010. Invited talk. URL: http://www.easychair.org/publications/paper/62805.

23 John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001. URL: https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning.

24 Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer, 2013. doi:10.1007/978-3-642-45221-5_49.

25 Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2019. doi:10.1007/978-3-030-29436-6_29.

26 Josef Urban. BliStr: The Blind Strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. EasyChair, 2015. URL: http://www.easychair.org/publications/paper/BliStr_The_Blind_Strategymaker, doi:10.29007/8n7m.

27 Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50:229–241, 2013. doi:10.1007/s10817-012-9269-y.

28 Petar Vukmirovic, Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, Visa Nummelin, and Sophie Tourret. Making higher-order superposition work. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2021.