Hammering Higher Order Set Theory

Chad E. Brown[1], Cezary Kaliszyk[2][0000−0002−8273−6059], Martin Suda[1], and
Josef Urban[1][0000−0002−1384−1613]

[1] Czech Technical University in Prague, Czech Republic
`martin.suda@gmail.com,josef.urban@gmail.com`
[2] University of Melbourne, Australia and University of Innsbruck, Austria
`ckaliszyk@unimelb.edu.au`

Abstract. We use automated theorem provers to significantly shorten a
formal development in higher order set theory. The development includes
many standard theorems such as the fundamental theorem of arithmetic
and irrationality of square root of two. Higher order automated theorem
provers are particularly useful here, since the underlying framework of
higher order set theory coincides with the classical extensional higher
order logic of (most) higher order automated theorem provers, so no sig-
nificant translation or encoding is required. Additionally, many subgoals
are first order and so first order automated provers often suffice. We com-
pare the performance of different provers on the subgoals generated from
the development. We also discuss possibilities for proof reconstruction,
i.e., obtaining formal proof terms when an automated theorem prover
claims to have proven the subgoal.

1 Introduction

In this work, we first describe (Section 2) a formal development in higher order
set theory (using the Megalodon system) containing several well-known mathe-
matical theorems, including Conway's surreal numbers and 12 of the Freek100
theorems. We then use this development (Section 3) to create a large set (and
a benchmark) of higher order problems for automated theorem provers (ATPs),
evaluating multiple state-of-the-art ATPs on the benchmark. When an ATP is
able to solve one of the problems, we are able to replace part of the development
with a call to an ATP. This results in a reduction of the text of the development
to less than a half its original size. We also describe the related hammer-style
proof automation for Megalodon in Emacs, and discuss proof reconstruction
(Section 4).

2 Megalodon and Formalization of 12 Freek100 Theorems

2.1 Megalodon

We start with a formal development in the Megalodon system. Megalodon is a
fork of the Egal system [15] and is based on higher order Tarski-Grothendieck

set theory. The underlying logical framework is simply-typed intuitionistic higher order logic (with Curry Howard proof terms). In addition to the built-in type o of propositions we use one base type ι which should be interpreted as sets. The remaining types are function types $\alpha \to \beta$, as usual. Simply typed terms are constructed via the grammar

$$c \mid x \mid (s\,t) \mid (\lambda x.t) \mid (s \Rightarrow t) \mid (\forall x.t)$$

where c ranges over (typed) names (which may be primitives or definitions) and x ranges over (typed) variables. We assume the usual conventions about simple type theory (e.g., $s\,t\,u$ means $((s\,t)u)$ and $\lambda xy.t$ means $\lambda x.\lambda y.t$) and leave the reader to consult [15] if necessary. Although we have only included implication and universal quantification, there are (several) well-known impredicative definitions of other connectives and quantifiers [40,38,10]. Consequently, we will freely use $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\Leftrightarrow$, $\exists$ and $=$ below without further comment.

There are well-known ways to axiomatize set theory within a simply typed framework [36,25]. In our case we add a primitive ε as a choice function over ι and primitives of the set theory: $\in: \iota \to \iota \to o$, $\emptyset : \iota$, $\bigcup : \iota \to \iota$ (unary union operator), $\wp : \iota \to \iota$ (power set operator), $\mathtt{Repl} : \iota \to (\iota \to \iota) \to \iota$ (Fraenkel Replacement operator) and $\mathtt{UnivOf} : \iota \to \iota$. The unary operator $\mathtt{UnivOf}$ takes a set X and returns the least Grothendieck universe with X as a member. A Grothendieck universe is a transitive set closed under $\bigcup$, $\wp$ and $\mathtt{Repl}$. Typically a Grothendieck universe is also required to have an infinite set, so that it would be a model of Zermelo Fraenkel. Here we also consider V_ω (the set of all hereditarily finite sets) to be a Grothendieck universe. Applying $\mathtt{UnivOf}$ to the empty set yields the set of hereditarily finite sets. Since V_ω is infinite, we do not need an explicit axiom of infinity. In addition to the primitives, we assume the expected set theoretic axioms, including set extensionality. We also assume a higher order axiom of $\in$-induction from which Regularity can be proven. Finally we assume axioms of propositional and functional extensionality so that the underlying higher order logic is extensional. The full set of primitives and axioms can be found in [15], with the minor change that here we only assume ε at type ι rather than at every type.

2.2   Initial Formalization of 12 Freek100 Theorems

As one would expect, the development continues by making definitions and proving theorems. Wiedijk maintains a collection of 100 theorems that have become standard challenges for ITPs.[3] For our purposes we limit the development to definitions and theorems that are sufficient to prove 12 of these 100 theorems, mostly removing definitions or lemmas that are not dependencies of at least one of these 12, or a variant of one of the 12. Listed in order of appearance in the development, the selected 12 theorems are mathematical induction, Cantor's Theorem, Schroeder Bernstein, number of subsets of a (finite) set, infinitude of

---

[3] https://www.cs.ru.nl/~freek/100/

primes, Ramsey's Theorem, Bezout's Theorem, the greatest common divisor algorithm, the Fundamental Theorem of Arithmetic, non-denumerability of the reals, denumerability of the rationals and the irrationality of $\sqrt{2}$. In order to state these results, the development must include a representation of functions, natural numbers, integers, rationals and reals.

The first part of the development contains basic logical definitions and theorems. For example, conjunction ($\wedge$) is defined as $\lambda AB.\forall p.(A \Rightarrow B \Rightarrow p) \Rightarrow p$ and a theorem `andI` $\forall AB.A \Rightarrow B \Rightarrow A \wedge B$ is proven with the `exact` tactic and the proof term $\lambda A\ B\ a\ b\ p\ H.H\ a\ b$. Later in the development when a subgoal has a conclusion of the form $A \wedge B$, we can reduce the goal to the obvious two subgoals (A and B) using the tactic `apply andI`. This already introduces a minor amount of automation, as the `apply` tactic uses matching to extract the A and B from the subgoal in order to construct the partial proof term `andI` $A\ B\ D\ E$ where D and E will be determined by the subsequent subproofs.

The tenth theorem of the entire development (well before any of the 12 Freek100 theorems) is excluded middle `xm` : $\forall p.p \vee \neg p$. The proof follows the Diaconescu argument [39,34] using ε at ι. Before proving excluded middle, it does not make sense to use classical ATPs as hammers. For this reason, we only start considering the use of ATPs after the first 10 theorems of the development.

Throughout the development basic infrastructure is included (e.g., ordinals, ordered pairs and functions as sets) as needed, largely following [10], though the setting there was intuitionistic. As usual, the finite ordinals are used to represent natural numbers and provide the material to state and prove mathematical induction, both in its usual form and in the form of complete induction. The ordinal ω is defined to be the set of natural numbers, and we define a set to be finite if it is equipotent to a member of ω. This basic material is enough to state and prove five more of the 12 theorems. The development up to the proofs of these first 6 theorems consists of 6649 lines with 59 definitions and 314 theorems (including the 6).

2.3 Conway's surreal numbers

The remaining 6 of the 12 require either integers, rationals or real numbers. We construct these numbers (and more) using a specific set-theoretic representation of Conway's surreal numbers [18]. Surreal numbers have also been formalized in Mizar [35] using representations closer to Conway's description. Here we use a different representation, for reasons that will become clear. A surreal number can be uniquely represented by an ordinal length branch on an arbitrarily large binary tree. That is, each surreal number can be represented by an ordinal α and a function $f : \alpha \rightarrow \{0,1\}$ where $f(\beta) = 0$ indicates the left option at stage $\beta \in \alpha$ and $f(\beta) = 1$ indicates the right option at stage $\beta \in \alpha$. Instead of representing surreal numbers in precisely this way, we represent them using a set that remembers the ordinal α and which ordinals $\beta \in \alpha$ would take the left option and which would take the right option. To represent the left option at stage β, we define the set β' to be $\beta \cup \{\{1\}\}$. It is easy to see that β' is never an ordinal (since the set $\{1\}$ is not transitive). It is also to see that if $\beta' = \gamma'$ for ordinals β and γ,

then $\beta = \gamma$. We now represent the surreal number given by α and $f : \alpha \to \{0, 1\}$ by the set $\{\beta \in \alpha | f(\beta) = 1\} \cup \{\beta' | \beta \in \alpha, f(\beta) = 0\}$. It is easy to see that α and f can be recovered from this set. It is also easy to see that the surreal number given by α and $f : \alpha \to \{0, 1\}$ with $f(\beta) = 1$ for all $\beta \in \alpha$ is represented simply by the set α. That is, ordinals are already surreal numbers using this representation, and correspond to the branch of length α that chooses the right option at each step. Other examples of surreal numbers are -2 represented by $\{0', 1'\}$ and $\frac{1}{2}$ represented by $\{0, 1'\}$. One can define recursion operators on surreal numbers and use these to define the basic operations as described in [18] essentially proving the surreal numbers form an ordered field (with a proper class as its carrier). The development up to this point consists of 27316 lines with 116 definitions and 759 theorems, so that the primary surreal number part of the development consists of 20668 lines with 57 definitions and 445 theorems (almost half the development). In the next 2544 lines (with 6 new definitions and 57 new theorems), the set Z of integers can easily be defined (carved from the surreal number field) allowing us to state and prove Bezout's Theorem, the greatest common divisor algorithm and the Fundamental Theorem of Arithmetic. The next 11211 lines (with 11 new definitions and 137 new theorems) define the set R of real numbers (carved out of the surreals), proven to have the expected properties including uncountability. With 730 more lines, 2 new definitions and 8 new theorems, we can define the set Q of rational numbers as a subset of R and prove Q is countable. Note that by construction we have $\omega \subseteq Z \subseteq Q \subseteq R$, which was our motivation for the specific representation of surreal numbers as sets.

The only remaining theorem of the 12 is irrationality of $\sqrt{2}$. Conway describes how to recursively define the square root operation on nonnegative surreal numbers. We make this definition and prove the square root of nonnegative reals are nonnegative reals. The development ends with the proof that $\sqrt{2}$ is irrational, i.e., $\sqrt{2} \in R \setminus Q$. This part of the development used 3203 more lines with 4 new definitions and 38 new theorems.

In total the development contains 139 definitions and 999 theorems. The development is given as a single file with 45004 lines and 346152 characters, and the proofs are given in significant detail, with little automation.

3   Development using ATPs

3.1   Automation Tactic

We have extended Megalodon to allow the option of using the tactic `aby` ("automated by") with a list of dependencies to justify a subgoal.[4] Megalodon can check the file and produce problem files for ATPs. Since the underlying framework is simply typed higher order logic, we can directly translate the given dependencies and current subgoal into the TH0 TPTP format [3], a common format for higher order ATPs. Hence every use of the `aby` tactic will result in one TH0 problem

---

[4] Megalodon with the data/code discussed here is available at `https://github.com/MgUser36/megalodon`.

file. In addition, if the conclusion and the dependencies are in the first order fragment of higher order logic, we can translate the problem to the FOF TPTP format and call first order theorem provers. Note that we do not attempt to encode higher order logic into first order [7,37,28,29]. The use of the `aby` tactic is justified if either a higher order ATP can prove the TH0 file, or if a first order ATP can prove the FOF file.

3.2   Experiments

In order to identify subproofs that could be replaced by calls to ATPs, we used Megalodon to generate TH0 problem files for most tactic calls, determining the dependencies when the subproof is completed. Each problem corresponds to a sequence of text (from before the tactic is called to where the subproof is completed) that can be replaced if the problem file can be proven by an ATP. This resulted in 41738 higher order problem files.[5] With a 60 second timeout, we ran Vampire [4] (with two different portfolios), Zipperposition [2], E [43], Lash [12] and cvc5 [1]. The results are shown in Table 1. In total, 34172 (82%) of the 41738 subgoals could be replaced by ATP calls. Megalodon additionally produced 29880 first order problem files when the subgoal had a first order proof. We ran Vampire for 5 seconds on each of these problems and 60 seconds on a selection of the problems, resulting in 11179 subproofs that could be replaced by ATP calls, although only 11 of these subproofs were not already identified by the higher order problems.

Vampire (sledgehammer)	Vampire (ho)	Zipperposition	E	Lash	cvc5
32675	32474	31310	23866	14987	13238
78.3%	77.8%	75%	57.2%	35.9%	31.7%

Table 1. Higher order ATPs on premise-selected subgoals

Some ATP calls may result in replacing a larger part of text than others. For example, a subproof in the original development contained a line with three tactics:

```
apply In_irref delta. rewrite H2 at 2. exact Ldsa.
```

This line resulted in three problem files, corresponding to whether the `exact` could be replaced, the `rewrite` and the `exact` could be replaced, and whether all three could be replaced. In this case, all three could be replaced. That is, one

---

[5] The resulting large HO-TP benchmark is at `https://github.com/MgUser36/MegalodonATPBenchmark`.

problem solved by an ATP would allow us to replace the `exact` tactic by an `aby` call. A second problem solved by an ATP would allow us to replace both the `rewrite` and the `exact` tactic in the line above with an `aby` call. A third problem solved by an ATP would allow us to replace all three tactics on the line by an `aby` call. Obviously replacing all three tactics supersedes replacing either the last two or the last one alone, so it makes sense to find the ATP problems that allow us to replace as much text as possible. After filtering for the problems that allow the maximum amount of text to be replaced, the first two ATP problems above (corresponding to replacing `exact` alone or only replacing `rewrite` and `exact`) would be filtered out, in favor of only including the third ATP problem allowing the replacement of all three tactics.

Once we restrict to the problems that allow the largest texts to be replaced (along with some manual modifications), we were left with 3401 calls to an ATP and a development with 17435 lines and 159363 characters. This is roughly 46% the original size of the development.[6] The majority of the proofs in the development (765 of the 989 proofs considered for replacement) could be replaced simply by a single `aby` call. The remaining 224 proofs contained 2636 ATP calls in subproofs, with approximately 12 ATP calls per proof.

In order to justify the 3401 ATP calls, Megalodon produced 3401 TH0 problem files and 1618 FOF problem files. Table 2 shows the results of higher order ATPs on these files. We also called a recent first order version of Vampire was called on the FOF problems, with it solving 1322 (81.7%) of the 1618 first order problems. Each call was with a 60s timeout.

Vampire (sledgehammer)	Vampire (ho)	Zipperposition	E	Lash
3223 (94.8%)	3165 (93.1%)	2801 (82.4%)	2403 (70.7%)	1567 (46.1%)

Table 2. Higher order ATPs on the 3401 ATP problems

3.3   Examples

We consider a few examples to give an idea of what kinds of proofs can be replaced by calls to an ATP.[7]

Example 1 The largest text replaced by a call to an ATP (in this case, Lash) is the proof of the following theorem `PNoLt_tra`[8]:

```
Theorem PNoLt_tra : forall alpha beta gamma,
   ordinal alpha → ordinal beta → ordinal gamma →
   forall p q r:set → prop,
     PNoLt alpha p beta q → PNoLt beta q gamma r
        → PNoLt alpha p gamma r.
```

This is essentially the proof of transitivity of the (strict) ordering on surreal numbers. At the point in the development where this theorem is stated and proven, surreal numbers are not yet defined. However, the ordinal α with the predicate $p : \iota \to o$ determines a surreal number where p indicates for which $\alpha' \in \alpha$ the right option is taken. Likewise, β with q determines a surreal number and γ with r determines a surreal number. The definition of PNoLt α p β q corresponds to when the surreal number determined by α and p is less than the surreal number determined by β and q. This can happen in three ways:

1. $\alpha = \beta$ and there is some $\delta \in \alpha$ such that p and q agree up to δ, $\neg p\ \delta$ and $q\ \delta$. That is, the two surreal numbers use the same left and right options until δ at which point p chooses the left option and q chooses the right option.
2. $\alpha \in \beta$, p and q agree up to α and $q\ \alpha$. That is, the surreal number given by β and q is an extension of the sequence given by α and p, where the first new option (at step α) is the right option.
3. $\beta \in \alpha$, p and q agree up to β and $\neg p\ \beta$. That is, the surreal number given by α and p is an extension of the sequence given by β and q, where the first new option (at step β) is the left option.

To give concrete examples, suppose $\alpha = 1$ and $p\ 0$ holds. Then α and p correspond to the surreal number 1, since the sequence is of length 1 and chooses the right option at step 0. Suppose $\beta = 2$ and both $q\ 0$ and $q\ 1$ hold. Then β and q correspond to the surreal number 2. In this case PNoLt α p β q holds since $\alpha \in \beta$, p and q agree up to 1 (i.e., $p\ 0 \Leftrightarrow q\ 0$) and $q\ 1$ holds. On the other hand, if $\beta = 2$, $q\ 0$ and $\neg q\ 1$ (corresponding to the surreal number $\frac{1}{2}$) then we have PNoLt β q α p holds since $\alpha \in \beta$, p and q agree up to 1 and $\neg q\ 1$ holds. Finally, suppose $\alpha = 2$, $p\ 0$, $\neg p\ 1$, $\beta = 2$, $q\ 0$ and $q\ 1$, then PNoLt α p β q holds since $\alpha = \beta$, p and q agree up to 1, $\neg p\ 1$ and $q\ 1$.

   The manual proof of transitivity of PNoLt proceeds as follows. Suppose α, β and γ are ordinals and $p, q, r : \iota \to o$ are such that PNoLt α p β q and PNoLt β q γ r hold. We need to prove PNoLt α p γ r holds. We split into the three cases based on why PNoLt α p β q holds. In each of the three cases we split into three subcases based on why PNoLt β q γ r holds. The full manual proof uses 311 lines. Of these 311 lines, the first 5 are simply introducing the variables and hypotheses. Roughly 61 lines correspond to splitting into the 3 cases and the 9 subcases, introducing the relevant variables and hypotheses for each case and subcase. On average completing the proof in each of the 9 subcases requires about 27 lines. When using the hammer, the full proof is reduced to the following single use of aby:

```
aby and3I binintersectI binintersectE ordinal_Hered
    ordinal_trichotomy_or PNoEq_tra_ PNoEq_antimon_ PNoLtI1
    PNoLtI2 PNoLtI3 PNoLtE.
```

While the automated proof is much shorter, a reader examining the proof may
have trouble determining that the proof involves splitting into 3 cases and 9
subcases. The only hint is that the call to `aby` references `PNoLtE`, which is a
theorem that can be applied when we know `PNoLt` α p β q to split the current
goal into three cases.

Example 2 The second largest text replaced in a proof is at the end of the proof
of the following theorem `PNo_rel_split_imv_imp_strict_imv`[9]:

```
Theorem PNo_rel_split_imv_imp_strict_imv :
 forall L R:set → (set → prop) → prop,
  forall alpha, ordinal alpha → forall p:set → prop,
      PNo_rel_strict_split_imv L R alpha p
    → PNo_strict_imv L R alpha p.
```

Again, this theorem is about surreal numbers before committing to the particular
set theoretic representation of surreal numbers. Here L and R of type $\iota \rightarrow$
$(\iota \rightarrow o) \rightarrow o$ can be thought of as sets of surreal numbers (given as β and
$q : \iota \rightarrow o$). The conclusion `PNo_strict_imv` L R α p means that the surreal
number given by α and p is an intermediate value between L and R. That
is, `PNoLt` β q α p whenever L β q and `PNoLt` α p β q whenever R β q. The
hypothesis `PNo_rel_strict_split_imv` L R α p is a similar property about
the two extensions of α and p of length α^+.

The manual proof of this theorem was 240 lines. The proof using `aby` is 27
lines. No ATP was able to prove the theorem completely. (If an ATP had proven
the full theorem, we could replace the full proof by one call to `aby`.) The proof
using `aby` starts the same way as the manual proof, assuming we have L, R, an
ordinal α and some $p : \iota \rightarrow o$ satisfying `PNo_rel_strict_split_imv` L R α p.
ATPs can automatically prove α^+ (the successor of α) is also an ordinal, so this
subproof is replaced by `aby`. As part of the beginning of the proof we define p_0 to
be the predicate that holds for δ if p $\delta \wedge \delta \neq \alpha$ and we define p_1 to be the predicate
that holds for δ if p $\delta \vee \delta = \alpha$. ATPs can automatically prove $\neg p_0$ α and p_1 α, so
these two subproofs are replaced by `aby`. ATPs can also prove `PNoLt` α^+ p_0 α p
and `PNoLt` α p α^+ p_1, so these two subproofs are replaced by `aby`. At this point
we need to prove the conclusion `PNo_strict_imv` L R α p. The rest of the
manual proof requires 200 lines. ATPs can finish the rest of the proof and so
these 200 lines are replaced by a final call to `aby`. In the manual proof, the first
of the remaining 200 lines explicitly expands the definition of `PNo_strict_imv`
and reduces the conclusion to proving α and p are greater than everything in L

---

[9] `https://mgwiki.github.io/mgw_test/Part2.mg.html#PNo_rel_split_imv_imp_strict_imv`

and to proving α and p are less than everything in R. Meanwhile, when these last 200 lines are replaced by an `aby` call, there is no clear indication that the reader should expand the definition of `PNo_strict_imv` and check the two conjuncts. This is another example where replacing subproofs by `aby` may make the proof less readable by making it less explicit.

Example 3 In many cases, replacing details of a proof may make it easier to read a proof. Consider the following theorem `exp_SNo_nat_mul_add`[10]:

```
Theorem exp_SNo_nat_mul_add : forall x, SNo x →
  forall m, nat_p m → forall n, nat_p n →
    x ^ m * x ^ n = x ^ (m + n).
```

Here x^n is the operation raising a surreal number x to the nth power (for a natural number n). The goal is to prove $x^m \cdot x^n = x^{m+n}$, where multiplication and addition are over surreal numbers (which agree with multiplication and addition on natural numbers). The manual proof fixes x and m and proves that m is a surreal number (since it is natural). We then apply natural number induction, reducing the problem to proving the case for $n = 0$ and the inductive case. At this point in the manual proof a series of explicit (though obvious) arithmetical steps are required – including steps to change between addition on natural numbers and addition on surreal numbers. Meanwhile, ATPs can complete the proofs for the base case and inductive case, resulting in the following shorter proof:

```
let x. assume Hx. let m. assume Hm.
claim Lm: SNo m.
{ aby nat_p_SNo Hm. }
apply nat_ind.
- aby add_SNo_OR mul_SNo_oneR exp_SNo_nat_0 SNo_exp_SNo_nat
      Lm Hm Hx.
- aby add_nat_SR add_nat_p nat_p_omega omega_ordsucc
      add_nat_add_SNo mul_SNo_com mul_SNo_assoc exp_SNo_nat_S
      SNo_exp_SNo_nat Hm Hx.
Qed.
```

The original manual proof is longer (29 lines instead of 6 lines) and all the extra details are simply distracting. Omitting these steps arguably makes the proof easier to read.

3.4   Use in Emacs and a Full Hammer

We have also implemented a simple Emacs mode[11] for Megalodon and a command in it (`aby`.) which implements the "hammer" behavior, similar to the Sledgehammer command in Isabelle/JEdit. The command calls Megalodon to

---

[10] https://mgwiki.github.io/mgw_test/Part6.mg.html#exp_SNo_nat_mul_add
[11] https://github.com/MgUser36/megalodon/blob/main/mg-advice.el

```
Theorem exp_SNo_nat_mul_add : forall x, SNo x -> forall m, nat_p m -
> forall n, nat_p n -> x ^ m * x ^ n = x ^ (m + n).
let x. assume Hx. let m. assume Hm.
claim Lm: SNo m.
{ exact nat_p_SNo m Hm. }
apply nat_ind.
- prove x ^ m * x ^ 0 = x ^ (m + 0).
  rewrite exp_SNo_nat_0 x Hx.
  rewrite add_SNo_0R m Lm.
  exact mul_SNo_oneR (x ^ m) (SNo_exp_SNo_nat x Hx m Hm).
- let n. assume Hn: nat_p n.
  assume IHn: x ^ m * x ^ n = x ^ (m + n).
  prove x ^ m * x ^ (ordsucc n) = x ^ (m + ordsucc n).
  rewrite exp_SNo_nat_S x Hx n Hn.
  prove x ^ m * (x * x ^ n) = x ^ (m + ordsucc n).
  rewrite <- add_nat_add_SNo m (nat_p_omega m Hm) (ordsucc n) (omega
_ordsucc n (nat_p_omega n Hn)).
  prove x ^ m * (x * x ^ n) = x ^ (add_nat m (ordsucc n)).
  rewrite add_nat_SR m n Hn.
  prove x ^ m * (x * x ^ n) = x ^ (ordsucc (add_nat m n)).
  rewrite exp_SNo_nat_S x Hx (add_nat m n) (add_nat_p m Hm n Hn).
  prove x ^ m * (x * x ^ n) = x * x ^ (add_nat m n).
  rewrite add_nat_add_SNo m (nat_p_omega m Hm) n (nat_p_omega n Hn).
  prove x ^ m * (x * x ^ n) = x * x ^ (m + n).
  rewrite <- IHn.
  prove x ^ m * (x * x ^ n) = x * (x ^ m * x ^ n).
  rewrite mul_SNo_assoc (x ^ m) x (x ^ n) (SNo_exp_SNo_nat x Hx m Hm
) Hx (SNo_exp_SNo_nat x Hx n Hn).
  prove (x ^ m * x) * x ^ n = x * (x ^ m * x ^ n).
  rewrite mul_SNo_com (x ^ m) x (SNo_exp_SNo_nat x Hx m Hm) Hx.
  prove (x * x ^ m) * x ^ n = x * (x ^ m * x ^ n).
  symmetry.
  exact mul_SNo_assoc x (x ^ m) (x ^ n) Hx (SNo_exp_SNo_nat x Hx m H
m) (SNo_exp_SNo_nat x Hx n Hn).
Qed.
```

Fig. 1. Original Megalodon Proof.

create a TH0 file which includes the whole previous development. Then it calls Vampire with a schedule constructed for the Sledgehammer division of the CASC competition.[12] If successful, the names of the used axioms are translated back to the Megalodon syntax and the aby call using them is automatically inserted into the current buffer. The operation is shown in Figures 1, 2, and 3.

---

[12] The schedule was constructed according to https://tptp.org/CASC/J11/
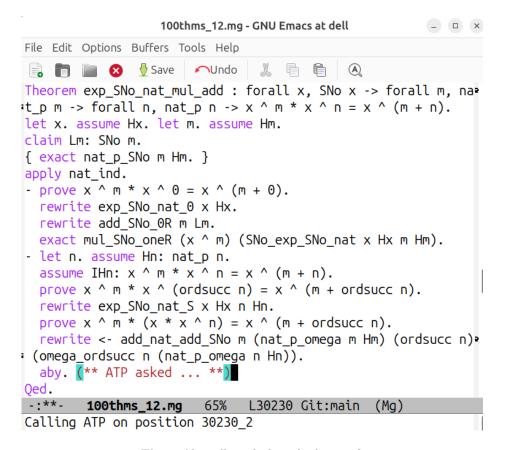SystemDescriptions.html#SnakeForV4.7---1.0

Fig. 2. Aby call invoked inside the proof.

To test this "full hammer" performance, we generated corresponding ATP problems each time a tactic was used in the original development. This resulted in 51243 TH0 problems.[13] Note that this is larger than the 41738 premise selected problem files. This difference arises from several factors, which we do not discuss in detail here. We called Vampire with the two portfolios for 60s. With the sledgehammer schedule, Vampire proved 29256 problems (57.1%). With the default higher-order (ho) schedule, Vampire proved 28811 problems (56.2%). The union of the two sets was 30473 problems (59.5%).

4 Proof Reconstruction

With the original development, complete proof terms are constructed by Megalodon from the given proof scripts. Once ATPs are used, Megalodon is essentially

---

[13] Also in our benchmark: `https://github.com/MgUser36/MegalodonATPBenchmark/tree/main/chainy`.

Fig. 3. Successful aby call with the axiom names inserted.

behaving like the Naproche system [19]. That is, when an ATP completes the proof of a subgoal, we trust that there is, indeed, a proof. Ultimately it would be more satisfying to do *proof reconstruction* so that we still obtain complete proof terms even when some subgoals are completed by ATPs.

A common technique [33] is to use an ATP to prune the dependencies and then use an internal search procedure (like Metis [28] for Isabelle) to create the appropriate ITP proof. Megalodon currently has no significant internal search procedure that could be used for this purpose. A more optimal state of affairs would be if ATPs returned sufficiently detailed proof objects that could be independently checked. Prover9 [32] is a rare example of an ATP that has provided explicit proof objects for decades, though it is limited to clausal proofs. This is used for proof reconstruction e.g. in the Metamath hammer [17].

4.1   Using Vampire with the Deducti Output

Due to recent work on Vampire [31] one can request proof output (for first order problems) that can be checked by Dedukti [22], with some limitations noted below.

We consider a very simple example. Suppose we are in a proof where we have a set α and a local assumption Ha that α is an ordinal (i.e., the proposition `ordinal` α). We have a local subgoal to prove $\alpha+$ is an ordinal (i.e., `ordinal (ordsucc` α)). The relevant previous theorem is `ordinal_ordsucc`: $\forall\alpha.$`ordinal` $\alpha \Rightarrow$ `ordinal (ordsucc` α). This is a first order subgoal and Vampire can easily prove it. The simplest Megalodon proof term for this subgoal would be `ordinal_ordsucc` α Ha. In Megalodon this part of the proof[14] looks as follows:

```
claim Lsa: ordinal (ordsucc alpha).
{ exact ordinal_ordsucc alpha Ha. }
```

Replacing the subproof with a call to an ATP simply changes the proof to[15]

```
claim Lsa: ordinal (ordsucc alpha).
{ aby ordinal_ordsucc Ha. }
```

We examine parts of the Dedukti output of Vampire. The previous theorem `ordinal_ordsucc` is given as an axiom (with some changes to the name for TPTP compliance) and this is reflected in the following declared axiom in the Dedukti output:

```
{|axiom_ordinal_5Fordsucc9|}:
  Prf (forall iota
          (0 : El iota =>
               (imp ({|ordinal|} 0)
                    ({|ordinal|} ({|ordsucc|} 0))))).
```

Likewise the local assumption that α is an ordinal is reflected in the following declared axiom:

```
{|axiom_c_Ha16|}: Prf ({|ordinal|} {|alpha|}).
```

The negated conclusion is also given as a declared axiom:

```
{|axiom_18|}: Prf (not ({|ordinal|} ({|ordsucc|} {|alpha|}))).
```

In principle the rest of the proof should be given as a sequence of Dedukti definitions, but in practice some of the steps to produce clausal form are left unjustified at the moment. If we skip forward past the unjustified steps (which are minor in this case), we can start from three Dedukti items with types corresponding to the three assumptions above, but in clausal form. The two resolution

---

[14] https://github.com/MgUser36/megalodon/blob/main/examples/form100/100thms_12.mg#L9964

[15] https://github.com/MgUser36/megalodon/blob/main/examples/hammer/100thms_12_h.mg#L3681

steps corresponds to two Dedukti definitions. In principle the Dedukti representation could be translated into Megalodon as follows: One begins the subproof by applying double negation, so that one may assume the negated conclusion and reduce to proving false. Then each Dedukti definition would correspond to a subgoal with a proof term translated from its definition. The last Dedukti definition should give a proof of false (corresponding to the empty clause in the resolution proof).

This translation is left as future work here. Another general idea for distributing ITP proofs – which includes proof reconstruction – is outsourcing to the (Megalodon-related) Proofgold network [13], which newly includes also the basis for a lightning network [14].

5   Related Work

ITP/ATP hammers and their various components have been developed and successfully used for over two decades [26,21,27,41,42,33,37,6,5,30,24,8,20,17], in systems such as Isabelle, HOL, Mizar, HOL Light, Coq, and Metamath. A comprehensive overview of the topic is given in [9]. Similarly, a number of large problem sets and benchmarks for higher-order ATPs have been extracted from various ITP libraries in the recent years [16,11,23].

6   Conclusion and Future Work

We have shown that calls to ATPs can be used to replace over half the content of formal proofs in a mathematical development in higher order set theory. The original development (without automation) can be used to generate tens of thousands of higher order ATP problems and this benchmark can be used to judge the performance of modern higher order ATPs, currently showing Vampire as the clear leader. After replacing as much text as possible with calls to ATPs, we obtain 3401 higher order ATP problems that are, in some sense, at the frontier of what current higher order ATPs can solve.

We have also discussed options for proof reconstruction, such as the emerging Dedukti proof format newly implemented in Vampire, and Megalodon's connection to the Proofgold blockchain, which targets decentralized and distributed proof development. Another interesting avenue would be the use of SMT solvers, specifically when the subgoal is about integers or real numbers. Of course, SMT solvers should only be called once integers (or reals) are defined (along with the corresponding operations) and the basic properties have been proven.

References

1. Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. `doi:10.1007/978-3-030-99524-9\_24`.

2. Alexander Bentkamp, Jasmin Blanchette, Simon Cruanes, and Uwe Waldmann. Superposition for lambda-free higher-order logic. *Log. Methods Comput. Sci.*, 17(2), 2021. URL: `https://lmcs.episciences.org/7349`.

3. Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 - the core of the TPTP language for higher-order logic. In *IJCAR 2008*, volume 5195 of *LNCS*, pages 491–506. Springer, 2008.

4. Ahmed Bhayat and Martin Suda. A higher-order vampire (short paper). In Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt, editors, *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I*, volume 14739 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2024. `doi:10.1007/978-3-031-63498-7\_5`.

5. Jasmin Christian Blanchette. *Automatic Proofs and Refutations for Higher-Order Logic*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2012.

6. 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.

7. Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. *Log. Methods Comput. Sci.*, 12(4), 2016. `doi:10.2168/LMCS-12(4:13)2016`.

8. Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning*, 57(3):219–244, 2016. URL: `http://dx.doi.org/10.1007/s10817-016-9362-8`, `doi:10.1007/s10817-016-9362-8`.

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

10. Chad E. Brown. Reconsidering Pairs and Functions as Sets. *Journal of Automated Reasoning*, 55(3):199–210, Oct 2015.

11. Chad E. Brown, Thibault Gauthier, Cezary Kaliszyk, Geoff Sutcliffe, and Josef Urban. GRUNGE: A grand unified ATP challenge. 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 123–141. Springer, 2019. `doi:10.1007/978-3-030-29436-6\_8`.

12. Chad E. Brown and Cezary Kaliszyk. Lash 1.0 (system description). In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning*

- *11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 350–358. Springer, 2022. `doi:10.1007/978-3-031-10769-6\_21`.

13. Chad E. Brown, Cezary Kaliszyk, Thibault Gauthier, and Josef Urban. Proof-gold: Blockchain for formal methods. In Zaynah Dargaye and Clara Schneidewind, editors, *4th International Workshop on Formal Methods for Blockchains, FMBC@CAV 2022, August 11, 2022, Haifa, Israel*, volume 105 of *OASIcs*, pages 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: `https://doi.org/10.4230/OASIcs.FMBC.2022.4`, `doi:10.4230/OASICS.FMBC.2022.4`.

14. Chad E. Brown, Cezary Kaliszyk, and Josef Urban. Payment channels with proofs. Available online., 2025. URL: `http://grid01.ciirc.cvut.cz/~mptp/FMBC_2025_paper_6.pdf`.

15. Chad E. Brown and Karol Pąk. A tale of two set theories. In Cezary Kaliszyk, Edwin C. Brady, Andrea Kohlhase, and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings*, volume 11617 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2019.

16. Chad E. Brown and Josef Urban. Extracting higher-order goals from the mizar mathematical library. In Michael Kohlhase, Moa Johansson, Bruce R. Miller, Leonardo de Moura, and Frank Wm. Tompa, editors, *Intelligent Computer Mathematics - 9th International Conference, CICM 2016, Bialystok, Poland, July 25-29, 2016, Proceedings*, volume 9791 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2016. `doi:10.1007/978-3-319-42547-4\_8`.

17. Mario Carneiro, Chad E. Brown, and Josef Urban. Automated theorem proving for Metamath. In Adam Naumowicz and René Thiemann, editors, *14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland*, volume 268 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.ITP.2023.9`, `doi:10.4230/LIPICS.ITP.2023.9`.

18. John H. Conway. *On numbers and games, Second Edition*. A K Peters, 2001.

19. Marcos Cramer. The Naproche system: Proof-checking mathematical texts in controlled natural language. In Hermann Cölfen, Ulrich Schmitz, and Bernhard Schröder, editors, *Sprache und Datenverarbeitung*, volume 38, pages 9–33. Universitätsverlag Rhein-Ruhr, Duisburg, 2014.

20. Lukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. *J. Autom. Reasoning*, 61(1-4):423–453, 2018. `doi:10.1007/s10817-018-9458-4`.

21. Ingo Dahn. Interpretation of a Mizar-like logic in first-order logic. In Ricardo Caferra and Gernot Salzer, editors, *FTP (LNCS Selection)*, volume 1761 of *LNCS*, pages 137–151. Springer, 1998.

22. The Dedukti logical framework. `https://deducteam.github.io`.

23. Martin Desharnais, Petar Vukmirovic, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer. In June Andronick and Leonardo de Moura, editors, *13th International Conference on Interactive Theorem Proving, ITP 2022, August 7-10, 2022, Haifa, Israel*, volume 237 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: `https://doi.org/10.4230/LIPIcs.ITP.2022.8`, `doi:10.4230/LIPICS.ITP.2022.8`.

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

25. Michael Gordon. Set theory, higher order logic or both? In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics, TPHOLs'96*, volume 1125 of *LNCS*, pages 191–201. Springer, 1996. `doi:10.1007/BFb0105405`.

26. Joe Hurd. Integrating Gandalf and HOL. In Yves Bertot, Gilles Dowek, André Hirschowitz, C. Paulin, and Laurent Théry, editors, *TPHOLs*, volume 1690 of *LNCS*, pages 311–322. Springer, 1999.

27. Joe Hurd. An LCF-style interface between HOL and first-order logic. In Andrei Voronkov, editor, *CADE*, volume 2392 of *LNCS*, pages 134–138. Springer, 2002.

28. Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003. URL: `http://techreports.larc.nasa.gov/ltrs/PDF/2003/cp/NASA-2003-cp212448.pdf`.

29. Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014. `doi:10.1007/s10817-014-9303-3`.

30. Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014. URL: `http://dx.doi.org/10.1007/s10817-014-9303-3`, `doi:10.1007/s10817-014-9303-3`.

31. Anja Petković Komel, Michael Rawson, and Martin Suda. Case study: Verified vampire proofs in the lambdapi-calculus modulo, 2025. URL: `https://arxiv.org/abs/2503.15541`, `arXiv:2503.15541`.

32. W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

33. Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *Journal of Applied Logic*, 7(1):41–57, 2009. Special Issue: Empirically Successful Computerized Reasoning. URL: `https://www.sciencedirect.com/science/article/pii/S1570868307000626`, `doi:10.1016/j.jal.2007.07.004`.

34. N. D. Goodman and J. Myhill. Choice Implies Excluded Middle. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 24:461, 1978.

35. Karol Pak and Cezary Kaliszyk. Conway normal form: Bridging approaches for comprehensive formalization of surreal numbers. In Yves Bertot, Temur Kutsia, and Michael Norrish, editors, *15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia*, volume 309 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: `https://doi.org/10.4230/LIPIcs.ITP.2024.29`, `doi:10.4230/LIPICS.ITP.2024.29`.

36. Lawrence C. Paulson. Set theory for verification: I. from foundations to functions. *Journal of Automated Reasoning*, 11:353–389, 1993.

37. Lawrence C. Paulson. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Renate A. Schmidt, Stephan Schulz, and Boris Konev, editors, *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010*, volume 9 of *EPiC*, pages 1–10. EasyChair, 2010. URL: `https://doi.org/10.29007/tnfd`, `doi:10.29007/TNFD`.

38. Dag Prawitz. *Natural deduction: a proof-theoretical study*. Dover, 2006.

39. R. Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51:176–178, 1975.

40. Bertrand Russell. *The Principles of Mathematics*. Cambridge University Press, 1903.

41. J. Urban. Translating Mizar for First Order Theorem Provers. In A. Asperti, B. Buchberger, and J.H. Davenport, editors, *Proceedings of the 2nd International Conference on Mathematical Knowledge Management*, number 2594 in LNCS, pages 203–215. Springer, 2003.

42. Josef Urban. MPTP – Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004. `doi:10.1007/s10817-004-6245-1`.

43. Petar Vukmirović, Jasmin Blanchette, and Stephan Schulz. Extending a high-performance prover to higher-order logic. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II*, volume 13994 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2023. `doi:10.1007/978-3-031-30820-8\_10`.