Hammering Higher Order Set Theory

Chad E. Brown, Cezary Kaliszyk, Martin Suda, Josef Urban

Czech Technical University in Prague, University of Melbourne, University of Innsbruck

September 2025

Outline

Motivation

- Use automated theorem provers (ATPs) to shorten formal developments in higher order set theory.
- Development includes well-known theorems: fundamental theorem of arithmetic, irrationality of $\sqrt{2}$, surreal numbers, etc.
- Higher order ATPs fit well with higher order set theory: minimal translation needed.
- Many subgoals are first-order: FO provers often suffice.

Goals

- Benchmark higher order ATPs on realistic higher-order mathematical problems.
- Replace large parts of proof scripts with automated calls.
- Study proof reconstruction for ATP-generated proofs.

Megalodon System

- Fork of the Egal system, based on higher-order Tarski-Grothendieck set theory.
- Logical framework: simply-typed intuitionistic HOL with Curry-Howard proofs.
- One base type ι (sets) + function types $\alpha \to \beta$.
- Built-in set theory primitives: $\in$, $\emptyset$, $\bigcup$, $\mathcal{P}$, Replacement, Grothendieck universes.

Formalization of 12 Freek100 Theorems

- Selected 12 classical theorems (e.g., induction, Cantor's theorem, infinitude of primes) from the Freek 100 List.
- Required infrastructure: ordinals, natural numbers, integers, rationals, reals.
- Used Conway's surreal numbers to uniformly represent $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$.
- Total of 999 theorems.

Automation via aby Tactic

- New tactic aby with dependencies $\Rightarrow$ call to ATP.
- Translate subgoal + dependencies to TPTP TH0 (HOL) or FOF (FOL) format.
- HO problems: sent to Vampire, Zipperposition, E, Lash, cvc5.
- FO problems: sent to Vampire.

Experimental Setup

- Generated 41,738 higher-order problems from development.
- Timeout: 60s for HO ATPs, 5s (and some 60s) for FO ATPs.
- Benchmarked multiple ATPs on premise-selected subgoals.

ATP Success Rates

Prover	Solved	%
Vampire (sledgehammer)	32,675	78.3%
Vampire (HO)	32,474	77.8%
Zipperposition	31,310	75.0%
E	23,866	57.2%
Lash	14,987	35.9%
cvc5	13,238	31.7%

Impact on Development Size

- Original: 45,004 lines, 346,152 characters.
- After automation: 17,435 lines, 159,363 characters.
- Reduction: ∼46% of original size.
- Most proofs replaced by single `aby` calls.

Example 1: Transitivity of Surreal Number Order

- Original manual proof: 311 lines, 3 case splits $\times$ 3 subcases.
- Automated: Single `aby` call.
- Shorter but less explicit $\Rightarrow$ readability trade-off.

Example 1: Transitivity of Surreal Number Order

```
Definition PNoLt : set -> (set -> prop) -> set -> (set -> prop) -> prop
 := fun alpha p beta q =>
        PNoLt_ (alpha :/\: beta) p q
     \/ alpha :e beta /\ PNoEq_ alpha p q /\ q alpha
     \/ beta :e alpha /\ PNoEq_ beta p q /\ ~p beta.

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

Example 2: Intermediate Value Property

```
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.
```

- Original proof: 240 lines.
- Automated proof: 27 lines.
- Some parts still needed manual structure before automation.

Example 2: Intermediate Value Property

```
let L R.
let alpha.
assume Ha: ordinal alpha.
let p.
assume Hp: PNo_rel_strict_split_imv L R alpha p.
claim Lsa: ordinal (ordsucc alpha).
{ aby ordinal_ordsucc Ha. }
set p0 : set -> prop := fun delta => p delta /\ delta <> alpha.
set p1 : set -> prop := fun delta => p delta \/ delta = alpha.
apply Hp.
assume Hp0: PNo_rel_strict_imv L R (ordsucc alpha) p0.
assume Hp1: PNo_rel_strict_imv L R (ordsucc alpha) p1.
apply Hp0.
assume Hp0a: PNo_rel_strict_upperbd L (ordsucc alpha) p0.
assume Hp0b: PNo_rel_strict_lowerbd R (ordsucc alpha) p0.
apply Hp1.
assume Hp1a: PNo_rel_strict_upperbd L (ordsucc alpha) p1.
assume Hp1b: PNo_rel_strict_lowerbd R (ordsucc alpha) p1.
claim Lnp0a: ~p0 alpha.
{ assume H10. aby H10. }
claim Lp1a: p1 alpha.
{ aby. }
claim Lap0p: PNoLt (ordsucc alpha) p0 alpha p.
{ aby ordsuccI2 PNoEq_sym_ PNoLtI3 PNo_extend0_eq Lnp0a. }
claim Lapp1: PNoLt alpha p (ordsucc alpha) p1.
{ aby ordsuccI2 PNoLtI2 PNo_extend1_eq Lp1a. }
aby dneg binintersectE ordsuccI1 ordsuccI2 ordsuccE ordinal_Hered PNoEq_ref_
    PNoEq_sym_ PNoEq_tra_ PNoEq_antimon_ PNoLtI2 PNoLtI3 PNoLtE PNoLt_irref
```

Example 3: Exponentiation Law for Naturals

$$x^m \cdot x^n = x^{m+n}$$

```
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.
{ aby nat_p_SNo Hm. }
apply nat_ind.
- aby add_SNo_0R 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.
```

- Original: 29 lines of explicit arithmetic manipulations.
- Automated: 6 lines.
- Omission of trivial steps improves readability.

Emacs Integration for Hammering

- Simple Emacs mode for Megalodon with `aby.` command.
- Generates TPTP problem, calls ATP (e.g., Vampire), and inserts `aby` proof.
- Inspired by Isabelle's `sledgehammer`.

Hammering in Action

```
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 Hm) (SNo_exp_SNo_nat x Hx n Hn).
Qed.
```

After Hammer Invocation

100thms_12.mg - GNU Emacs at dell

```
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)).
  aby. (** ATP asked ... **)
Qed.

Theorem exp_SNo_nat_mul_add' : forall x, SNo x -> forall m n :e omega, x ^ m * x ^ n = x ^ (m + n).
let x. assume Hx. let m. assume Hm. let n. assume Hn.
exact exp_SNo_nat_mul_add x Hx m (omega_nat_p m Hm) n (omega_nat_p n Hn).
Qed
-:**-  100thms_12.mg   65%   L30231 Git:main   (Mg)
Calling ATP on position 30231_2
```

Aby Call Inserted with Dependencies

Jul 25 18:24

100thms_12.mg - GNU Emacs at dell

File Edit Options Buffers Tools Help

Save Undo

```
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)).
  aby mul_SNo_com_3_0_1 IHn SNo_exp_SNo_nat exp_SNo_nat_S Hx nat_p_omega add_nat_SR add_nat_add_SNo a
dd_nat_p Hm Hn.
Qed.

Theorem exp_SNo_nat_mul_add' : forall x, SNo x -> forall m n :e omega, x ^ m * x ^ n = x ^ (m + n).
let x. assume Hx. let m. assume Hm. let n. assume Hn.
exact exp_SNo_nat_mul_add x Hx m (omega_nat_p m Hm) n (omega_nat_p n Hn).
```

```
-:**-  100thms_12.mg   65%   L30231 Git:main  (Mg)
% Success in time 7.403 s
```

Proof Reconstruction

- ATPs can prune dependencies; internal prover (like Metis) could reconstruct proof terms.
- Vampire now outputs Dedukti-checkable proofs for FOL problems.
- Potential to translate back into Megalodon proofs.

Conclusion

- ATPs can replace large parts of higher order set theory developments.
- Significant compression: $> 50\%$ proofs automated.
- Vampire currently best-performing HO ATP on benchmark.
- Future: Better proof reconstruction, SMT integration, decentralized proof sharing.

Acknowledgments

Supported by

- the Czech Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902,
- the ERC PoC grant *FormalWeb3* no. 101156734,
- Amazon Research Awards,
- and the Czech Science Foundation grant no. 25-17929X.