# The Equational Theories Project

The project was launched in 2024 by Fields medalist Terence Tao.

It aims at exploring equational theories of magmas.
A *magma* is a set equipped with a binary operation $\diamond$.

# The Equational Theories Project

The project was launched in 2024 by Fields medalist Terence Tao.

It aims at exploring equational theories of magmas.
A *magma* is a set equipped with a binary operation $\diamond$.

The project initially focused on problems of the form $A \Longrightarrow C$,
where axiom $A$ and conjecture $C$ are $\forall$-quantified equations with at most four $\diamond$'s.

# A Tricky Equational Problem

$$\begin{aligned} &\left(\forall x, y, z.\ x = x \diamond (y \diamond ((z \diamond x) \diamond y))\right) \\ \Longrightarrow &\left(\forall x, y, z.\ x = x \diamond (y \diamond (z \diamond (x \diamond z)))\right) \end{aligned}$$

# The Vampire-Generated Proof (62 Steps)

```
1.  !   [X0,X1,X2] : op(X0,op(X1,op(op(X2,X0),X1))) = X0 [input]
2.  !   [X0,X1,X2] : op(X0,op(X1,op(X2,op(X0,X2)))) = X0 [input]
3.  ?   [X0,X1,X2] : op(X0,op(X1,op(X2,op(X0,X2)))) != X0 [negated conjecture 2]
4.  ?   [X0,X1,X2] : op(X0,op(X1,op(X2,op(X0,X2)))) != X0 [ennf transformation 3]
5.  ?   [X0,X1,X2] : op(X0,op(X1,op(X2,op(X0,X2)))) != X0 <=> sK0 = op(sK0,op(sK1,op(sK2,op(sK0,sK2)))) [choice axiom]
6.  sK0 != op(sK0,op(sK1,op(sK2,op(sK0,sK2)))) [skolemisation 4,5]
7.  op(X0,op(X1,op(op(X2,X0),X1))) = X0 [cnf transformation 1]
8.  sK0 != op(sK0,op(sK1,op(sK2,op(sK0,sK2)))) [cnf transformation 6]
9.  op(X1,op(op(X2,X0),X1)) = op(op(X1,op(op(X2,X0),X1)),op(X0,X0)) [superposition 7,7]
10. op(X1,op(op(X2,op(op(X3,op(X0,X1)),X2)),op(X0,X1))) = X1 [superposition 7,7]
11. op(op(X2,X0),X1) = op(op(X2,X0),X1),op(op(X3,op(X0,X3)),op(X1,op(op(X2,X0),X1)))) [superposition 10,7]
14. op(X7,op(op(op(op(X8,op(op(X9,op(X10,op(X11,op(op(X6,X7)),X8))),X9)),X10),X11),X6)) = op(X10,op(X5,op(X6,X7))),op(X5,op(X6,X7))) = X7 [superposition 10,10]
15. op(X1,op(op(X2,X0),X1)) = op(op(X1,op(op(X2,X0),X1)),op(op(X3,op(op(X4,X0),X3)),X0)) [superposition 10,7]
16. op(X3,op(X0,X3)) = op(op(X1,op(X0,X3)),op(X4,op(op(X1,op(op(X2,X0),X1)),X4))) [superposition 9,7]
21. op(X15,op(op(X11,op(op(X12,X11),X11),X15)) = op(op(X15,op(op(X11,op(op(X12,X13),X11),X11),X15)),op(X16,op(op(X14,op(X13,X14)),X16))) [superposition 9,8]
30. op(X2,op(X3,op(op(X1,X0),X0),X2)) = op(op(X2,op(op(X3,op(op(X1,X0),X0),X2)),op(X0,op(X2,X0)),op(X0,op(op(X1,X0),X0))) [superposition 9,9]
34. op(X15,X16) = op(op(X15,X16),op(op(X17,op(X13,op(op(X14,X15),X13)),X17)),op(X16,op(X15,X16))))) [superposition 10,9]
35. op(X21,op(X28,X21)) = op(op(X21,op(X28,X21)),op(op(X22,op(X23,op(op(X24,X22),X23))),X22)),op(X18,op(X19,X20),X18))),X22)),op(X18,op(X19,X20),X18))))) [superposition 10,9]
37. op(X1,op(op(X2,X0),X1)),X0) = op(op(X1,op(op(X2,X0),X1)),X0),op(X3,op(X4,op(op(X5,X0),X4),X2))) [superposition 10,7]
53. op(X4,op(X2,X4)) = op(op(X4,op(X2,X4)),op(op(X5,op(X2,X2)),op(X6,op(op(X4,X2),X6)))) [superposition 10,9]
104. op(op(X3,X0),X4) = op(op(op(X3,X0),X4),op(op(op(X1,op(op(X2,X0),X1)),X0),op(X4,op(op(X3,X0),X4)))) [superposition 11,7]
232. op(X35,op(X32,X35)) = op(op(X35,op(X32,X35)),op(op(op(X33,op(X34,X32)),X32),op(X30,op(op(X31,X32),X30)))) [superposition 19,15]
237. op(X63,op(op(X66,op(op(X61,op(op(X62,X63),X61)),X66)),X64),op(op(X64,op(op(X65,X62),X64)),X63))) = X63 [superposition 10,15]
269. op(X2,X4) = op(op(X2,X4),op(op(op(X3,op(X2,X3)),op(X0,op(X1,X2),X0)),op(X4,op(X2,X4))) [superposition 10,9]
741. op(X2,op(op(op(X3,op(X2,X3)),op(X0,op(op(X1,X2),X0))),op(op(X4,op(op(X5,X2),X4)),X2))) = X2 [superposition 37,9]
1896. op(X38,op(op(op(X8,X9),X8)),X10)) = op(op(X38,op(op(op(X8,X9),X10)),op(op(op(X8,X9),X9),X10)),op(op(op(X8,X9),X9),op(X8,X9)))),op(X9,op(op(X8,X9),X9))))) [superposition 29,11]
1791. op(op(X40,op(op(X41,op(X40,X40)),X40)),X43) = op(op(op(X40,op(X41,X40),X40),X40),X43),op(op(X39,op(op(op(X40,op(X41,X40),X40),X39)),op(X43,op(op(X40,op(X41,X40),X40),X43))))) [superposition 269,21]
1792. op(X48,op(op(X45,op(op(X46,X45),X45)),X48)) = op(op(X48,op(op(X46,X45),X48)),op(X44,op(X45,op(op(X46,X45),X45)),X48))) [superposition 53,21]
1919. op(X40,op(op(X41,X40),X40)),X43) = op(op(X40,op(op(X41,X40),X40)),X43),op(X39,op(op(X40,op(op(X41,X40),X40)),X39))) [backward demodulation 1791,1792]
2162. op(X61,op(X58,X61)) = op(op(X61,op(X58,X61)),op(X58,X61)),op(X62,op(op(X58,op(op(X59,op(op(X60,X58),X59)),X58),X62))) [forward demodulation 2162,2]
2615. op(X68,op(op(op(X69,op(op(X67,X68),X69)),X68),X68)),X68) = op(op(X68,op(op(X67,X68),X68)),X68),op(X64,op(X65,X68),X64)),X68))) = X68 [superposition 14,237]
2869. op(X29,op(X27,X29)) = op(op(X29,op(X27,X29)),op(op(X27,op(op(X28,X27),X27)),op(X27,op(op(X27,op(op(X28,X27),X27)),X27)))) [superposition 2326,20]
2896. op(X48,X39) = op(op(X40,X39),op(op(X41,op(op(X40,op(op(X42,op(op(X43,X40),X42)),X40)),X41)),op(X39,op(X40,X39)))) [superposition 7,2326]
5621. op(X36,op(op(X37,op(op(X38,X31),X37)),X36)) = op(op(X36,op(op(X37,op(op(X38,X31),X37)),X36)),op(op(op(X33,X31),X32),X31),op(X34,op(op(X35,X31),X34))),op(X36,op(X31,X30)))) [superposition 21,232]
5844. op(X27,X29) = op(op(X27,X29),op(op(op(X27,op(op(X28,X27),X27)),op(X27,op(op(X27,op(op(X28,X27),X27)),X27))),op(X29,op(X27,X29)))) [superposition 2896,29]
5872. op(X0,op(op(X3,op(op(X8,op(op(X6,op(op(X5,X6),X4)),X3)),X1)),X2)),op(X1,op(op(X2,X1)),X0) = X0 [superposition 2896,7]
6230. op(X59,op(op(op(X60,X55),X59)),X55) = op(op(op(X59,op(op(X60,X55),X59)),X55),op(op(op(X56,op(op(X54,X55),X56)),op(X57,op(op(X58,op(X54,X55)),X57)),X57)),op(X53,op(op(X54,X55),X53)))) [superposition 37,53]
6242. op(X0,op(op(X1,X0),X0)) = op(op(X0,op(X1,X0)),X0),op(op(X2,op(op(X3,X0),X2)),X0)) [superposition 1810,19]
10389. op(X985,op(op(op(X983,op(op(X984,X983),X983)),op(X985,op(op(X986,op(op(X987,X985),X986)),X985))),X985)) = X985 [superposition 5872,1209]
10945. op(op(X1209,op(op(op(X1210,X1209),X1209)),X1211)) = op(op(op(X1209,op(op(X1210,X1209),X1209)),X1211),op(op(X1210,op(X1213,X1212)),X1212)),X1211),op(X1214,op(op(X1215,X1211),X1214)))) [superposition 232,1919]
10362. op(X27,X29) = op(op(X27,X29),op(op(X27,op(op(X28,X27),X27)),op(X29,op(X27,X29)))) [backward demodulation 5844,10142]
10363. op(X29,op(X27,X29)) = op(op(X29,op(X27,X29)),op(X27,op(X27,X27))) [backward demodulation 2869,10142]
10366. op(X27,X29) = op(op(X27,X29),op(X27,op(X28,X27),X27)) [forward demodulation 10362,9]
10435. op(X18,op(op(op(X8,X9),X9),X18)) = op(op(X18,op(op(op(X8,X9),X9),X18)),op(op(op(X8,X9),X9),op(op(op(X8,X9),X9))))) [backward demodulation 1896,10363]
10440. op(X10,op(op(op(X8,X9),X9),X10)) = op(op(X10,op(op(op(X8,X9),X9),X10)),op(X9,op(op(op(X8,X9),X9))) [forward demodulation 10435,9]
10453. op(X3,op(X0,op(X1,op(op(X2,X0),X1)))) = op(op(X3,op(X0,op(X1,op(op(X2,X0),X1)))),op(X3,op(op(X2,X0),X1)),op(X0,op(X1,op(op(X2,X0),X1)))) [superposition 10440,7]
10514. op(X3,op(X0,X3)) = op(X3,op(X0,X3)),op(X3,op(op(X2,X0),X1),X0)) [forward demodulation 10453,7]
10871. op(X153,op(X151,X153)) = op(op(X153,op(X151,X153)),op(X151,op(X151,X153))) [superposition 53,10362]
10874. op(X160,op(op(X160,op(X160,X160)),op(X162,op(op(X163,X160),X162)),X160))) = X160 [superposition 741,10366]
11290. op(X160,op(X160,op(X160,X160))) = X160 [forward demodulation 10874,10514]
11496. op(X8,op(X7,X8)) = op(op(X8,op(X7,X8)),op(op(X9,op(op(X6,X7),X9)),op(op(X6,X7),op(op(X6,X7),op(X6,X7)))) [superposition 35,11290]
11498. op(X7,X8)) = op(op(X8,op(X7,X8)),op(X7,X8)),op(X9,op(op(X6,X7),X9))) [forward demodulation 11496,10871]
11689. op(op(X59,op(op(X68,X55),X59)),X55) = op(op(op(X59,op(op(X68,X55),X59)),X55),op(op(X56,op(op(X54,X55),X56)),op(X53,op(op(X54,X55),X53)))) [backward demodulation 6230,11690]
11716. op(op(X2,X8),X1) = op(op(op(X2,X8),X1),op(X3,op(X8,X1))) [backward demodulation 11,11690]
11722. op(X4,op(X2,X4)) = op(op(X4,op(X2,X4)),op(X3,op(X2,X3))) [backward demodulation 53,11690]
12250. op(X36,op(op(X37,op(op(X38,X31),X37)),X36)) = op(op(X36,op(op(X37,op(op(X38,X31),X37)),X36)),op(op(X32,op(op(X33,X31),X32)),X31),op(X34,op(op(X35,X31),X34)))) [backward demodulation 5621,11716]
12322. op(op(X59,op(op(X68,X55),X59)),X55) = op(op(op(X59,op(op(X68,X55),X59)),X55),op(X56,op(op(X54,X55),X56))) [backward demodulation 11689,11722]
12335. op(op(X3,X8),X4) = op(op(op(X3,X8),X4),op(op(X1,op(op(X2,X8),X1)),X8)) [backward demodulation 104,12322]
12421. op(op(X1209,op(op(X1210,X1209),X1209)),X1211) = op(op(X1209,op(op(X1210,X1209),X1209)),X1211),op(op(X1212,op(op(X1213,X1211),X1212)),X1211)) [backward demodulation 10345,12322]
12428. op(X36,op(op(X37,op(op(X38,X31),X37)),X36)) = op(op(X36,op(op(X37,op(op(X38,X31),X37)),X36)),op(op(X32,op(op(X33,X31),X32)),X31)) [backward demodulation 12250,12322]
12483. op(X68,op(op(op(X66,op(op(X67,X68),X66)),X68),op(op(X62,op(op(X63,X68),X62)),X61)),op(X64,op(op(X65,X68),X64)),X68))) = X68 [backward demodulation 2615,12335]
12572. op(X60,op(op(op(X66,op(op(X67,X68),X66)),X68),op(X61,op(op(X62,op(op(X63,X68),X62)),X61)))) = X68 [backward demodulation 12483,12428]
12659. op(X68,op(op(X66,op(op(X67,X68),X66)),X68)) = X68 [forward demodulation 12572,37]
12744. op(X985,op(op(op(X983,op(op(X984,X983),X983)),X985),op(X988,op(op(X989,X985),X988)),X985)) = X985 [backward demodulation 10389,12659]
12854. op(X985,op(op(X983,op(op(X984,X983),X983)),X985)) = X985 [forward demodulation 12744,12421]
12865. op(X48,op(X45,op(op(X46,X45),X45)),X48)) = op(X48,op(op(X45,op(op(X46,X45),X45),X48)),X44) [backward demodulation 1792,12854]
13881. sK0 != X48 [forward demodulation 12865,12854]
13945. sK0 != op(sK0,sK1) [backward demodulation 8,13881]
14946. $false [subsumption resolution 14845,13881]
```

# Tao's Equational Proof Challenge

Given that the Vampire proof is unintelligible, Terence Tao challenged the community to find "an alternate proof, by whatever means you wish—human, semi-automated, or automated."

# Tao's Equational Proof Challenge Accepted

Lydia Kondylidou[1] Jasmin Blanchette[1] Marijn J. H. Heule[2]

[1]Ludwig-Maximilians-Universität München
[2]Carnegie Mellon University

# Let’s Try the Twee Prover

# Let’s Try the Twee Prover (137 Steps)





# Introducing Krympa

Krympa is a proof minimization tool that

1. converts a given Vampire proof into a direct proof,
2. extracts candidate intermediate lemmas,
3. proves each lemma independently using Vampire and Twee,
4. combines the resulting proofs into a single, more concise proof using heuristics,
5. outputs a Lean proof.

Krympa is implemented in Rust, OCaml, and Python.

# Krympa's Proof for Tao's Challenge (20 Steps)

```
theorem Equation650_implies_Equation448 (G : Type _) [Magma G]
    (op_law : Equation650 G) : Equation448 G :=
  have lemma1 (x y z w : G) :
    x ◇ ((y ◇ z) ◇ x) = (x ◇ ((y ◇ z) ◇ x)) ◇ (w ◇ (z ◇ w)) := by
    nth_rw 3 [op_law z x y]
    exact op_law (x ◇ ((y ◇ z) ◇ x)) w z

  have lemma2 (x y z w v u : G) :
    x ◇ ((y ◇ ((z ◇ w) ◇ y)) ◇ x) =
    (x ◇ ((y ◇ ((z ◇ w) ◇ y)) ◇ x)) ◇ (v ◇ ((u ◇ (w ◇ u)) ◇ v)) := by
    nth_rw 1 2 [lemma1 y z w u]
    exact lemma1 x (y ◇ ((z ◇ w) ◇ y)) (u ◇ (w ◇ u)) v

  have lemma3 (x y z w v : G) :
    x ◇ (y ◇ x) =
    (x ◇ (y ◇ x)) ◇ (z ◇ ((w ◇ ((v ◇ y) ◇ w)) ◇ z)) := by
    nth_rw 1 [lemma1 w v y x]
    exact op_law (x ◇ (y ◇ x)) z (w ◇ ((v ◇ y) ◇ w))

  have lemma4 (x y z w v : G) :
    x ◇ (y ◇ x) =
    (x ◇ (y ◇ x)) ◇ ((z ◇ (y ◇ z)) ◇ (w ◇ ((v ◇ y) ◇ w))) := by
    nth_rw 1 [lemma1 w v y z]
    exact lemma3 x y (z ◇ (y ◇ z)) w v

  have lemma5 (x y z w : G) :
    x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x) =
    (x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) ◇ (w ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ w)) := by
    nth_rw 1 [lemma2 w y z y x ((z ◇ y) ◇ y)]
    exact lemma4 x (y ◇ ((z ◇ y) ◇ y)) w x ((z ◇ y) ◇ y)

  have lemma6 (x y z w : G) :
    (x ◇ ((y ◇ x) ◇ x)) ◇ z =
    ((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ ((w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) ◇
      (z ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ z))) := by
    nth_rw 1 [lemma5 z x y w]
    exact op_law ((x ◇ ((y ◇ x) ◇ x)) ◇ z) (w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) z

  have lemma7 (x y z w : G) :
    (x ◇ ((y ◇ x) ◇ x)) ◇ z =
    ((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ (w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) := by
    nth_rw 1 [lemma5 w x y z]
    exact lemma6 x y z w
```

```
have lemma8 (x y z w : G) :
  (((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) =
  ((x ◇ ((y ◇ x) ◇ x)) ◇ z) := by
  let T := x ◇ ((y ◇ x) ◇ x)
  calc
    (T ◇ z) ◇ (T ◇ w) =
  (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇ (T ◇ (T ◇ w))))) := by
      nth_rw 1 [←op_law]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇
        (T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w)))))))) := by
      nth_rw 1 [←lemma7]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇ ((T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w)))) ◇
        (((T ◇ w) ◇ (w ◇ (T ◇ w))) ◇ (T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w))))))))) := by
      nth_rw 2 [←lemma7]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ (w ◇ (T ◇ w)))) := by
      nth_rw 1 [←op_law]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ (T ◇ (T ◇ w))) := by
      nth_rw 1 [←lemma7]
  _ = ((x ◇ ((y ◇ x) ◇ x)) ◇ z) := by
      nth_rw 1 [←lemma7]

have lemma9 (x y z : G) :
  (x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) = x := by
  calc
    (x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) =
  x ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) := by
      nth_rw 1 [lemma8]
  _ = x ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x))) := by
      nth_rw 2 [lemma8]
  _ = x := by
      nth_rw 1 [←op_law]

show _ by
  intros x y z
  calc
    x = (x ◇ ((x ◇ ((x ◇ x) ◇ x)) ◇ x)) := by
      nth_rw 1 [lemma9]
  _ = ((x ◇ ((x ◇ ((x ◇ x) ◇ x)) ◇ x)) ◇ ((y ◇ (z ◇ (x ◇ z))) ◇ ((x ◇ ((x ◇ x) ◇ x)) ◇ (y ◇ (z ◇ (x ◇ z)))))) := by
      nth_rw 1 [←lemma5]
  _ = (x ◇ ((y ◇ (z ◇ (x ◇ z))) ◇ ((x ◇ ((x ◇ x) ◇ x)) ◇ (y ◇ (z ◇ (x ◇ z)))))) := by
      nth_rw 1 [lemma9]
  _ = (x ◇ (y ◇ (z ◇ (x ◇ z)))) := by
      nth_rw 1 [lemma9]
```

# Vampire

Vampire is a saturation-based theorem prover for first-order logic with equality and higher-order logic.

It is developed by Andrei Voronkov and colleagues.

It is based on the superposition calculus and implements highly optimized search strategies and data structures.

It finished first in the unit equality division of CASC 2025.

# Vampire Proofs

Vampire proofs are refutational.
They derive $\bot$ from the axioms and the negated conjecture.

# Vampire Proofs

Vampire proofs are refutational.
They derive $\bot$ from the axioms and the negated conjecture.

**Example:**

| | |
|---|---|
| 1. $a = b$ | axiom |
| 2. $f(x) = x$ | axiom |
| 3. $h(f(b)) \neq h(a)$ | negated conjecture |
| 4. $h(b) \neq h(a)$ | from 2 and 3 |
| 5. $h(a) \neq h(a)$ | from 1 and 4 |
| 6. $\bot$ | from 5 |

# Introducing Krympa

Krympa is a proof minimization tool that

1. converts a given Vampire proof into a direct proof,
2. extracts candidate intermediate lemmas,
3. proves each lemma independently using Vampire and Twee,
4. combines the resulting proofs into a single, more concise proof using heuristics,
5. outputs a Lean proof.

# Converting Vampire Proofs to Direct Proofs

We introduce quantifiers and apply the contrapositive to get positive equations.

# Converting Vampire Proofs to Direct Proofs

We introduce quantifiers and apply the contrapositive to get positive equations.

**Before:**

$$\frac{\mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \mathsf{h}(x, \mathsf{a}) \neq x}{\mathsf{b} \neq \mathsf{a}}$$

# Converting Vampire Proofs to Direct Proofs

We introduce quantifiers and apply the contrapositive to get positive equations.

**Before:**

$$\frac{\mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \mathsf{h}(x, \mathsf{a}) \neq x}{\mathsf{b} \neq \mathsf{a}}$$

**With quantifiers:**

$$\frac{\forall y.\ \mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \forall x.\ \mathsf{h}(x, \mathsf{a}) \neq x}{\mathsf{b} \neq \mathsf{a}}$$

# Converting Vampire Proofs to Direct Proofs

We introduce quantifiers and apply the contrapositive to get positive equations.

**Before:**

$$\frac{\mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \mathsf{h}(x, \mathsf{a}) \neq x}{\mathsf{b} \neq \mathsf{a}}$$

**With quantifiers:**

$$\frac{\forall y.\ \mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \forall x.\ \mathsf{h}(x, \mathsf{a}) \neq x}{\mathsf{b} \neq \mathsf{a}}$$

**After contrapositive:**

$$\frac{\forall y.\ \mathsf{h}(\mathsf{a}, y) = \mathsf{b} \qquad \mathsf{b} = \mathsf{a}}{\exists x.\ \mathsf{h}(x, \mathsf{a}) = x}$$

# Converting Vampire Proofs to Direct Proofs

**Before:**

| | |
|---|---|
| 1. $a = b$ | axiom |
| 2. $f(x) = x$ | axiom |
| 3. $h(f(b)) \neq h(a)$ | negated conjecture |
| 4. $h(b) \neq h(a)$ | from 2 and 3 |
| 5. $h(a) \neq h(a)$ | from 1 and 4 |
| 6. $\bot$ | from 5 |

# Converting Vampire Proofs to Direct Proofs

**Before:**

| | |
|---|---|
| 1. $a = b$ | axiom |
| 2. $f(x) = x$ | axiom |
| 3. $h(f(b)) \neq h(a)$ | negated conjecture |
| 4. $h(b) \neq h(a)$ | from 2 and 3 |
| 5. $h(a) \neq h(a)$ | from 1 and 4 |
| 6. $\bot$ | from 5 |

**After:**

| | |
|---|---|
| 1. $a = b$ | axiom |
| 2. $\forall x.\, f(x) = x$ | axiom |
| 3. $h(b) = h(a)$ | from 1 and $h(a) = h(a)$ |
| 4. $h(f(b)) = h(a)$ | from 2 and 3 |

# Twee

Twee is an automatic prover specialized for equational reasoning.

It is developed by Nicholas Smallbone.

It is based on unfailing completion, an extension of Knuth–Bendix completion.

It finished second in the unit equality division of CASC 2025.

## Twee Proofs

Twee proofs consist of a sequence of lemmas,
where the lemmas and the conjecture are proved by chains of equalities.

# Twee Proofs

Twee proofs consist of a sequence of lemmas,
where the lemmas and the conjecture are proved by chains of equalities.

**Example:**

Axiom 1: $a = b$

Axiom 2: $f(x) = x$

Lemma 3: $f(b) = a$
Proof:
$f(b)$
$= \{$ by axiom 1 right-to-left $\}$
$f(a)$
$= \{$ by axiom 2 $\}$
$a$

Goal 1: $h(f(b)) = h(a)$
Proof:
$h(f(b))$
$= \{$ by lemma 3 $\}$
$h(a)$

# Introducing Krympa

Krympa is a proof minimization tool that

1. converts a given Vampire proof into a direct proof,
2. extracts candidate intermediate lemmas,
3. proves each lemma independently using Vampire and Twee,
4. combines the resulting proofs into a single, more concise proof using heuristics,
5. outputs a Lean proof.

# Proof Generation for Intermediate Lemmas

**Theorem**

$A \Longrightarrow C$

**Vampire proof**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | from $A$ |
| $L_2$ | from $A$ and $L_1$ |
| $C$ | from $L_1$ and $L_2$ |

# Proof Generation for Intermediate Lemmas

| **Theorem** | **Vampire proof** | |
|---|---|---|
| $A \Longrightarrow C$ | $A$ | axiom |
| | $L_1$ | from $A$ |
| | $L_2$ | from $A$ and $L_1$ |
| | $C$ | from $L_1$ and $L_2$ |

For each intermediate lemma and $C$, we generate three problem variants:

1. big-step problem,
2. small-step problem,
3. abstracted problem.

Each problem is given to Vampire and Twee.

# Big-Step Problems

Big-step problems contain the axioms as well as the lemma as the conjecture.

# Big-Step Problems

Big-step problems contain the axioms as well as the lemma as the conjecture.

**Vampire proof**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | from $A$ |
| $L_2$ | from $A$ and $L_1$ |
| $C$ | from $L_1$ and $L_2$ |

**Problem for lemma $L_1$**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | conjecture |

**Problem for lemma $L_2$**

| | |
|---|---|
| $A$ | axiom |
| $L_2$ | conjecture |

## Small-Step Problems

Small-step problems contain the axioms, the lemma as the conjecture, and all previously derived lemmas as additional axioms.

# Small-Step Problems

Small-step problems contain the axioms, the lemma as the conjecture, and all previously derived lemmas as additional axioms.

**Vampire proof**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | from $A$ |
| $L_2$ | from $A$ and $L_1$ |
| $C$ | from $L_1$ and $L_2$ |

**Problem for lemma $L_1$**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | conjecture |

**Problem for lemma $L_2$**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | axiom |
| $L_2$ | conjecture |

# Abstracted Problems

Abstracted problems contain the axioms and an abstracted version of the lemma where some subterms are replaced with fresh variables.

# Abstracted Problems

Abstracted problems contain the axioms and an abstracted version of the lemma where some subterms are replaced with fresh variables.

**Vampire proof**

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | from $A$ |
| $L_2$ | from $A$ and $L_1$ |
| $C$ | from $L_1$ and $L_2$ |

**Problem for lemma $L_1$**

| | |
|---|---|
| $A$ | axiom |
| $L_1'$ | conjecture |

**Problem for lemma $L_2$**

| | |
|---|---|
| $A$ | axiom |
| $L_2'$ | conjecture |

# Introducing Krympa

Krympa is a proof minimization tool that

1. converts a given Vampire proof into a direct proof,
2. extracts candidate intermediate lemmas,
3. proves each lemma independently using Vampire and Twee,
4. combines the resulting proofs into a single, more concise proof using heuristics,
5. outputs a Lean proof.

# Proof Construction for the Main Theorem

We construct the minimized proof by concatenating three segments:

1. derivation of the departure lemma,
2. derivation of the arrival lemma,
3. derivation of the conjecture.

# Proof Construction for the Main Theorem

We construct the minimized proof by concatenating three segments:

1. derivation of the departure lemma,
2. derivation of the arrival lemma,
3. derivation of the conjecture.

We identify up to six intermediate steps near the end of the Vampire proof as candidate arrival lemmas.

For each candidate arrival lemma, its transitive dependencies are considered as candidate departure lemmas.

# Proof Construction for the Main Theorem

We construct the minimized proof by concatenating three segments:

1. derivation of the departure lemma,
2. derivation of the arrival lemma,
3. derivation of the conjecture.

We identify up to six intermediate steps near the end of the Vampire proof as candidate arrival lemmas.

For each candidate arrival lemma, its transitive dependencies are considered as candidate departure lemmas.

Why three and six? It is a trade-off between performance and flexibility.

# Proof Construction for the Main Theorem

For each segment:

1. We create a problem.
2. We run Vampire and Twee.
3. We keep the shorter proof.

To obtain the final minimized proof, we concatenate the three segments, pruning unreferenced lemmas.

# Example of Our Approach

Consider this 7-step Vampire proof of $A \Longrightarrow C$, which we want to minimize:

$A$ axiom
$L_1$ from $A$
$L_2$ from $A$ and $L_1$
$L_3$ from $L_1$ and $L_2$
$L_4$ from $L_2$ and $L_3$
$L_5$ from $L_3$ and $L_4$
$L_6$ from $A$ and $L_5$
$C$ from $L_5$ and $L_6$

# Example of Our Approach

Consider this 7-step Vampire proof of $A \Longrightarrow C$, which we want to minimize:

$A$ axiom
$L_1$ from $A$
$L_2$ from $A$ and $L_1$
$L_3$ from $L_1$ and $L_2$
$L_4$ from $L_2$ and $L_3$
$L_5$ from $L_3$ and $L_4$
$L_6$ from $A$ and $L_5$
$C$ from $L_5$ and $L_6$

For each intermediate lemma and $C$, we create the problem variants and try to prove them, keeping the shortest proof.

## Example of Our Approach

Steps $L_2, \ldots, L_6$, and $C$ are considered as candidate arrival lemmas.
Let's focus on $L_6$.

# Example of Our Approach

Steps $L_2, \ldots, L_6$, and $C$ are considered as candidate arrival lemmas.
Let's focus on $L_6$.

Here is the shortest proof of $L_6$:

| | |
|---|---|
| $A$ | axiom |
| $L_1$ | from $A$ |
| $L_2$ | from $A$ and $L_1$ |
| $L_3$ | from $L_1$ and $L_2$ |
| $L_4$ | from $L_2$ and $L_3$ |
| $L_5$ | from $L_3$ and $L_4$ |
| $L_6$ | from $A$ and $L_5$ |

# Example of Our Approach

Steps $L_2, \ldots, L_6$, and $C$ are considered as candidate arrival lemmas.
Let's focus on $L_6$.

Here is the shortest proof of $L_6$:

$A$ axiom
$L_1$ from $A$
$L_2$ from $A$ and $L_1$
$L_3$ from $L_1$ and $L_2$
$L_4$ from $L_2$ and $L_3$
$L_5$ from $L_3$ and $L_4$
$L_6$ from $A$ and $L_5$

Steps $L_1, \ldots, L_5$ are considered as candidate departure lemmas.
Let's focus on $L_3$.

# Example of Our Approach

**Segment 1:**

We create a problem with $A$, $L_1$, and $L_2$ as axioms and $L_3$ as the conjecture.
The shortest proof is found by Vampire:

$A$ axiom
$L_1$ from $A$
$L_2'$ from $A$ and $L_1$
$L_3$ from $L_1$ and $L_2'$

# Example of Our Approach

**Segment 2:**

We create a problem with $A$, $L_1$, $L_2'$, and $L_3$ as axioms and $L_6$ as the conjecture.
The shortest proof is found by Twee:

$L_6$ by a 2-step equality chain using $L_1$ and $L_3$

# Example of Our Approach

**Segment 3:**

We create a problem with $A$, $L_1$, $L_2'$, $L_3$, and $L_6$ as axioms and $C$ as the conjecture.
The shortest proof is found by Twee:

$C$ by a 2-step equality chain using $L_2'$ and $L_3$

Since this proof does not use the arrival lemma $L_6$, we skip segment 2.

# Example of Our Approach

**Final proof:**

| | | |
|---|---|---|
| segment 1 | $A$ | axiom |
| | $L_1$ | from $A$ |
| | $L_2'$ | from $A$ and $L_1$ |
| | $L_3$ | from $L_1$ and $L_2'$ |
| segment 3 | $C$ | by a 2-step equality chain using $L_2'$ and $L_3$ |

# Extensions: The term-size option

Given two proofs *A* and *B*:

**Default** if *A* has **fewer steps** than *B*, prefer *A* (and vice versa).

**Override** if *A* has a **smaller average term size** than *B* and is at most 1.5× as long, prefer *A* (and vice versa).

**Tie** if *A* and *B* have the same number of steps, prefer the one with the **smaller average term size**.

# A Simpler Proof for Tao's Challenge (19 Steps)

```
theorem Equation650_implies_Equation448 (G : Type _) [Magma G]
  (op_law : ∀ x y z : G, x = x ◇ (y ◇ ((z ◇ x) ◇ y))) :=
  have lemma1 (x y z w : G) :
    x ◇ ((y ◇ z) ◇ x) = (x ◇ ((y ◇ z) ◇ x)) ◇ (w ◇ (z ◇ w)) := by
    nth_rw 3 [op_law z x y]
    exact op_law (x ◇ ((y ◇ z) ◇ x)) w z

  have lemma2 (x y z w v u : G) :
    x ◇ ((y ◇ ((z ◇ w) ◇ y)) ◇ x) =
    (x ◇ ((y ◇ ((z ◇ w) ◇ y)) ◇ x)) ◇ (v ◇ ((u ◇ (w ◇ u)) ◇ v)) := by
    nth_rw 1 2 [lemma1 y z w u]
    exact lemma1 x (y ◇ ((z ◇ w) ◇ y)) (u ◇ (w ◇ u)) v

  have lemma3 (x y z w v : G) :
    x ◇ (y ◇ x) = (x ◇ (y ◇ x)) ◇ (z ◇ ((w ◇ ((v ◇ y) ◇ w)) ◇ z)) := by
    nth_rw 1 [lemma1 w v y x]
    exact op_law (x ◇ (y ◇ x)) z (w ◇ ((v ◇ y) ◇ w))

  have lemma4 (x y z w v : G) :
    x ◇ (y ◇ x) = (x ◇ (y ◇ x)) ◇ ((z ◇ (y ◇ z)) ◇ (w ◇ ((v ◇ y) ◇ w))) := by
    nth_rw 1 [lemma1 w v y z]
    exact lemma3 x y (z ◇ (y ◇ z)) w v

  have lemma5 (x y z w : G) :
    x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x) =
    (x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) ◇ (w ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ w)) := by
    nth_rw 1 [lemma2 w y z y x ((z ◇ y) ◇ y)]
    exact lemma4 x (y ◇ ((z ◇ y) ◇ y)) w x ((z ◇ y) ◇ y)

  have lemma6 (x y z w : G) :
    (x ◇ ((y ◇ x) ◇ x)) ◇ z =
    ((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ ((w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) ◇ (z ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ z))) := by
    nth_rw 1 [lemma5 z x y w]
    exact op_law ((x ◇ ((y ◇ x) ◇ x)) ◇ z) (w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) z

  have lemma7 (x y z w : G) :
    (x ◇ ((y ◇ x) ◇ x)) ◇ z =
    ((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ (w ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w)) := by
    nth_rw 1 [lemma5 w x y z]
    exact lemma6 x y z w
```

```
have lemma8 (x y z w : G) :
  ((x ◇ ((y ◇ x) ◇ x)) ◇ z) ◇ ((x ◇ ((y ◇ x) ◇ x)) ◇ w) =
  (x ◇ ((y ◇ x) ◇ x)) ◇ z := by
  let T := x ◇ ((y ◇ x) ◇ x)
  calc
    (T ◇ z) ◇ (T ◇ w) =
    (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇ (T ◇ (T ◇ w))))) := by
      nth_rw 1 [←op_law]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇
        (T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w)))))) := by
      nth_rw 1 [←lemma7]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ ((w ◇ (T ◇ w)) ◇ ((T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w)))) ◇
        (((T ◇ w) ◇ (w ◇ (T ◇ w))) ◇ (T ◇ ((T ◇ w) ◇ (w ◇ (T ◇ w))))))))) := by
      nth_rw 2 [←lemma7]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ ((T ◇ (T ◇ w)) ◇ (w ◇ (T ◇ w)))) := by
      nth_rw 1 [←op_law]
  _ = (T ◇ z) ◇ ((T ◇ w) ◇ (T ◇ (T ◇ w))) := by
      nth_rw 1 [←lemma7]
  _ = ((x ◇ ((y ◇ x) ◇ x)) ◇ z) := by
      nth_rw 1 [←lemma7]

have lemma9 (x y z : G) :
  x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x) = x := by
  calc
    x ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x) =
    x ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x)) := by
      nth_rw 1 [lemma8]
  _ = x ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ (((y ◇ ((z ◇ y) ◇ y)) ◇ x) ◇ ((y ◇ ((z ◇ y) ◇ y)) ◇ x))) := by
      nth_rw 2 [←lemma8]
  _ = x := by
      nth_rw 1 [←op_law]

have lemma10 (x y z w : G) :
  x ◇ (y ◇ ((z ◇ ((w ◇ z) ◇ z)) ◇ y)) = x := by
  symm
  nth_rw 1 2 [←lemma9 x z w]
  exact lemma5 x z w y

have lemma11 (x y : G) : x ◇ y = x := by
  nth_rw 1 [←lemma9 y x x]
  exact lemma10 x y x x

show _ by
  intros x y z
  nth_rw 1 [lemma11 x (y ◇ (z ◇ (x ◇ z)))]
```

# Another Challenge

$$\begin{aligned} &\left(\forall x, y, z.\ x = y \diamond ((z \diamond x) \diamond (y \diamond x))\right) \\ \Longrightarrow &\left(\forall x, y, z.\ x \diamond x = (y \diamond (z \diamond x)) \diamond x\right) \end{aligned}$$

# The Vampire Proof for the New Challenge (151 Steps)



# Krympa's Proof for the New Challenge (10 Steps)

```
theorem Equation947_implies_Equation3897 (G : Type _) [Magma G]
    (op_law : ∀ x y z : G, x = y ◇ ((z ◇ x) ◇ (y ◇ x))) :
    ∀ x y z : G, x ◇ x = (y ◇ (z ◇ x)) ◇ x :=
  have lemma_1 (x y z w : G) :
    ((x ◇ y) ◇ (z ◇ y)) =
    (w ◇ (y ◇ (w ◇ ((x ◇ y) ◇ (z ◇ y))))) := by
    nth_rw 3 [op_law y z x]
    exact op_law ((x ◇ y) ◇ (z ◇ y)) w z

  have lemma_2 (x y z : G) :
    ((x ◇ y) ◇ (z ◇ y)) = (z ◇ (y ◇ y)) := by
    nth_rw 4 [op_law y z x]
    exact lemma_1 x y z z

  have lemma_3 (x y : G) :
    y = x ◇ (x ◇ (y ◇ y)) := by
    nth_rw 1 [←lemma_2 x y x]
    exact op_law y x x
```

```
have lemma_4 (x y z : G) :
  (x ◇ x) = ((y ◇ (z ◇ x)) ◇ x) := by
  calc
    (x ◇ x) = ((y ◇ (z ◇ x)) ◇ ((y ◇ (z ◇ x)) ◇ ((x ◇ x) ◇ (x ◇ x)))) := by
      nth_rw 1 [←lemma_3]
    _ = ((y ◇ (z ◇ x)) ◇ ((y ◇ (z ◇ x)) ◇
          (z ◇ (z ◇ ((x ◇ x) ◇ (x ◇ x)) ◇ ((x ◇ x) ◇ (x ◇ x)))))) := by
      nth_rw 2 [←lemma_3]
    _ = ((y ◇ (z ◇ x)) ◇ ((y ◇ (z ◇ x)) ◇ (z ◇ (z ◇ ((x ◇ x) ◇ ((x ◇ x) ◇ (x ◇ x)))))) := by
      nth_rw 1 [lemma_2]
    _ = ((y ◇ (z ◇ x)) ◇ ((y ◇ (z ◇ x)) ◇ (z ◇ (z ◇ x)))) := by
      nth_rw 1 [←lemma_3]
    _ = ((y ◇ (z ◇ x)) ◇ (z ◇ ((z ◇ x) ◇ (z ◇ x)))) := by
      nth_rw 1 [lemma_2]
    _ = ((y ◇ (z ◇ x)) ◇ (z ◇ (z ◇ (x ◇ x)))) := by
      nth_rw 1 [lemma_2]
    _ = ((y ◇ (z ◇ x)) ◇ x) := by
      nth_rw 1 [←lemma_3]

show _ by
  exact lemma_4
```

# Evaluation on the Equational Theories Project

## (Vampire Proofs of at Least 15 Steps)

| File | Num. problems | Avg. steps before | Avg. steps after | | | |
|---|---|---|---|---|---|---|
| | | | BA | SA | BS | BSA |
| `Proofs1.lean` | 9 | 17.3 | | | | |
| `Proofs2.lean` | 8 | 16.9 | | | | |
| `Proofs3.lean` | 7 | 19.3 | | | | |
| `Proofs4.lean` | 11 | 19.1 | | | | |
| `Proofs5.lean` | 9 | 20.1 | | | | |
| `Proofs6.lean` | 11 | 25.6 | | | | |
| `Proofs7.lean` | 11 | 37.2 | | | | |
| `Proofs8.lean` | 7 | 24.4 | | | | |
| `Proofs9.lean` | 14 | 39.8 | | | | |
| `Proofs10.lean` | 2 | 21.5 | | | | |
| `Proofs11.lean` | 5 | 25.4 | | | | |
| `Proofs12.lean` | 13 | 24.6 | | | | |
| `Proofs13.lean` | 10 | 35.3 | | | | |
| Total | 117 | 26.3 | | | | |

# Evaluation on the Equational Theories Project

## (Vampire Proofs of at Least 15 Steps)

| File | Num. problems | Avg. steps before | Avg. steps after | | | |
|---|---|---|---|---|---|---|
| | | | BA | SA | BS | BSA |
| `Proofs1.lean` | 9 | 17.3 | 16.0 | **13.1** | 13.3 | 13.3 |
| `Proofs2.lean` | 8 | 16.9 | 14.4 | **11.5** | **11.5** | **11.5** |
| `Proofs3.lean` | 7 | 19.3 | 15.6 | 10.9 | **10.7** | 10.9 |
| `Proofs4.lean` | 11 | 19.1 | 14.3 | **10.9** | 11.4 | 11.4 |
| `Proofs5.lean` | 9 | 20.1 | 17.6 | 12.8 | **11.9** | **11.9** |
| `Proofs6.lean` | 11 | 25.6 | 18.9 | **12.5** | 12.6 | 12.6 |
| `Proofs7.lean` | 11 | 37.2 | 19.7 | **11.8** | 11.9 | 11.9 |
| `Proofs8.lean` | 7 | 24.4 | 15.6 | **12.3** | 13.1 | 13.1 |
| `Proofs9.lean` | 14 | 39.8 | 29.0 | **13.1** | 14.1 | 14.1 |
| `Proofs10.lean` | 2 | 21.5 | 16.0 | **8.0** | 11.0 | 11.0 |
| `Proofs11.lean` | 5 | 25.4 | 22.4 | **13.0** | 14.0 | 14.0 |
| `Proofs12.lean` | 13 | 24.6 | 16.5 | **8.0** | 8.5 | 8.5 |
| `Proofs13.lean` | 10 | 35.3 | 27.7 | **9.1** | 10.1 | 10.1 |
| Total | 117 | 26.3 | 19.2 | **11.4** | 11.8 | 11.8 |

# Conclusion

We showed how to minimize equational proofs to increase their readability
by mixing and matching the output of separate runs of Vampire and Twee.

We implemented the approach in a new tool, Krympa.
We used Krympa to reduce the proof of Terence Tao's problem from 62 to 19 steps.

# Conclusion

We showed how to minimize equational proofs to increase their readability
by mixing and matching the output of separate runs of Vampire and Twee.

We implemented the approach in a new tool, Krympa.
We used Krympa to reduce the proof of Terence Tao's problem from 62 to 19 steps.

Proof automation and readability can go hand in hand.

# Tao's Equational Proof Challenge Accepted

Lydia Kondylidou[1] Jasmin Blanchette[1] Marijn J. H. Heule[2]

[1]Ludwig-Maximilians-Universität München
[2]Carnegie Mellon University