Automatically Generalizing Proofs and Statements

Anshula Gandhi
University of Cambridge, UK

Anand Rao Tadipatri
University of Cambridge, UK

Timothy Gowers
Collège de France, Paris, France
University of Cambridge, UK

An Algorithm to Generalize Proofs

We've implemented a **proof generalization algorithm** in Lean. That is, we've developed an algorithm that can take in a mathematical proof, and outputs a more general statement that the "same" proof works for.

This algorithm builds on the work of Olivier Pons ("Generalization in type theory based proof assistants"), who implemented a precursor to this algorithm in Rocq.

An Algorithm to Generalize Proofs

Suppose we prove:

$$\sqrt{17} \text{ is irrational.}$$

```
example := by
  let irrat_sqrt : Irrational (√17) := by {apply irrat_de
```

▼Tactic state

1 goal

`irrat_sqrt` : Irrational √17

An Algorithm to Generalize Proofs

Suppose we prove:

$$\sqrt{17} \text{ is irrational.}$$

```
example := by
  let irrat_sqrt : Irrational (√17) := by {apply irrat_de
```

▼Tactic state

1 goal

`irrat_sqrt` : Irrational √17

The only fact that we really use about 17 in proving this is that it is prime.

An Algorithm to Generalize Proofs

Suppose we prove:

$$\sqrt{17} \text{ is irrational.}$$

```
example := by
  let irrat_sqrt : Irrational (√17) := by {apply irrat_de
  autogeneralize (17:ℕ) in irrat_sqrt
```

▼ Tactic state

1 goal

```
irrat_sqrt : Irrational √17
irrat_sqrt.Gen : ∀ (n : ℕ),
Nat.Prime n → Irrational √↑n
```

The only fact that we really use about 17 in proving this is that it is prime.

So, this algorithm examines the statement and its proof, and by checking which lemmas in the proof are used, **generalizes** to the theorem:

$$\forall \text{ primes } p, \sqrt{p} \text{ is irrational.}$$

(Live Demo)

An Algorithm to Generalize Proofs

The algorithm does not derive the more general theorem — that the square root of a non-perfect square is irrational — when that reasoning step is not evident in the proof term.

That is, the algorithm generalizes the theorem *only as far as the proof allows.*

Other Proof Generalizers

A non-comprehensive list of other software to generalize proofs in dependent type theory:

- (Generalizing proofs in Rocq) *Generalization in type theory based proof assistants* by Olivier Pons.
- (Generalizing inductive types in Rocq) *Ornaments for Proof Reuse in Coq* by Talia Ringer, Nathaniel Yazdani, John Leo, and Dan Grossman.
- (Generalizing typeclasses in Lean 3) *Automatically Generalizing Theorems Using Typeclasses* by Alex J. Best.

Our work aims to advance the capabilities of proof generalizers.

An Algorithm to Generalize Proofs:

What's New?

In particular, we've implemented two improvements to state-of-the-art proof generalization in dependent type theory:

- Generalization of *repeated* constants
- Generalization of *dependent* constants

Here are some examples...

Generalizing Repeated Constants

Suppose we prove:

$$\sqrt{17} + 17 \text{ is irrational.}$$

```
example:= by
  let irrat_sum_sqrt : Irrational (sqrt (17:ℕ)+17) := by
```

▼Tactic state

1 goal

```
irrat_sum_sqrt : Irrational
(√↑17 + 17)
```

Generalizing Repeated Constants

Suppose we prove:

$$\sqrt{17} + 17 \text{ is irrational.}$$

```
example:= by
  let irrat_sum_sqrt : Irrational (sqrt (17:ℕ)+17) := by
```

▼Tactic state

1 goal

`irrat_sum_sqrt` : Irrational (√↑17 + 17)

It would be suboptimal to generalize all the 17s in the same way, and yield the overspecialized proof:

$$\forall \text{ primes } p, \sqrt{p} + p \text{ is irrational.}$$

Generalizing Repeated Constants

Suppose we prove:

$$\sqrt{17} + 17 \text{ is irrational.}$$

```
example:= by
  let irrat_sum_sqrt : Irrational (sqrt (17:ℕ)+17) := by
  autogeneralize (17:ℕ) in irrat_sum_sqrt
```

▼Tactic state

1 goal

```
irrat_sum_sqrt : Irrational
(√↑17 + 17)
irrat_sum_sqrt.Gen : ∀ (n : ℕ),
Nat.Prime n → ∀ (m : ℕ),
Irrational (√↑n + ↑m)
```

The algorithm should recognize that **the 17s play different roles in the proof**, and should be generalized separately. Indeed, it **generalizes** to the theorem:

$$\forall \text{ primes } p \text{ and natural numbers } n, \sqrt{p} + n \text{ is irrational.}$$

(Live Demo)

Generalizing Dependent Constants

Suppose we prove:

The union of two sets of size 2 *has size at most* 4.

```
example := by
  let union_of_finsets {α : Type} [Fintype α] [DecidableEq α] (A B
    (hA : A.card = 2) (hB : B.card = 2) : (A ∪ B).card ≤ 4 := by a
```

▼Tactic state

1 goal

```
union_of_finsets : ∀ {α : Type} [inst : Fintype α] [inst
: DecidableEq α] (A B : Finset α), A.card = 2 → B.card =
2 → (A ∪ B).card ≤ 4
```

Generalizing Dependent Constants

Suppose we prove:

The union of two sets of size 2 *has size at most* 4*.*

```
example := by
  let union_of_finsets {α : Type} [Fintype α] [DecidableEq α] (A B
    (hA : A.card = 2) (hB : B.card = 2) : (A ∪ B).card ≤ 4 := by a

  autogeneralize (2:ℕ) in union_of_finsets
```

▼Tactic state

1 goal

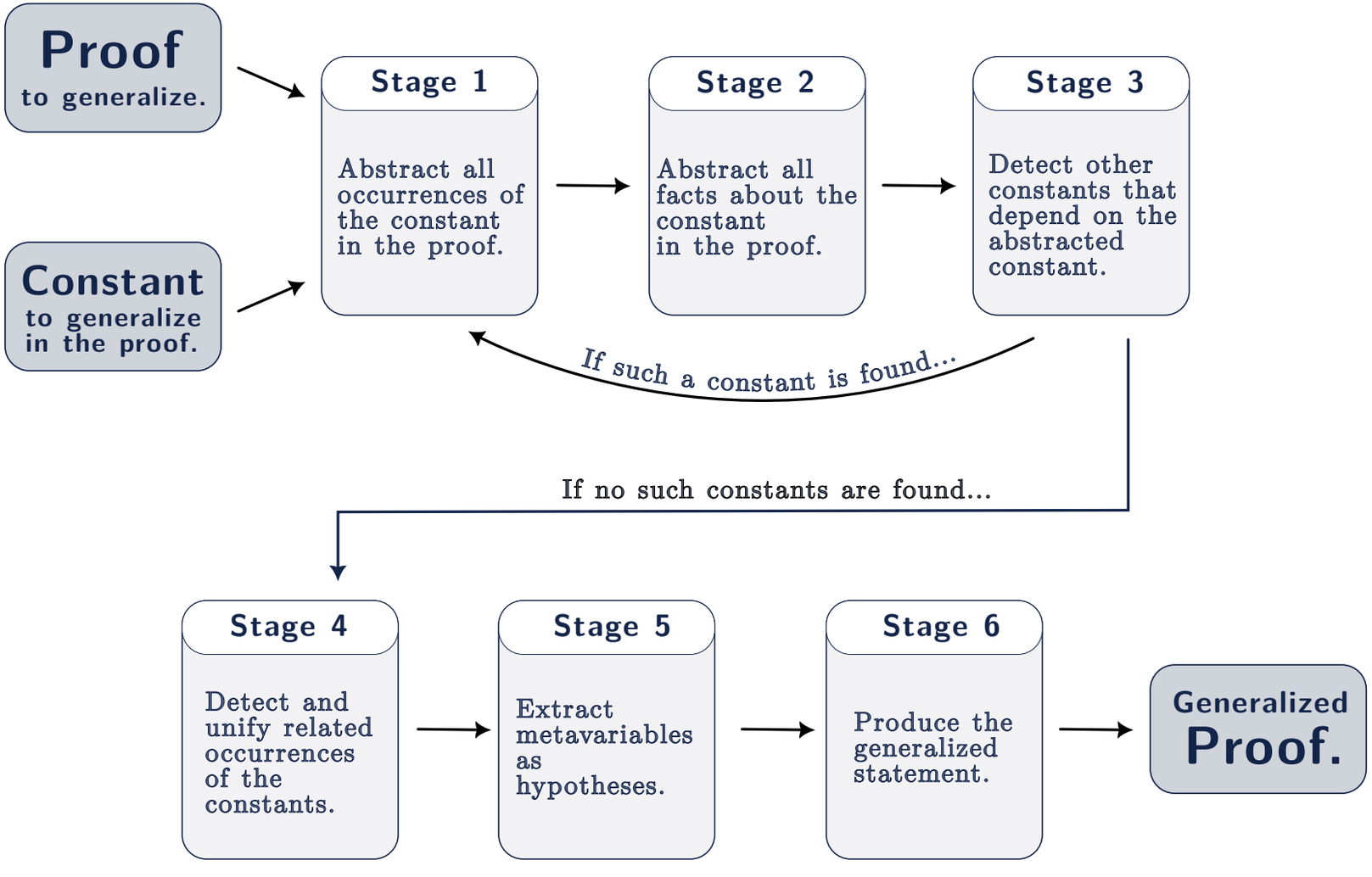
```
union_of_finsets : ∀ {α : Type} [inst : Fintype α] [inst
: DecidableEq α] (A B : Finset α), A.card = 2 → B.card =
2 → (A ∪ B).card ≤ 4
union_of_finsets.Gen : ∀ (n m : ℕ) {α : Type} [inst :
Fintype α] [inst : DecidableEq α] (A B : Finset α),
  A.card = n → B.card = m → (A ∪ B).card ≤ n + m
```

The algorithm recognizes that **the 4 is actually a $2+2$**, and that the 2s need not be generalized to the same variable (abilities we've added to the algorithm which weren't present in the precursor). So it **generalizes** to the theorem:

The union of sets of size n *and* m *has size at most* $n+m$*.*

(Live Demo)

Algorithm Description



Generalizing Non-Numerical Constants

This tactic also works on longer, non-trivial proofs....

Suppose we prove Bézout's identity in the integers.

> *For any two integers x and $y \neq 0$, there exist integers h and k such that their greatest common divisor g can be expressed as a linear combination $hx + ky = g$.*

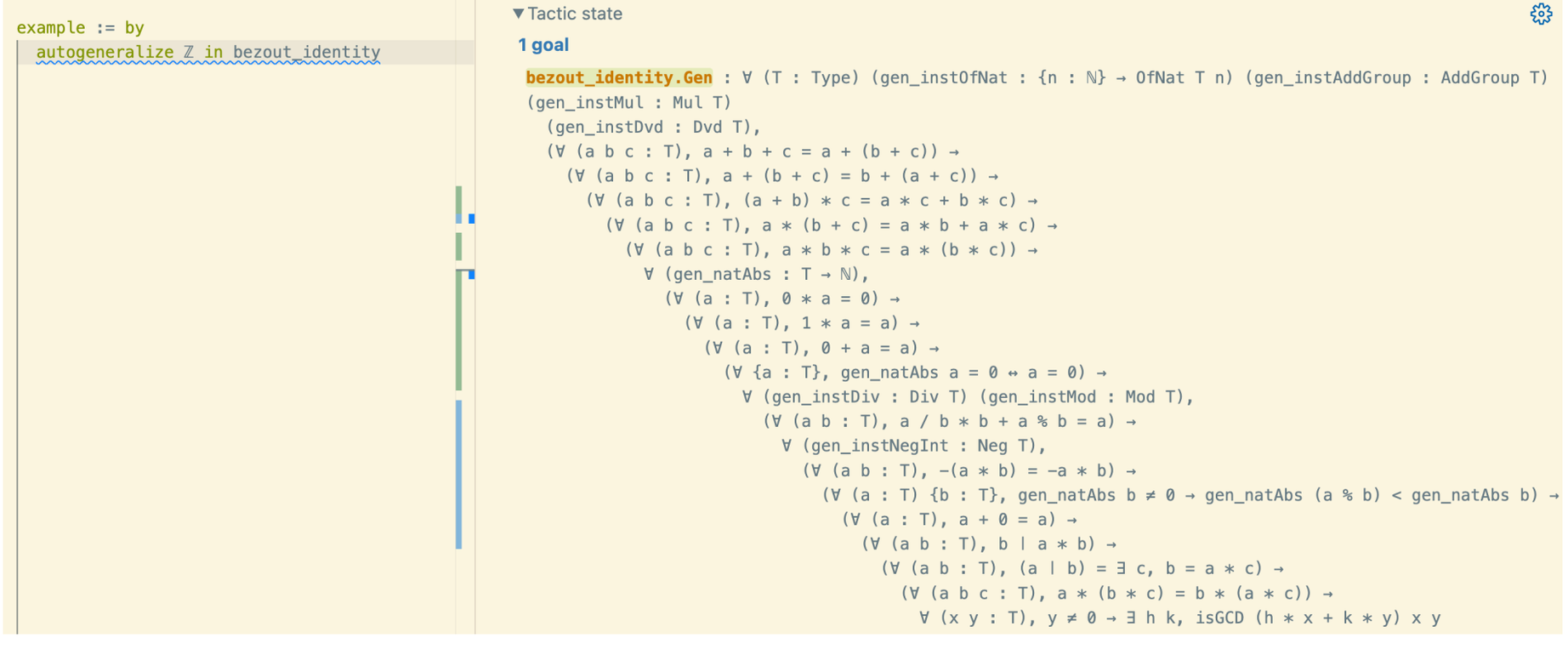
```
#check bezout_identity
```

▼Expected type

```
⊢ ∀ (x y : ℤ), y ≠ 0 → ∃ h k,
isGCD (h * x + k * y) x y
```

Generalizing Non-Numerical Constants

When we want to generalize the integers $\mathbb{Z}$, the algorithm recognizes that the same argument works in (something close to) an arbitrary Euclidean domain.



(Live Demo)

Limitations of Proof Generalization

Proofs with **black boxes** cannot be directly generalized.

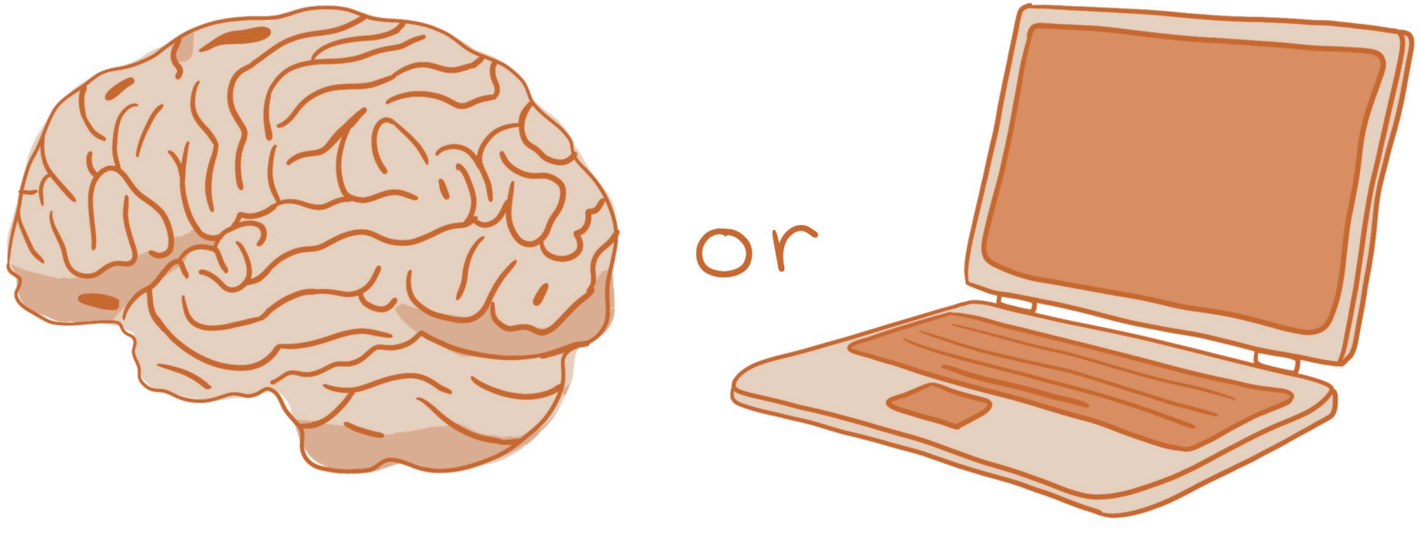
For example...

- Implicit uses of definitional equality, or
- Reliance on computation rules (e.g. `reduceMul`)

...all create "black boxes" in the proof term, that then make it difficult to generalize the proof.

An Algorithm to Generalize Proofs

Proof generalization is (of course) done not only by computers. Human mathematicians generalize proofs so frequently (e.g. when generalizing the proof of a counterexample) that we rarely think about it explicitly.



Any system that employs **human-oriented theorem proving** will be a system that employs **proof generalization**.

Questions?