Canonical

Simplify, Monomorphize, Destruct

```lean
theorem add_assoc (a b c : Nat) : (a + b) + c = a + (b + c) :=
  by canonical
```

```lean
theorem add_comm (a b : Nat) : a + b = b + a :=
  by canonical 60
```

```lean
theorem pow_add (a m n : Nat) : a^(m + n) = a^m * a^n
  by canonical [mul_one, mul_assoc]
```

“Why can’t Canonical output tactics?”

Tactics do not commute with metavariable assignments.

```
?n : Nat          ?n + x = x

   0                 simp
```

Tactics do not produce constraints on metavariables.

Tactics require interaction with the Lean server.

Tactics do not have soundness and completeness guarantees.

`simp`

Essential for reasoning about equality and bi-implication.

> *"all it does is repeatedly replace (or rewrite) subterms of the form A by B, for all applicable facts of the form A = B or A ↔ B"*

Type theorists have a tool for this too: **definitional reduction rules.**

```
Nat.rec m a b zero ↦ a
Nat.rec m a b n.succ ↦ b n (Nat.rec m a b n)

Prod.fst ⟨a, b⟩ ↦ a
Prod.snd ⟨a, b⟩ ↦ b

(let t := M; t) ↦ M
```

General Reduction Rule System

Canonical now accepts custom definitional reduction rules: `0 + n ↦ n`

Reduction rules are applied **pervasively**, as though `simp` were always applied.

Instead of having Canonical generate `simp` invocations (impossibly difficult) we now only need to reason about terms in the quotient of `simp` lemmas!

We forbid Canonical from writing reducible expressions, like `0 + n`

canonical

We register Lean's definitional equalities as reduction rules.

We register equation compiler lemmas as reduction rules.

We register relevant simp lemmas as reduction rules.

We proactively identify premises that can be registered as reduction rules.

Canonical carefully tracks the usage of these reduction rules.

Non-definitional reduction rules are reconstructed as `simp` invocations.

```
Nat.add : (a.277 : Nat) → (a.278 : Nat) → Nat := Canonical.Pi.f Nat
  (λ x.350 ↦ Nat) (PProd.fst (Canonical.Pi Nat (λ a.293 ↦ Nat))
  (Nat.rec (λ t.294 ↦ Sort) PUnit (λ n.295 n_ih.296 ↦ PProd
  (Canonical.Pi Nat (λ a.300 ↦ Nat)) n_ih.296) a.278) (Nat.rec (λ
  t.301 ↦ PProd (Canonical.Pi Nat (λ a.303 ↦ Nat)) (Nat.rec (λ
  t.304 ↦ Sort) PUnit (λ n.305 n_ih.306 ↦ PProd (Canonical.Pi Nat
  (λ a.308 ↦ Nat)) n_ih.306) t.301)) (PProd.mk (Canonical.Pi Nat (λ
  a.310 ↦ Nat)) PUnit (Canonical.Pi.mk Nat (λ a.317 ↦ Nat) (λ a.318
  ↦ a.318)) PUnit.unit) (λ n.326 n_ih.327 ↦ PProd.mk (Canonical.Pi
  Nat (λ a.329 ↦ Nat)) (PProd (Canonical.Pi Nat (λ a.331 ↦ Nat))
  (Nat.rec (λ t.332 ↦ Sort) PUnit (λ n.333 n_ih.334 ↦ PProd
  (Canonical.Pi Nat (λ a.336 ↦ Nat)) n_ih.334) n.326))
  (Canonical.Pi.mk Nat (λ a.338 ↦ Nat) (λ a.339 ↦ Nat.succ
  (Canonical.Pi.f Nat (λ x.349 ↦ Nat) (PProd.fst (Canonical.Pi Nat
  (λ a.343 ↦ Nat)) (Nat.rec (λ t.344 ↦ Sort) PUnit (λ n.345
  n_ih.346 ↦ PProd (Canonical.Pi Nat (λ a.348 ↦ Nat)) n_ih.346)
  n.326) n_ih.327) a.339))) n_ih.327) a.278)) a.277
```

→

```
Nat.add : (a.412 : Nat) → (a.413 : Nat) → Nat := [
  Nat.add x.409 Nat.zero ↦ x.409,
  Nat.add x.414 (Nat.succ b.415) ↦ Nat.succ (Nat.add x.414 b.415)
]
```

Unsoundness

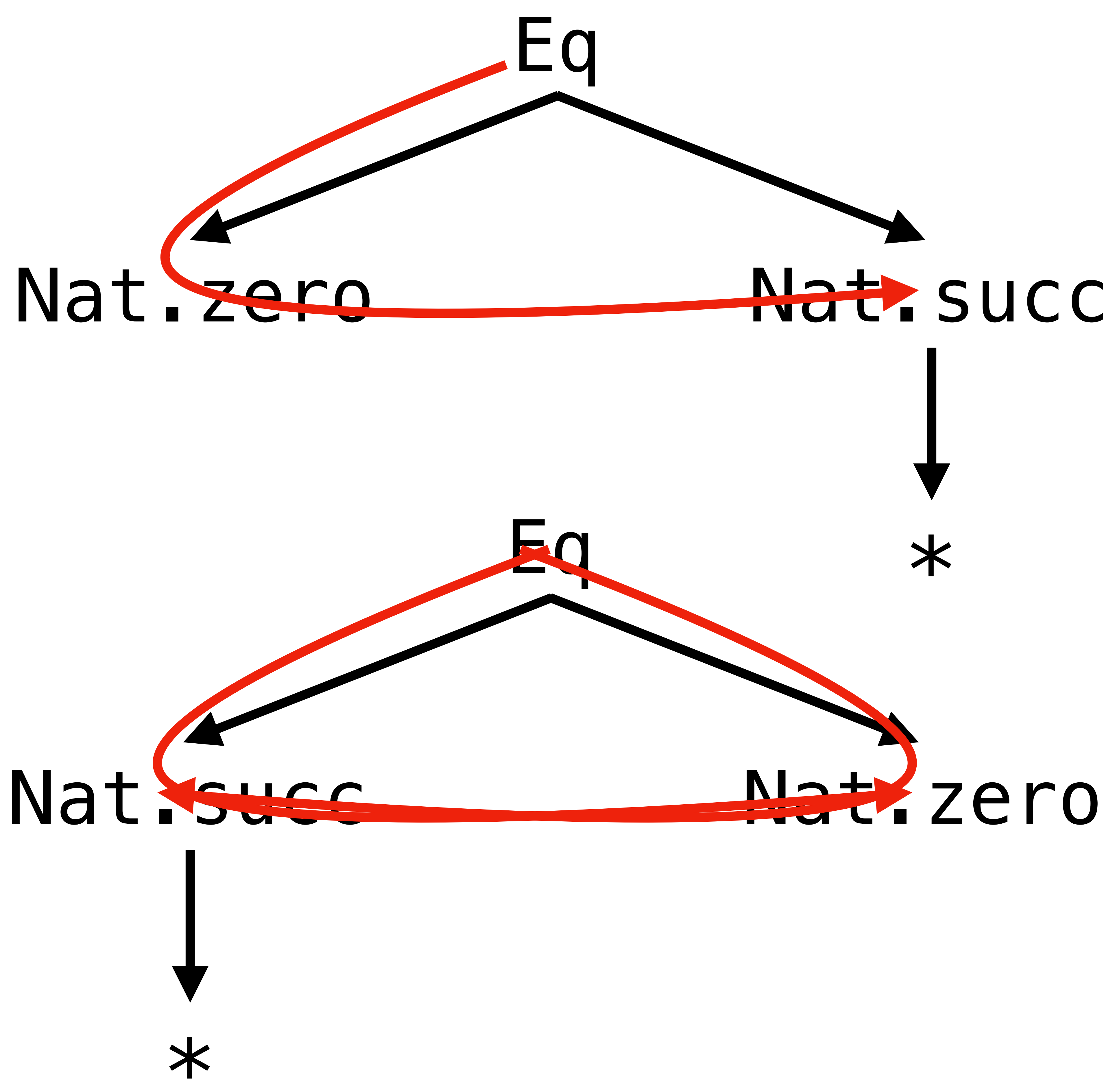
Definitional equality is stronger than `simp`.

```
variable (n : Nat) (i : Fin n) (P : {n : Nat} → Fin n → Prop)

example (h : P i) : ∃ (j : Fin (0 + n)), P j :=
  by simp only [Nat.zero_add] <;> exact Exists.intro i h
```

Uncommon, but you can always `-simp`.

Pattern Matching



Implementation

Lexicographic path ordering

Forbidding redexes syntactically

Should `?X` be a match?

```
Proj.fst ?X ≡ a
```

Enrich reasoning with type-theoretic structure, and it becomes free.

Monomorphize

```
example (a b : ℕ) : a + b = b + a := by canonical [add_comm]
```

```
Nat.add : (a.496 : Nat) → (a.497 : Nat) → Nat := [Nat.add x.493 Nat.zero ↦ x.493, Nat.add x.498 (Nat.succ b.499) ↦
  Nat.succ (Nat.add x.498 b.499)]
instAddNat : Add Nat := [instAddNat ↦ Add.mk Nat (λ a.376 a.377 ↦ Nat.add a.376 a.377)]

Add : (α.348 : Sort) → Sort := []
Add.mk : (α.361 : Sort) → (add.362 : (a.365 : α.361) → (a.366 : α.361) → α.361) → Add α.361 := []
Add.add : (α.371 : Sort) → (self.372 : Add α.371) → (a.373 : α.371) → (a.374 : α.371) → α.371 := [Add.add ∗ (Add.mk ∗
  field) arg0 arg1 ↦ field arg0 arg1]

HAdd : (α.313 : Sort) → (β.314 : Sort) → (γ.315 : Sort) → Sort := []
HAdd.mk : (α.302 : Sort) → (β.303 : Sort) → (γ.304 : Sort) → (hAdd.305 : (a.308 : α.302) → (a.309 : β.303) → γ.304) →
  HAdd α.302 β.303 γ.304 := []
HAdd.hAdd : (α.322 : Sort) → (β.323 : Sort) → (γ.324 : Sort) → (self.325 : HAdd α.322 β.323 γ.324) → (a.326 : α.322)
  → (a.327 : β.323) → γ.324 := [HAdd.hAdd ∗ ∗ ∗ (HAdd.mk ∗ ∗ ∗ field) arg0 arg1 ↦ field arg0 arg1]
instHAdd : (α.345 : Sort) → (inst.346 : Add α.345) → HAdd α.345 α.345 α.345 := [instHAdd α.341 inst.342 ↦ HAdd.mk
  α.341 α.341 α.341 (λ a.349 a.350 ↦ Add.add α.341 inst.342 a.349 a.350)]

AddCommMagma : (G.845 : Sort) → Sort := []
AddCommMagma.mk : (G.863 : Sort) → (toAdd.864 : Add G.863) → (add_comm.865 : (a.868 : G.863) → (b.869 : G.863) → Eq
  G.863 (HAdd.hAdd G.863 G.863 G.863 (instHAdd G.863 toAdd.864) a.868 b.869) (HAdd.hAdd G.863 G.863 G.863 (instHAdd
  G.863 toAdd.864) b.869 a.868)) → AddCommMagma G.863 := []
AddCommMagma.toAdd : (G.888 : Sort) → (self.889 : AddCommMagma G.888) → Add G.888 := [AddCommMagma.toAdd ∗
  (AddCommMagma.mk ∗ field ∗) ↦ field]

add_comm : (G.840 : Sort) → (inst.841 : AddCommMagma G.840) → (a.842 : G.840) → (b.843 : G.840) → Eq G.840 (HAdd.hAdd
  G.840 G.840 G.840 (instHAdd G.840 (AddCommMagma.toAdd G.840 inst.841)) a.842 b.843) (HAdd.hAdd G.840 G.840 G.840
  (instHAdd G.840 (AddCommMagma.toAdd G.840 inst.841)) b.843 a.842) := []
```

Monomorphize

```
HAdd_hAdd_0 : (a.331 : Nat) → (a.332 : Nat) → Nat := [
  HAdd_hAdd_0 n.796 Nat.zero ↦ n.796,
  HAdd_hAdd_0 n.822 (Nat.succ m.823) ↦ Nat.succ (HAdd_hAdd_0 n.822 m.823)
]
add_comm_0 : (a.457 : Nat) → (b.458 : Nat) →
  Eq Nat (HAdd_hAdd_0 a.457 b.458) (HAdd_hAdd_0 b.458 a.457) := []
```

Implementation

```
example {F : Type} [Field F] (x y : F) : x + y = x + y

Field F      HAdd F F F      Add F

                        ?G ← F

AddCommMagma FG   HAdd F F F ?G   Add FG

theorem add_comm (a b : F) : a + b = b + a) : a + b = b + a+ b = b + a
```

Implementation

```
@HAdd.hAdd Nat Nat Nat (@instHAdd Nat instAddNat) 1 2
```

↓

```
@HAdd.hAdd _ _ _ (@instHAdd _ instAddNat) _ _
```

↓

```
@HAdd.hAdd Nat Nat Nat (@instHAdd Nat instAddNat) _ _
```

↓

```
HAdd_hAdd_0 : Nat → Nat → Nat := ...
```

Destruct

```
∃ x > 0, ∀ y, y < x
```

```
Canonical.Pi : (A.2214 : Sort) → (B.2215 : (a.2217 : A.2214) → Sort) → Sort := []
Canonical.Pi.f : (A.2244 : Sort) → (B.2245 : (a.2249 : A.2244) → Sort) → (self.2246 :
  Canonical.Pi A.2244 (λ a.2252 ↦ B.2245 a.2252)) → (a.2247 : A.2244) → B.2245 a.2247 :=
  [Canonical.Pi.f ∗ ∗ (Canonical.Pi.mk ∗ ∗ field) arg0 ↦ field arg0]
Canonical.Pi.mk : (A.2223 : Sort) → (B.2224 : (a.2228 : A.2223) → Sort) → (f.2225 : (a.2229 :
  A.2223) → B.2224 a.2229) → Canonical.Pi A.2223 (λ a.2233 ↦ B.2224 a.2233) := []

And : (a.2142 : Sort) → (b.2143 : Sort) → Sort := []
And.right : (a.2171 : Sort) → (b.2172 : Sort) → (self.2173 : And a.2171 b.2172) → b.2172 :=
  [And.right ∗ ∗ (And.intro ∗ ∗ ∗ field) ↦ field]
And.left : (a.2160 : Sort) → (b.2161 : Sort) → (self.2162 : And a.2160 b.2161) → a.2160 :=
  [And.left ∗ ∗ (And.intro ∗ ∗ field ∗) ↦ field]
And.intro : (a.2148 : Sort) → (b.2149 : Sort) → (left.2150 : a.2148) → (right.2151 : b.2149)
  → And a.2148 b.2149 := []

Exists : (α.1988 : Sort) → (p.1989 : (a.1991 : α.1988) → Sort) → Sort := []
Exists.rec : (α.2044 : Sort) → (p.2045 : (a.2053 : α.2044) → Sort) → (motive.2046 : (t.2054 :
  Exists α.2044 (λ a.2057 ↦ p.2045 a.2057)) → Sort) → (intro.2047 : (w.2059 : α.2044) →
  (h.2060 : p.2045 w.2059) → motive.2046 (Exists.intro α.2044 (λ a.2067 ↦ p.2045 a.2067)
  w.2059 h.2060)) → (t.2048 : Exists α.2044 (λ a.2071 ↦ p.2045 a.2071)) → motive.2046
  t.2048 := [Exists.rec α.2036 p.2037 motive.2038 intro.2039 (Exists.intro ∗ ∗ w.2040 h.2041)
  ↦ intro.2039 w.2040 h.2041]
Exists.intro : (α.1997 : Sort) → (p.1998 : (a.2002 : α.1997) → Sort) → (w.1999 : α.1997) →
  (h.2000 : p.1998 w.1999) → Exists α.1997 (λ a.2006 ↦ p.1998 a.2006) := []

LT_lt_0 : (a.2203 : Nat) → (a.2204 : Nat) → Sort := []

Exists Nat (λ a.2137 ↦ And (LT_lt_0 Nat.zero a.2137) (Canonical.Pi Nat (λ a.2255 ↦ LT_lt_0
  a.2255 a.2137)))
```

→

```
a_w.2089 : Nat
a_h_left.2090 : LT_lt_0 Nat.zero
  a_w.2089
a_h_right.2091 : (y.2197 : Nat) →
  LT_lt_0 y.2197 a_w.2089
```

Implementation

Destruct constructs a **bijection** from the input type to the unpacked types.

```
destruct(A × B) ↔ destruct(A) ++ destruct(B)
```

```
destruct(A → B) ↔
  (destruct B).map λ b ↦ (destruct A → b)
```

Dependence makes this tricky…

When a problem is too hard,
change the problem.