*) If `a` and `b` are both `0`, return "yes".

*) If one is `0` and the other is `succ n`, return "no".

*) If `a = succ m` and `b = succ n`, then return the answer to "does `m = n`?"

Our job now is to *prove* that this algorithm always gives the correct answer. The proof that `0 = 0` is `rfl`. The proof that `0 ≠ succ n` is `zero_ne_succ n`, and the

Next

m n : ℕ

Assumptions:

h : `m ≠ n`

Goal:

`succ m ≠ succ n`

`contrapose! h` Retry

Active Goal

Objects:

m n : ℕ

Assumptions:

h : `succ m = succ n`

Goal:

`m = n`

`apply succ_inj at h` Retry

Active Goal

Objects:

m n : ℕ

Assumptions:

h : `m = n`

Goal:

`m = n`

`exact h` Retry

`level completed!` 🎉

succ_inj

```
(a b : ℕ) (h
: MyNat.succ
a =
MyNat.succ b)
: a = b
```

Statement

If a and b are numbers, then `succ_inj a b` is the proof that $(\mathrm{succ}(a) = \mathrm{succ}(b)) \implies a = b$.

More technical details

There are other ways to think about `succ_inj`.

You can think about `succ_inj` itself as a function which takes two numbers a and b as input, and outputs a proof of $(\mathrm{succ}(a) =$