Tactics

- `rw[HYPOTHESIS]`: Substitutes the left hand side of an equality in `[HYPOTHESIS]` with the right hand side
- `rel[HYPOTHESIS]`: Substitutes the left hand side of an inequality in `[HYPOTHESIS]` with the right hand side; also applies rules of modular arithmetic such as the addition, subtraction, negative, and multiplication rules.
- `addarith[HYPOTHESIS]`: basic addition and subtraction with the provided hypothesis.
- `have [HYPOTHESIS_NAME] : [STATEMENT] := by [REASONING]`: creates a new hypothesis called `HYPOTHESIS_NAME` for future reference.
- `ring`: used as justification for things related to ring properties such as addition, subtract, multiplication (and sometimes division) with variables or ring elements
- `extra`: used as justification for when an inequality differs by a neutral positive value. Eg: `y < 1 + y := by extra`
- `cancel [ITEM] at [HYPOTHESIS]`: essentially divides both sides of an (in)equality by `[ITEM]` and rewrites the hypothesis, as long at the value you are canceling is not 0. Eg: if `h1 : t^2 = t` and `h2: t ≥ 0`, then `cancel t at h1` would give `h1 : t = 1`. Note as well that you can cancel powers: if you have `h3 : a^2≥ 1` you can `cancel 2 at h3` to give `h3 : a ≥ 1`
- `numbers`: Provides justification for any statement involving only numbers. If there is a hypothesis that cannot be true (such as $0 > 7$), `numbers at [HYPOTHESIS]` can be used to show a contradiction
- `exact [HYPOTHESIS]`: Given `[HYPOTHESIS]` that matches the goal exactly, close the goal using `exact [HYPOTHESIS]`
- `apply [HYPOTHESIS]`: Used to choose a specific value for hypothesis' involving the universal quanitifier.
- `mod_cases hx : x % n`: Creates n different cases with the hypothesis `hx : x ≡ k [ZMOD n]` for $0 \leq k < n$
- `interval_cases x`: If x is bounded above and below, `interval_cases` creates a case for each possible value of x
- `contradiction`: If there are two hypothesis that connot be simultaneously true, then `contradiction` closes the goal

OR Goals and Hypothesis

- `obtain h1 | h2 := h3`: splits a hypothesis in the form $x \vee y$ into two seperate goals.
- `left`: Given a goal in the form `a ∨ b`, chooses to prove the left side, `a`
- `right`: Given a goal in the form `a ∨ b`, chooses to prove the right side `b`

AND Goals and Hypothesis

- `obtain ⟨ h1 , h2 ⟩ := h3`: splits a hypothesis in the form $x \wedge y$ into two sepreate hypothesis's
- `constructor`: Splits a goal involving AND into two seperate goals to be closed sequentially

Existential Quanitifier Goals and Hypothesis

- `use [VALUE(S)]`: For goals involving existance, provided `[VALUE(S)]` are the value(s) for the quanities that we must prove the existance of.
- `obtain ⟨ x, hx ⟩ := h`: Creates a value and a hypothesis from another hypothesis that has the existential quantifer in it. Eg. if `h1 : ∃ a : ℝ, a * t < 0` then, `obtain ⟨ x, hx ⟩ := h1` gives `hx : x * t < 0` (often used with even/odd and divisibility proofs)

Universal Quantifier Goals and Hypothesis

- `intro x`: For goals that contain JUST the universal quantifier, introduces a variable `x` for use
- `intro x hx`: For goals that contain implication, introduces a variable `x` and a hypothesis `hx` such that `x` satisfies the proposition.

Lemmas

- `apply ne_of_lt`: Changes any goal in the form of $x \neq y$ to $x < y$ (usually paired with `apply ne_of_gt`)
- `apply ne_of_gt`: Changes any goal in the form of $x \neq y$ to $y < x$ (usually paired with `apply ne_of_lt`)
- `le_or_lt x y`: Creates an OR hypothesis of the form `x ≤ y ∨ x < y`
- `le_or_gt x y`: Creates an OR hypothesis of the form `x ≤ y ∨ x > y` (best used with hypothesis in the form $x \neq y$)
- `apply abs_le_of_sq_le_sq'`: if `x ^ 2 ≤ y ^ 2` and `0 ≤ y`, then `-y ≤ x ∧ x ≤ y`
- `apply le_antisymm`: Changes a goal in the form of `a = b` into two subgoals `a ≤ b` and `a ≥ b`
- `rw[mul_eq_zero] at [HYPOTHESIS]`: If you have `[HYPOTHESIS]` in the form `xy = 0`, then `rw[mul_eq_zero] at [HYPOTHESIS]` gives `[HYPOTHESIS : x = 0 ∨ y = 0]`
- `Int.even_or_odd n`: Creates a hypothesis in the form `Even n ∨ Odd n` that any integer `n` must be even or odd
- `apply Int.not_dvd_of_exists_lt_and_lt`: Used to prove that one integer does not divide another. Creates a goal in the form `∃ q, b * q < q ∧ a < b * (q + 1)`, or essentially that if `a` is between two multiple of `b`, then `b` cannot divide `a`

Definitions

- `dsimp[DEFINITION] at *`: rewrites all written occurences of `[DEFINITION]` in terms of its mathematical definition.
- `Odd (n : ℤ)`: `n = 2 * k + 1`
- `Even (n : ℤ)`: `n = 2 * k`
- `( · | · )`: Definition of divisibility, used to unpack goals in the form `a | b`