18 - Our Own Definition

Lean: First Steps

Tariq Rashid

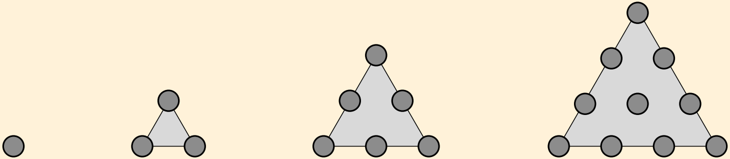
December 3, 2024

Using Our Own Lemma

- Previously we used Mathlib's definition of odd and even numbers.
- Here we'll create our own definition of triangle numbers.

Triangle Numbers

- The following picture illustrates **triangle numbers**. The first few are 1, 3, 6, 10, 15, 21 and 28.



1 3 6 10

- In general, the nth triangle number is

$$\frac{n \cdot (n+1)}{2}$$

Task

- The Mathlib definition of `Odd` doesn't produce the *n*th odd number. The definition is a **proposition** about a supplied number.
  - `Odd 3` is true
  - `Odd 4` is false
- Our task is to create a definition of triangle numbers that is a proposition.
  - True only if the supplied number is actually a triangle number.

- A proposition for a triangle number T could be:

$$\exists n \in \mathbb{N} \quad [\ T = \frac{n \cdot (n+1)}{2}\ ]$$

- This proposition is only true if T is a triangle number.
  - That is, if T can be expressed in the form $n \cdot (n+1)/2$ for some natural number n.
- When working with natural numbers, we should be cautious about dividing them.
  - In this case division is safe because either n or $(n+1)$ is an even number, and so $n \cdot (n+1)$ is divisible by 2.

Code

```
-- 18 - Our Own Definition

import Mathlib.Tactic

def Triangle (a : ℕ) : Prop := ∃ n, 2 * a = n * (n + 1)

example : Triangle 10 := by
  dsimp [Triangle]
  use 4
```

Code

- Keyword `def` signals we're about to create a named definition. Here the name is `Triangle`.
- After that is a declaration of variables, here `a : ℕ`.
  - Round brackets require anyone using the definition to provide `a` as a parameter.
- `Prop` specifies `Triangle` will be a **proposition**, a statement that can be true or false.
- After `:=` is the detail of the definition of a triangle number, `∃ n, 2 * a = n * (n + 1)`.

Definitions

- The following summarises the structure of simple definitions.

name variables type definition

```
def Triangle (a : ℕ) : Prop := ∃ n, 2 * a = n * (n + 1)
```

Minimal Example

- When creating our own definitions, it is considerate to provide a minimal example that illustrates how to use the definition.
- Here the example is a proof of `Triangle 10`, a proposition that says 10 is a triangle number.
  - The `dsimp [Triangle]` **unfolds** the definition in the Infoview.
- Because the definition is a "there exists" statement, the proof is resolved by a simple `use 4`.

Infoview

- Placing the cursor before `dsimp [Triangle]` in the illustrative example shows the proof objective.

```
⊢ Triangle 10
```

- Moving the cursor to the start of the next line shows the goal with the definition of `Triangle` unfolded.

```
⊢ ∃ n, 20 = n * (n + 1)
```

Types & Terms

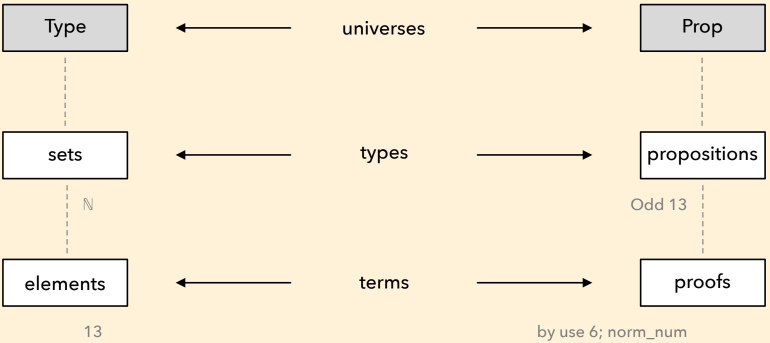
- Compare the definition of `Triangle` with a definition of `Triple`:

```
def Triangle (a : ℕ) : Prop := ∃ n, 2 * a = n * (n + 1)

def Triple (a : ℕ) : ℕ := 3 * a
```

- The **type** of `Triangle` is `Prop`, a proposition.
    - The **type** of `Triple` is `ℕ`, a natural number.
- We say the detail `∃ n, 2 * a = n * (n + 1)` is a **term** of **type** `Prop`.
    - `3 * a` is a **term** of **type** `ℕ`.

Terms & Types



- Perhaps surprisingly, proofs are a term of type proposition.

Easy Exercise

- Create a definition of **square numbers** named `Square`.
- It should be a **proposition** which is only true if a given number can be written in the form n^2, for some natural number n.
- Write a proof showing 25 is a square number, illustrating the use of `Square`.