21 - Recursion

Lean: First Steps

Tariq Rashid

January 19, 2025

Recursion

- The famous Fibonacci sequence

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ \ldots$$

- From 2 onwards, each number is the sum of the previous two.
- This is an example of a recursive definition.

Task

- The following is a **recursively** defined function $f_n : \mathbb{N} \to \mathbb{N}$.

$$f_n = \begin{cases} 0 & \text{if } n = 0 \\ 2n - 1 + f_{n-1} & \text{otherwise} \end{cases} \tag{1}$$

- Prove this recursively defined function has a neater closed form,

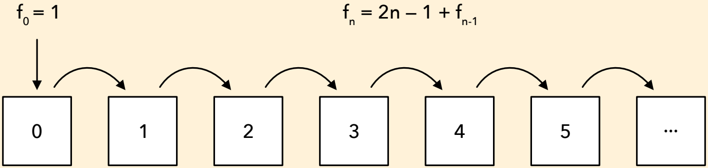
$$f_n = n^2$$

Task

- Let's see how it works.
  - For $n = 0$, the definition tells us $f_0 = 0$.
  - For $n = 1$, the definition tells us $f_1 = 2 \cdot 1 - 1 + f_0$. Using $f_0 = 0$, this becomes $f_1 = 1$.
  - For $n = 2$, the definition tells us $f_2 = 2 \cdot 2 - 1 + f_1$. Using $f_1 = 1$, this gives us $f_2 = 4$.
- What makes the definition **recursive** is that, apart from f_0, each value of the function f_n is defined in terms of an earlier value f_{n-1}.
- Informally, the function is "defined in terms of itself."

Similarity To Induction

- The process of calculating values of f_n looks similar to induction .



- There is a base case $f_0 = 1$.
- There is a way to get f_n from f_{n-1}.

Maths

- We'll try a proof by induction. Let $P(n)$ be the proposition $f_n = n^2$.

- The **base case** $P(0)$ is the proposition $f_0 = 0^2$. The definition tells us $f_0 = 0$ so the base case $P(0)$ is indeed true.

- The **inductive step** is the implication $P(n) \implies P(n+1)$. To prove this we assume $f_n = n^2$ and derive $f_{n+1} = (n+1)^2$.

    - The definition (1) tells us $f_{n+1} = 2(n+1) - 1 + f_n$.

    - By assumption $f_n = n^2$, so $f_{n+1} = 2(n+1) - 1 + n^2$.

    - That $2(n+1) - 1 + n^2$ simplifies to $(n+1)^2$.

    - Putting all this together, we have $f_{n+1} = (n+1)^2$.

- So by induction, $f_n = n^2$ is true for all n.

Maths

$$P(n) : f_n = n^2 \qquad \text{proof objective}$$

base case $P(0)$

$$\begin{aligned} f(0) &= 0 && \text{by definition (1)} \\ &= 0^2 && \text{so } P(0) \text{ is true} \end{aligned}$$

inductive step $P(n) \implies P(n+1)$

$$P(n) : f_n = n^2 \qquad \text{induction hypothesis} \tag{2}$$

$$\begin{aligned} f_{n+1} &= 2 \cdot n - 1 + f_n && \text{by definition (1)} \\ &= 2 \cdot n - 1 + n^2 && \text{by (2)} \\ &= (n+1)^2 && \text{so } P(n+1) \text{ is true} \end{aligned}$$

$$P(n) : f_n = n^2 \qquad \text{by induction} \quad \square$$

Code 1/2

```
-- 22 - Recursion

import Mathlib.Tactic

def f : ℕ → ℕ
  | 0 => 0
  | n + 1 => 2 * n + 1 + f n

#eval f 2
```

Code 2/2

```
example {n: ℕ} : f n = n^2 := by
  induction n with
  | zero =>
    calc
      f 0 = 0 := by rw [f]
      _ = 0^2 := by norm_num
  | succ n ih =>
    calc
      f (n + 1) = 2 * n + 1 + f n := by rw [f]
      _ = 2 * n + 1 + n^2 := by rw [ih]
      _ = (n + 1)^2 := by ring
```

Code

- The **first part** is the recursive definition of `f`.
- The `f : ℕ → ℕ` declares `f` as a function which maps `ℕ` to `ℕ`.
- We can see how it maps `0` to `0`, and `n + 1` to `2 * n + 1 + f n`.
- Here `f n` means "`f` applied to `n`", which is just f_n.
- Our earlier definition (1) mapped $n \mapsto 2n - 1 + f_{n-1}$. We can't use f_{n-1} in the Lean definition because $(n-1)$ does not exist for all natural numbers n. By replacing n with $n+1$, we get the equivalent $n+1 \mapsto 2n+1+f_n$.

Code

- The **second part** tests our function behaves as we expect.
- The `#eval` is a special instruction which evaluates an expression and prints the answer in the Infoview.
- Just as `#eval 2 + 3` will print `5`, `#eval f 2` will evaluate the function `f` applied to `2`, and print the answer.
- Try evaluating `f` with other parameters.

Code

- The **third part** is the proof of the proposition `f n = n^2`.
- This **proof by induction** is not very different to the one we saw in the last chapter.
  - The **base case** needs to prove `f 0 = 0 ^ 2`. The first step `f 0 = 0` is justified by `rw [f]` which uses the first part of the recursive definition `| 0 => 0`.
  - The **inductive step** needs to prove `f (n + 1) = (n + 1) ^ 2`. The first step which equates `f (n + 1)` with `2 * n + 1 + f n` is similarly justified by `rw [f]`, which this time uses the recursive part of the definition.

Recursion & Induction

- We saw earlier the similarity between the **induction process** and a **recursively defined function**.
- We now see the similarity between an **induction proof** and a **recursively defined function**.
- The two are clearly linked. Inside Lean, induction and recursion are unified.

Infoview

- Placing the cursor after `#eval f 2` shows the result of the function `f` applied to `2`.

```
4
```

- Changing the code to `#eval f 3` shows the result of the function `f` applied to `3`.

```
9
```

Infoview

- Placing the curser just before `| succ n ih =>` shows the goal for the inductive step, and also the inductive hypothesis `ih`.

```
ih : 2 ^ n ≥ n + 1
⊢ 2 ^ (n + 1) ≥ n + 1 + 1
```

Easy Exercise

- The following is a recursively defined function $g_n : \mathbb{N} \to \mathbb{Z}$,

$$g_n = \begin{cases} 1 & \text{if } n = 0 \\ (-1) \cdot g_{n-1} & \text{otherwise} \end{cases}$$

- Write a recursive definition for g_n in Lean, then show, by induction, that $g_n = (-1)^n$.