THE PIGEONHOLE PRINCIPLE

JONATHAN PRIETO-CUBIDES

Abstract. There are many formulations of the so-called Pigeonhole principle, some of them proved in constructive mathematics and formalized in some proof-assistants. In this work, we show one these formulations proved in the context of Univalent Type theory and formalized in the proof-assistant Agda.

1. Introduction

The pigeonhole principle says that *if you put n pigeons in m holes, with $m < n$, then at least one hole has more than one pigeon in it.*

In [1] the authors give the following formulation (Theorem 1.1) of the Pigeonhole Principle joint with a constructive proof that it holds. The proof can be found formalized in `Coq`[1].

Theorem 1.1 (Lemma 2.15.6 in [1])**.** For all $N : \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ such that $f(n) < N$ for all $n < N + 1$, there exist $m < n < N + 1$ such that $f(n) = f(m)$.

The function f above intends to maps the number of a pigeon in $\{0 \cdots n-1\}$ to the number of its hole in $\{0 \cdots m - 1\}$.

In this work, we consider an equivalent formulation in Theorem 1.2, which can be seen as a generalization of Theorem 1.1. The formal statement says, there is no injective function that can go from a finite set to a smaller one. We denote the finite set of n elements with $[n]$.

Theorem 1.2 (Pigeonhole Theorem)**.** For any $n, m : \mathbb{N}$ when $m < n$, a function $f : [n] \to [m]$ can not be injective.

Another similar formulation can be stated when instead of finite sets, we consider for any function, the cardinality of its domain and its image. We give the following formulations that follow the same fashion of Theorem 1.2.

For any $n : \mathbb{N}$, $f : [n] \to [n]$.

- If f is a surjective, then f is injective, and
- If f is injective, then f is surjective.

2. Notation

2.1. **Agda formalization.** To check the proof in Section 3 of Theorem 1.2, we use `Agda v2.6.0` without the *Axiom K* to be consistent with Univalent mathematics. We also import a library to work with Homotopy Type Theory called `MiniHoTT`[2] that includes some type definitions and theorems we need.

[1] In the `Coq` standard library https://coq.inria.fr/library/Coq.Sets.Image.html.
[2] Available on http://jonaprieto.github.io/mini-hott.

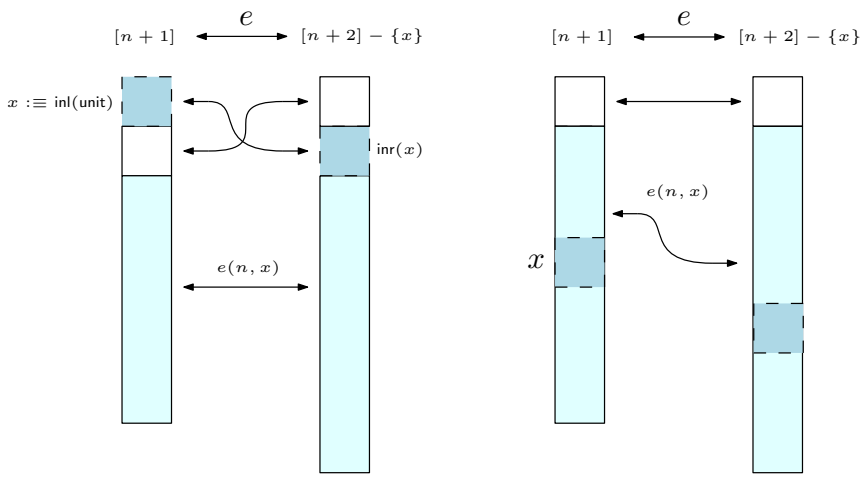

FIGURE 1. Construction of the equivalence e in (Lemma 3.3). The directions of the arrows (forward and backward) correspond to the functions in Definition 3.1 and Definition 3.2, respectively.

```
{-# OPTIONS -without-K #-}
open import MiniHoTT
```

2.2. **Types.** We assume the reader is familiar with type theory notation as in Book HoTT [2] and with the basic Agda syntax. For any type $A : \mathsf{Type}$, we define in Equation (2.1) the type $A - \{x\}$ as the type[3] of elements of which are different with x. We will use this type to talk about the finite set $[n]$ without a point x, i.e., $[n] - \{x\}$. The finite set of n elements in Agda is denoted by `⟦ n ⟧`. We are using for finite sets the definition that says $[n+1] :\equiv \mathbb{1} + [n]$ and $[0] :\equiv \mathbb{0}$. Propositional equality is denoted by $(\equiv)$ instead of $(=)$. The type $\mathbb{N}$ for natural numbers has two constructors `zr`, and `succ`. The coproduct of types A and B is denoted by $A + B$ and it has two introduction rules named `inr`, and `inl`. We use the direct composition of functions also called *diagramatic* composition, denoted in Agda by (`f :> g`) for $f : A \to B$ and $g : B \to C$, for types A, B, and C.

$$A - \{x\} :\equiv \sum_{a:A} (a \equiv x) \to \bot. \tag{2.1}$$

3. A proof of Theorem 1.2

We first need to show there exists an equivalence between two (finite) types that differs only by one point. So we define the following (recursive) functions `e→` and `e←`.

Definition 3.1.

```
e→ : ∀ (n : ℕ) → (x : ⟦ succ n ⟧)
  → ⟦ n ⟧ → ((⟦ succ n ⟧) \\ (x))
```

[3]We denote this symbol, minus, as double-back-slashes in Agda.

```
e→ (succ n) (inl x) y = inr y , λ ()
e→ (succ n) (inr x) (inl y) = inl _ , λ ()
e→ (succ n) (inr x) (inr y)
  = (inr (π₁ $ e→ n x y)) , λ p → π₂ (e→ n x y) (inr-is-injective p)
```

Definition 3.2.

```
e← : ∀ (n : ℕ) → (x : ⟦ succ n ⟧)
   ──────────
   → (⟦ succ n ⟧ \\ x) → ⟦ n ⟧

e← zr (inl ⋆) (inl ⋆ , b) = b (ap inl idp)
e← (succ n) (inl x) (inl x₁ , b) = inl _
e← n (inl x) (inr y , b)  = y
e← (succ zr) (inr x) (inl x₁ , π₄) = x
e← (succ (succ n)) (inr x) (inl x₁ , π₄) = inl ⋆
e← (succ n) (inr x) (inr y , π₄)
  = inr (e← n x (y , (π₄ ∘ (ap inr ))))
```

Lemma 3.3. For any $n : \mathbb{N}$, $x : [n+1]$, the types $[n]$ and $[n+1]-\{x\}$ are (homotopy) equivalent. We shall denote this equivalence as $e(n, x)$.

Sketch of the Proof. We can show that the equivalence follows by showing `e→` has as inverse the function `e←`. The corresponding homotopies (e.g `e→ :> e← ~ id`) also follow but non-trivially (See the `Agda` source for all details). □

Proposition 3.4. The `e→` and `e←` functions are both injective functions.

Proof. By the equivalence in Lemma 3.3, we can get a (proper) bijection since the corresponding types of that equivalence are in fact (homotopy) sets. Bijections are injections, so the result follows. For the function `e←` repeat the argument but we the symmetry of the aforementioned equivalence. □

Proof of Theorem 1.2. The idea here is basically consider the point $x :\equiv \mathsf{inl}(\mathsf{unit})$ (that represents the first point on each type $[n+1]$ and $[m+1]$) and its image by the function f. Then, we construct a function $h : [n+1]-\{x\} \to [m+1]-\{x\}$ which acts as the restriction of the given function f. Because f is injective, and any restriction of it is also injective, h is injective. Now, we do induction on n, and the induction hypothesis says any function $g : [n] \to [m]$, g is not injective.

$$\begin{array}{ccc} [n+1]-\{x\} & \xrightarrow{\quad h \quad} & [m+1]-\{f(x)\} \\ {\scriptstyle e(n,x)}\uparrow & & \downarrow{\scriptstyle e^{-1}(m,f(x))} \\ [n] & \xrightarrow[\quad g \quad]{} & [m] \end{array}$$

FIGURE 2. Construction of the function g. In particular, we use $x :\equiv \mathsf{inl}(\mathsf{unit})$.

But with h we can construct an injective function from $[n] \to [m]$ as follows. Recall the equivalence functions given by Lemma 3.3. By composing

h with those functions, as it is stated in the diagram showed above, we get the function g. Since composition of injective functions yields an injective function (see Proposition 3.4), therefore, g is injective. As we expect by applying the induction hypothesis, we get the absurd.

The Agda term for this proof is the following.

```
pigeon-theorem
  : ∀ (n m : ℕ)
  → (m < n) → (f : ⟦ n ⟧ → ⟦ m ⟧ )
  ----------------------
  → ¬ (isInjective f)

pigeon-theorem (succ n) zr * f f-is-inj = f (inl *)
pigeon-theorem (succ n) (succ m) p f f-is-inj
  = pigeon-theorem n m (succ-<-inj {n = m} p) g g-is-injective
  where
  h : (⟦ succ n ⟧ \\ inl unit) → (⟦ succ m ⟧ \\ (f (inl unit)))
  h w@(a , b) = (f a , λ b → π₂ w $ f-is-inj b)

  h-is-inj : isInjective h
  h-is-inj {a₁ , b₁} {(a₂ , b₂)} p
    = pair= (f-is-inj (ap π₁ p)
            , pi-is-prop (λ a x ()) (tr _ (f-is-inj (ap π₁ p)) b₁) b₂)

  g : ⟦ n ⟧ → ⟦ m ⟧
  g = lemap (e n (inl *)) :> h :> remap (e m (f (inl *)))

  g-is-injective : isInjective g
  g-is-injective =
    ∘-injectives-is-injective _ (π₁ $ e-is-bijection _ _) _
      (∘-injectives-is-injective _ h-is-inj _ (π₁ $ inv-e-is-bijection _ _))
```

□

References

[1] The Homotopy Type Theory and Univalent Foundations CAS project. *Symmetry Book*. Centre for Advanced Study, at the Norwegian Academy of Science and Letters, 2019.

[2] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.