Bridging Combinatorial and Algebraic proof

An Algebraic Approach with Agda

游棫荃

Introduction

What are combinatorial and algebraic proofs (or argument) ?

(Ross TE1.12) Consider the following combinatorial identity:

$$\boxed{\sum_{k=1}^{n}}\boxed{k}\boxed{\binom{n}{k}} = \boxed{n}\boxed{\cdot\, 2^{n-1}}.$$

(a) Present a combinatorial argument for this identity by considering a set of n people and determining, in two ways, the number of possible selections of a committee of any size and a chairperson for the committee.

Hint:

(i) How many possible selections are there of a committee of size k and its chairperson?

(ii) How many possible selections are there of a chairperson and the other committee members?

(b) Now present an algebraic argument for this identity.

Hint: Differentiate the binomial theorem.

Algebraic proof

To prove the identity algebraically, use the binomial theorem:

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

Differentiate both sides with respect to x:

$$n(1+x)^{n-1} = \sum_{k=1}^{n} k \binom{n}{k} x^{k-1}$$

Now, set $x = 1$:

$$n \cdot 2^{n-1} = \sum_{k=1}^{n} k \binom{n}{k}$$

This gives the desired identity.

Comparison

Combinatorial proofs are often more intuitive and easier to understand due to their reliance on counting and reasoning about sets, which often makes them more accessible.

Algebraic proofs, while more formal, typically require the buildup of numerous auxiliary lemmas and the use of more complex mathematical tools such as calculus and generating functions. This complexity can make proving intricate combinatorial identities in Agda a challenging task.

Goal

1. Construct the correctness of combinatorial proofs.

$$S_n \simeq S_m \to n \equiv m$$

2. Explore the equivalence between combinatorial and algebraic proofs.

$$S_n \simeq S_m \leftrightarrow n \equiv m$$

3. Automate the process of transforming proofs.

Key ideas

1. The same algebraic structure underlying both sets and natural numbers
2. FinSet: Serves as a medium between sets and natural numbers
3. Embedding: An sufficient condition for transforming proofs

$$S_n \simeq S_m = F_n \sim F_m \xrightarrow{E\ \sim} n \equiv m$$

1. Algebraic structure

1.1 Corresponding Operations

For sets S_n and S_m of sizes n and m respectively:

- $n + m$ corresponds to $S_n \uplus S_m$, where $\uplus$ denotes the disjoint union.
- $n * m$ corresponds to $S_n \times S_m$, where $\times$ denotes the Cartesian product.
- The binomial coefficient $\binom{n}{m}$ corresponds to $\binom{S_n}{m}$, the collection of all subsets of S_n of size m.
- The permutation count P_m^n corresponds to the collection of all sequences of S_n of size m.

1.2 N, List and Set

$\mathbb{N}$	$List$	Set
0	$[]$	$\bot$
1	$[*]$	$\top$
$+$	$++$	$\uplus$
$*$	$cartesianProduct$	$\times$
$\equiv$	$\lambda x\ y \to length\ x \equiv length\ y$	$\simeq$

cartesianProduct : List A → List B → List (A × B)

2. FinSet

2.1 Why we need medium (FinSet)?

$$S_n \simeq S_m \longrightarrow n \equiv m$$

2.1 Why we need medium (FinSet)?

FinSet

$\mathbb{N}$	$List$	Set
$+$	$++$	$\uplus$
$*$	$cartesianProduct$	$\times$
$\equiv$	$\lambda x\ y \to length\ x \equiv length\ y$	$\simeq$

2.2 Membership of List

```
data _∈_ (a : A) : (x : List A) → Set i where
  here  : ∀ {x} → a ∈ (a ∷ x)
  there : ∀ {b} {x} → (a∈x : a ∈ x) → a ∈ (b ∷ x)

_∉_ : (a : A) → (x : List A) → Set i
a ∉ x = ¬ (a ∈ x)

data _∈₁_ (a : A) : (x : List A) → Set i where
  here₁  : ∀ {x} → (a∉x : a ∉ x) → a ∈₁ (a ∷ x)
  there₁ : ∀ {b x} → (a∉b : a ∉ [ b ]) → (a∈₁x : a ∈₁ x) → a ∈₁ (b ∷ x)
```

2.3 FinSet

A finite set is defined as "the existence of a list that enumerates all the inhabitants of the type."

```
record FinSet {i : Level} : Set (lsuc i) where
  field
    Carrier : Set i
    list : List Carrier

    -- Every inhabitant of Carrier is a member of the list.
    enum : (aₑ : Carrier) → aₑ ∈ list

    -- Every element in list appears exactly once.
    once : (a₁ : Carrier) → a₁ ∈ list → a₁ ∈₁ list
```

2.3 Operators and relation

```
_R+_  = λ X Y → record
       { Carrier = Carrier X ⊎ Carrier Y
       ; list = (map inj₁ (list X)) ++ (map inj₂ (list Y))
       ; ...}


_R*_ = λ X Y → record
      { Carrier = Carrier X × Carrier Y
      ; list = cartesianProduct (list X) (list Y)
      ; ...}


_~_  = λ X Y → Carrier X ≃ Carrier Y
```

$List$	Set
$++$	$\uplus$
$cartesianProduct$	$\times$
$\lambda x\ y \to length\ x \equiv length\ y$	$\simeq$

$$F_n \sim F_m$$

$$FinSet$$

$$F_n$$

$$F_m$$

3. Embedding

3.1 Why we need Embedding?

$$EF : FinSet \to \mathbb{N} \qquad\qquad EF = length \circ list$$

$$\begin{array}{c}
F_n +_{FS} F_m \sim_{FS} F_m +_{FS} F_n \\
\Big\downarrow E \sim \\
EF(F_n +_{FS} F_m) \equiv EF(F_m +_{FS} F_n) \\
\rightsquigarrow E+ \\
EF(F_n) + EF(F_m) \equiv EF(F_m) + EF(F_n) \\
\rightsquigarrow EFF \\
n + m \equiv m + n
\end{array}$$

3.2 Embedding

$FinSet$	$\mathbb{N}$

$$EF : FinSet \to \mathbb{N}$$

Example 1 : Commutativity of Addition

```
(n m : ℕ) → n + m ≡ m + n
```

Commutativity of Addition Combinatorial proof

$$X \uplus Y \underset{from}{\overset{to}{\rightleftarrows}} Y \uplus X$$

Commutativity of Addition Transform into Algebraic proof

```
algeb-pf : (n m : ℕ) → n + m ≡ m + n
algeb-pf n m =
  begin
    n + m
  ≡⟨ sym (cong₂ _+_ (EFF n) (EFF m)) ⟩
    EF (F n) + EF (F m)
  ≡⟨ sym (E+ (F n) (F m)) ⟩
    EF ((F n) R+ (F m))
  ≡⟨ E~ ((F n) R+ (F m)) ((F m) R+ (F n)) (combi-pf n m)  ⟩
    EF ((F m) R+ (F n))
  ≡⟨ E+ (F m) (F n) ⟩
    EF (F m) + EF (F n)
  ≡⟨ cong₂ _+_ (EFF m) (EFF n) ⟩
    m + n
  ∎
```

$$
\begin{array}{c}
F_n +_{FS} F_m \sim_{FS} F_m +_{FS} F_n \\
\Big\downarrow E\sim \\
EF(F_n +_{FS} F_m) \equiv EF(F_m +_{FS} F_n) \\
\Big\downarrow E+ \\
EF(F_n) + EF(F_m) \equiv EF(F_m) + EF(F_n) \\
\Big\downarrow EFF \\
n + m \equiv m + n
\end{array}
$$

```
EFF : ∀ (n : ℕ) → EF (F n) ≡ n
```

Commutativity of Addition Comparison

```
-- By framework
combi-pf : (n m : ℕ) → ((F n) R+ (F m)) ~ ((F m) R+ (F n))
combi-pf n m = record {...}

algeb-pf : (n m : ℕ) → n + m ≡ m + n
algeb-pf n m = auto combi-pf n m


-- Normal, plfa
lemma-1 : ∀ (m : ℕ) → m + zero ≡ m
lemma-1 zero = refl
lemma-1 (suc m) rewrite lemma-1 m = refl

lemma-2 : ∀ (m n : ℕ) → m + suc n ≡ suc (m + n)
lemma-2 zero n = refl
lemma-2 (suc m) n rewrite lemma-2 m n = refl

+-comm : ∀ (m n : ℕ) → m + n ≡ n + m
+-comm m zero rewrite lemma-1 m = refl
+-comm m (suc n) rewrite lemma-2 m n | +-comm m n = refl
```

$$\begin{tikzcd} X \uplus Y \arrow[r, shift left, "to"] & Y \uplus X \arrow[l, shift left, "from"] \end{tikzcd}$$

$$\begin{tikzcd} inj_1\ x \arrow[rrd, leftrightarrow, "to"] & & inj_1\ y \\ inj_2\ y \arrow[rru, leftrightarrow, "from"'] & & inj_2\ x \end{tikzcd}$$

Example 2 : Associativity of Product

```
(n m l : ℕ) → n * (m * l) ≡ (n * m) * l
```

Associativity of Product Combinatorial proof

$$\begin{tikzcd} X \times (Y \times Z) \arrow[r, shift left, "to"] & (X \times Y) \times Z \arrow[l, shift left, "from"] \end{tikzcd}$$

Associativity of Product Comparison

```
-- By framework
combi-pf2 : (n m l : ℕ) → ((F n) R* ((F m) R* (F l))) ~ (((F n) R* (F m)) R* (F l))
combi-pf2 n m l = record {...}

algeb-pf2 : (n m l : ℕ) → n * (m * l) ≡ (n * m) * l
algeb-pf2 n m l = auto combi-pf2 n m l


-- Normal, plfa
*-assoc : ∀ (m n p : ℕ) → (m * n) * p ≡ m * (n * p)
*-assoc zero n p = refl
*-assoc (suc m) n p =
  begin
    (suc m * n) * p
  ≡⟨ *-distrib-+ n (m * n) p ⟩
    (n + m * n) * p
  ≡⟨ *-assoc m n p ⟩
    n * p + (m * n) * p
  ≡⟨⟩
    suc m * (n * p)
  ∎
```

$$X \times (Y \times Z) \underset{from}{\overset{to}{\rightleftarrows}} (X \times Y) \times Z$$

$$(i, (j, k)) \underset{from}{\overset{to}{\rightleftarrows}} ((i, j), k)$$

Conclusion

1. Abstraction Achievement:
   Creating a promising proof system in Agda.

2. Term Automation:
   Once Term automation is complete, the system could lower proof difficulty and improve readability in Agda.

3. Unfinished Work:
   The proof of FinSet multiplication and the inject-∈ Lemma are still pending.

Future Studies

1. Additional Operations to Implement

   ```
   Σ, Π, _!, P, C, _^_
   ```

2. Automatic Proof Generation Using data Term

   ```
   data Term `_ _`+_ _`*_ `Σ[_∈_]_ `Π[_∈_]_ [_]`! `P[_,_] `C[_,_]
   ```

3. Term Reasoning

   ```
   ≈-begin_    _≈⟨_⟩_    _≈-∎
   ```

Thank you for listening

References

- FRUMIN, Dan, et al. "Finite sets in homotopy type theory." *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2018. pp. 201-214.

- EDMONDS, Chelsea. *Formalising Combinatorial Structures and Proof Techniques in Isabelle/HOL*. 2024. PhD Thesis.

- RIJKE, Egbert; SPITTERS, Bas. "Sets in homotopy type theory." *Mathematical Structures in Computer Science*. 2015, 25(5): 1172-1202.

- The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. arXiv preprint arXiv:1308.0729, 2013.