Bridging Combinatorial and Algebraic proof: An Algebraic Approach with Agda

Yu-Chuan YU

Advisor : Liang-Ting Chen

August 29, 2024

Abstract

In combinatorics, proving combinatorial identities often involves either double counting the same set or constructing a bijection between two sets. In contrast, algebraic proofs typically rely on operations like addition and multiplication on natural numbers or techniques such as calculus and generating functions. Our study focuses on the relationship between these two proof methods. There is a corresponding relationship between operations on natural numbers and sets, such as addition corresponding to disjoint union ($\uplus$) and multiplication corresponding to the Cartesian product ($\times$). However, it is not trivial to define the size of a type in type theory. To address this, we define the type 'FinSet', which also incorporates specific lists as a medium. By defining the size function $EF : FinSet \to \mathbb{N}$ on 'FinSet', we find that as long as the embedding properties are satisfied, this ensures the correctness of the proof transformation."

Keywords. Agda, Commutative ring, Combinatorial reasoning

e-mail: yuyuch0303@gmail.com

Introduction Combinatorial and algebraic proofs represent two distinct approaches to mathematical reasoning. Combinatorial proofs often employ double counting or bijections, demonstrating equal set sizes or one-to-one correspondences between sets. In contrast, algebraic proofs leverage operational properties and may involve complex mathematical tools like calculus or generating functions. Consider the combinatorial identity:

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

This identity can be proved using both methods:
1. Combinatorial Proof: Count the ways to select a committee of any size and a chairperson from n people in two ways: a) Select a committee of size k and its chairperson. b) Select a chairperson and the remaining committee members.
2. Algebraic Proof: Differentiate the binomial theorem.

Comparison and Motivation The example in the introduction highlights key differences between combinatorial and algebraic proof methods:

- **Intuitiveness**: Combinatorial proofs often provide more intuitive understanding, relying on counting principles and set-based reasoning. This approach can make complex identities more accessible to a broader audience.
- **Formality**: Algebraic proofs, while more formal, typically require extensive auxiliary lemmas and sophisticated mathematical tools like calculus or generating functions. This formality can pose challenges when implementing proofs in systems like Agda, especially for intricate combinatorial identities.
- **Accessibility**: The intuitive nature of combinatorial proofs can make them easier to grasp, particularly for those less versed in advanced algebraic techniques.

Given these distinctions, our research focuses on:

1. Establishing the correctness of combinatorial proofs. $S_n \simeq S_m \rightarrow n \equiv m$
2. Investigating the equivalence between these approaches. $S_n \simeq S_m \leftrightarrow n \equiv m$
3. Automate the process of transforming proofs.

By bridging the gap between combinatorial intuition and algebraic rigor, we aim to develop a unified framework that leverages the strengths of both methods. This could potentially simplify the proof process for complex combinatorial identities, especially in formal verification systems, while maintaining mathematical rigor.

Corresponding Operations between ℕ, List, and Set

- $\_+\_$ corresponds to list append $\_++\_$ and disjoint union $\_\uplus\_$.
- $\_*\_$ corresponds to `cartesianProduct` and Cartesian product $\_\times\_$.

FinSet In Agda, defining the cardinality function `card : Set` → ℕ is challenging. To address this, we bind the `Carrier : Set` to a specific `list : List Carrier` that satisfies two conditions:

1. `enum`: Every inhabitant of `Carrier` is a member of the `list`.
2. `once`: Every element in the `list` appears exactly once.

This allows us to define the cardinality function: `EF = λ X → length list X`
Additionally, we define a relation on `FinSet`: `X ∼ Y = Carrier X ≃ Carrier Y`

Embedding Embedding functions, denoted `EF`, are crucial for preserving the structure of operations during transitions between domains. They ensure that operations and equivalence relations are preserved, maintaining consistency across structures.

$$\texttt{EF : FinSet} \rightarrow \mathbb{N}$$

$$\texttt{E+} : \forall\ (X\ Y : \texttt{FinSet}) \rightarrow \texttt{EF}\ (X +_{FS} Y) \equiv (\texttt{EF}\ X) + (\texttt{EF}\ Y)$$

$$\texttt{E*} : \forall\ (X\ Y : \texttt{FinSet}) \rightarrow \texttt{EF}\ (X *_{FS} Y) \equiv (\texttt{EF}\ X) * (\texttt{EF}\ Y)$$

$$\texttt{E}\sim : \forall\ (X\ Y : \texttt{FinSet}) \rightarrow X \sim Y \rightarrow (\texttt{EF}\ X) \equiv (\texttt{EF}\ Y)$$

Given `EF = λ X → length list X`, this can be rewritten as:

$$\texttt{E}\sim : \forall\ (X\ Y : \texttt{FinSet}) \rightarrow \texttt{Carrier}\ X \simeq \texttt{Carrier}\ Y \rightarrow (\texttt{length list}\ X) \equiv (\texttt{length list}\ Y)$$

This forms the foundation of verifying the correctness of combinatorial proofs.

Conclusion

- **Abstraction Achievement:** A promising proof system has been developed in Agda, demonstrating the potential for abstraction in formal proof systems.
- **Term Automation:** Once fully implemented, Term automation is expected to significantly reduce the complexity of proofs and improve readability within Agda. This advancement will make the proof-writing process more accessible and efficient.
- **Unfinished Work:** Despite the progress made, there are still pending tasks, including the completion of the FinSet multiplication proof and the `inject-` lemma. Addressing these will be essential for the overall integrity and functionality of the proof system.

Future Studies

- **Additional Operations to Implement:** Define more combinatorial operators, such as Σ, Π, $\_!$, P, and C.
- **Automatic Proof Generation Using Data Term:** Develop a data type that logs the structure of operators to enable automatic transformation of proofs.
- **Term Reasoning:** Create an environment in Agda that facilitates writing more complex combinatorial proofs.

For more details, refer to the project on GitHub: https://github.com/yych0303/2024-IIS-summer-intern/tree/main.

References

[1] 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.

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

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

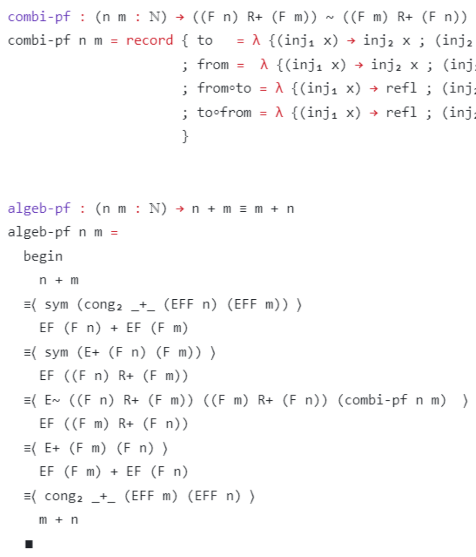
[4] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. arXiv preprint arXiv:1308.0729, 2013.

Examples of Transforming Proofs The following examples utilize the `Embedding` structure, specifically focusing on the preservation of operations and equivalence relations through `E+`, `E*`, `E∼`, and `EFF`. These transformations are used to demonstrate the commutativity of addition $n + m \equiv m + n$ and the associativity of multiplication $n \times (m \times l) \equiv (n \times m) \times l$ by analyzing the operations within the embedded structure.

Here, `EFF` is defined as:

$$\texttt{EFF} : \forall (n : \mathbb{N}) \to \texttt{EF}\ (\texttt{F}\ n) \equiv n$$

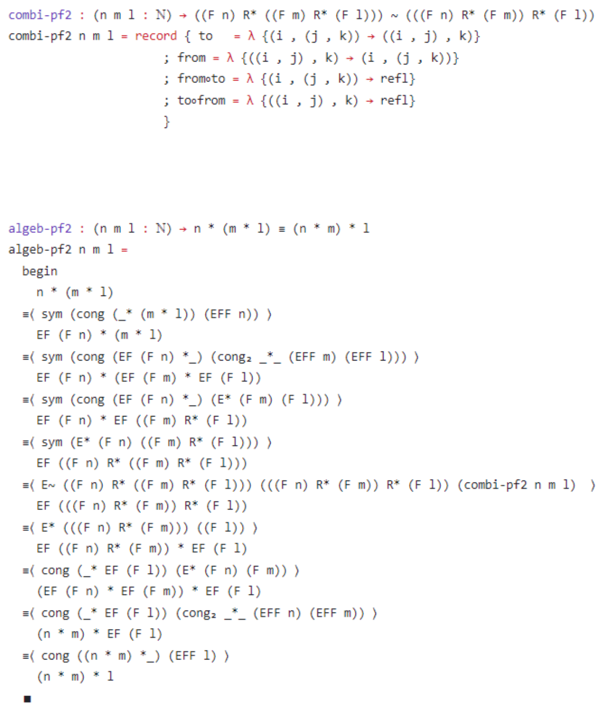
where `F` is a function that maps n to a `FinSet` of size n.



```
combi-pf : (n m : ℕ) → ((F n) R+ (F m)) ~ ((F m) R+ (F n))
combi-pf n m = record { to   = λ {(inj₁ x) → inj₂ x ; (inj₂ y) → inj₁ y}
                      ; from =  λ {(inj₁ x) → inj₂ x ; (inj₂ y) → inj₁ y}
                      ; from∘to = λ {(inj₁ x) → refl ; (inj₂ y) → refl}
                      ; to∘from = λ {(inj₁ x) → refl ; (inj₂ y) → refl}
                      }


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
  ∎
```

(a) Commutativity of Addition



```
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 { to   = λ {(i , (j , k)) → ((i , j) , k)}
                         ; from = λ {((i , j) , k) → (i , (j , k))}
                         ; from∘to = λ {(i , (j , k)) → refl}
                         ; to∘from = λ {((i , j) , k) → refl}
                         }


algeb-pf2 : (n m l : ℕ) → n * (m * l) ≡ (n * m) * l
algeb-pf2 n m l =
  begin
    n * (m * l)
  ≡⟨ sym (cong (_* (m * l)) (EFF n)) ⟩
    EF (F n) * (m * l)
  ≡⟨ sym (cong (EF (F n) *_) (cong₂ _*_ (EFF m) (EFF l))) ⟩
    EF (F n) * (EF (F m) * EF (F l))
  ≡⟨ sym (cong (EF (F n) *_) (E* (F m) (F l))) ⟩
    EF (F n) * EF ((F m) R* (F l))
  ≡⟨ sym (E* (F n) ((F m) R* (F l))) ⟩
    EF ((F n) R* ((F m) R* (F l)))
  ≡⟨ E~ ((F n) R* ((F m) R* (F l))) (((F n) R* (F m)) R* (F l)) (combi-pf2 n m l)  ⟩
    EF (((F n) R* (F m)) R* (F l))
  ≡⟨ E* (((F n) R* (F m))) ((F l)) ⟩
    EF ((F n) R* (F m)) * EF (F l)
  ≡⟨ cong (_* EF (F l)) (E* (F n) (F m)) ⟩
    (EF (F n) * EF (F m)) * EF (F l)
  ≡⟨ cong (_* EF (F l)) (cong₂ _*_ (EFF n) (EFF m)) ⟩
    (n * m) * EF (F l)
  ≡⟨ cong ((n * m) *_) (EFF l) ⟩
    (n * m) * l
  ∎
```

(b) Associativity of Multiplication

Figure 1: Proof by framework without automation