Proof Engineering in Isabelle/HOL: From Combinatorics to Rely-Guarantee

FMOz 2026

Chelsea Edmonds
chelsea.edmonds@uwa.edu.au | https://cledmonds.github.io
04/06/2026

University of Western Australia

Overview

1. What I mean by proof engineering...
2. A combinatorics case study:
   - A locale-centric hierarchy for combinatorial structures
   - Formalising techniques vs theorems
3. Rely-Guarantee is Coinductive
   - A new coinductive definition for Rely-Guarantee
   - Proof engineering soundness proofs
   - Joint Work with Andrei Popescu, Jamie Wright, and John Derrick

Why Proof Engineering

What do I mean by Proof Engineering...

- Formalisation using proof assistants (Isabelle/HOL, Rocq, Lean, etc) is often primarily viewed as a way to guarantee correctness - but is this our only goal?
- In a ITP context, "Proof engineering" has been used to mean several similar but different things ...
- I'm referring to the process of not just formalising a proof, but carefully *planning*, *designing*, *implementing*, and *maintaining* a formal proof (library)
- Creating formal proof frameworks (e.g. definitions, tactics, proof patterns)
- Careful examination and refactoring of proofs for constant improvement
- Originally referred to in very large formalisation projects
- But it can still be crucial at a much smaller scale.

Why do we Formalise?

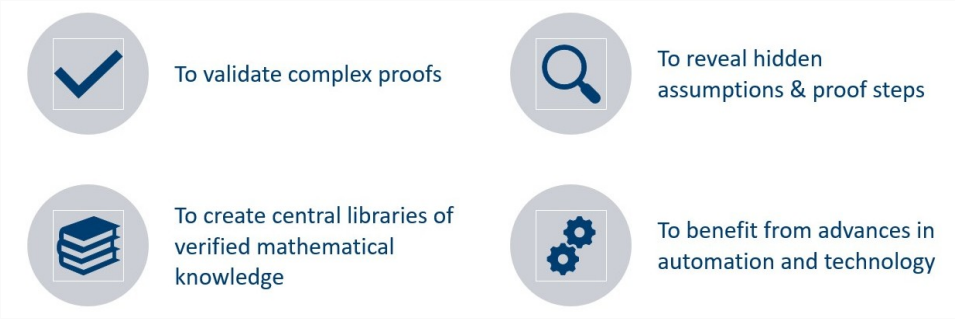


Figure 1: Some typical answers for "Why formalise Mathematics"

Formalisation is not just about verifying a proof!

It can also identify hidden assumptions, new proof approaches, gaps in proofs, and other new insights

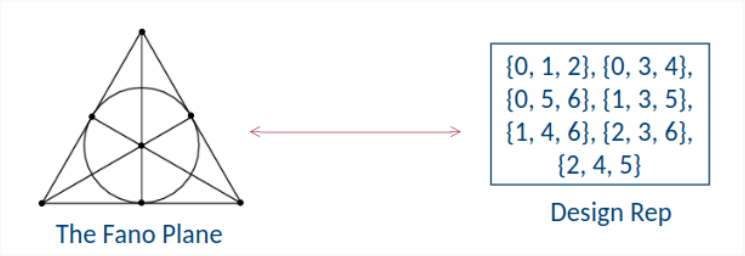
What's the end goal?

- Modular, reusable, and extensible easy to maintain proof libraries.
- Libraries that are easy to use - both to develop new proofs and understand existing ones.
- The development of interesting new (formal) proof techniques and approaches. This could lead to, e.g.:
  - Development of entirely new or more general proofs (or proof techniques)
  - More efficient/concise/intuitive proofs.
  - Deeper understanding of concepts thought to be already "known"

A Combinatorics Case Study

Formalising Combinatorial Structures

- The initial task: formalise combinatorial design theory
- The proof engineering challenges:
  - Creating an extensible, reusable library and easy to use library
  - Manage equivalent structures with different languages/representations (e.g. hypergraphs)
  - Building frameworks for techniques not just verifying theorems

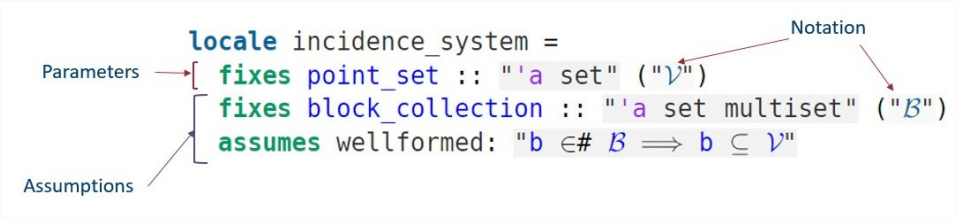


Introducing the Locale Centric Approach

Locales are Isabelle's *module system*. From a logical perspective, locales = persistent contexts:

$$\bigwedge x_1 \ldots x_n . [\![A_1; \ldots; A_m]\!] \Longrightarrow C$$

A simple example for combinatorics:



Locales enable us to establish direct inheritance, indirect inheritance, and specific interpretations (instances) of a locale context.

Introducing the Locale Centric Approach

The *Locale-centric approach* in Isabelle models mathematical structures:

- Using locales to model different structures
- A “little theories” approach to locale definitions - avoid duplication and redundant assumptions
- Definitions specific to a structure should be contained in the locale
- Leverage locale inheritance to transfer results between structure definitions

The Naive Approach

We want to prove a theorem on a Balanced Incomplete Block Design (BIBD), so need a definition for a BIBD!

```
locale bibd = proper_design +
  fixes u_block_size :: nat ("k") and index :: nat ("Λ2")
  and rep_number :: nat ("r")
  assumes uniform [simp]: "bl ∈# ℬ ⟹ card bl = k"
  assumes incomplete: "k < v"
  assumes balanced [simp]: "ps ⊆ 𝒱 ⟹ card ps = 2 ⟹ ℬ index ps = Λ2"
  assumes rep_number [simp]: "x ∈ 𝒱 ⟹  ℬ rep x = r"
```

This definition introduces redundancy, and is not easily extensible

The modular approach!

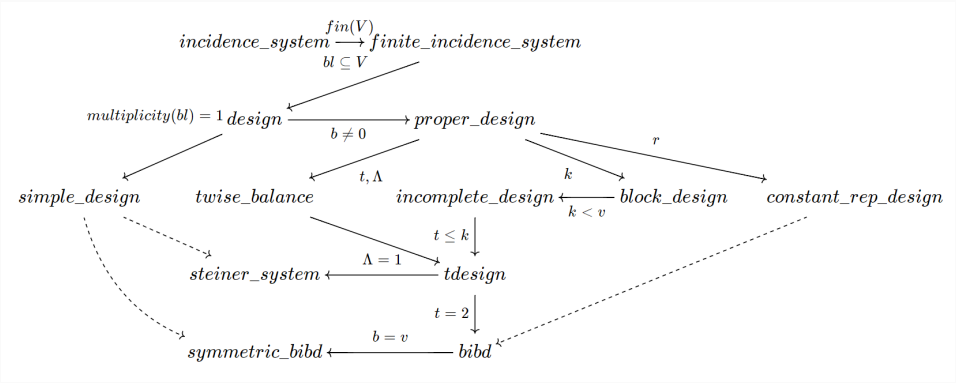


Figure 2: A Block Design Hierarchy

Modular Combinatorial Structure Libraries

Using reverse and mutual sublocales (indirect inheritance) we can establish an equivalence between different definitions.

- For *Hypergraphs* we use language like vertices, edges, adjacent, degree...
- For *Designs* we use language like points, blocks, replication, index ...

The equivalence means theorems automatically translate between the two!

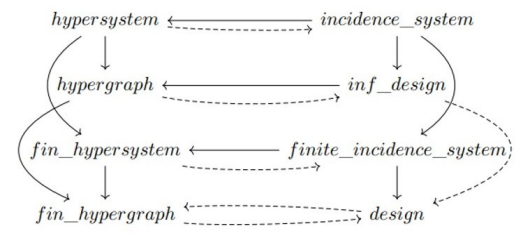


Figure 3: Reverse sublocales: Designs and Hypergraphs

Modular Combinatorial Structure Libraries

We can do something similar for different representations of graphs:

- We can use a set-based definition of graph edges
- And an equivalent relation-based definition of a graph

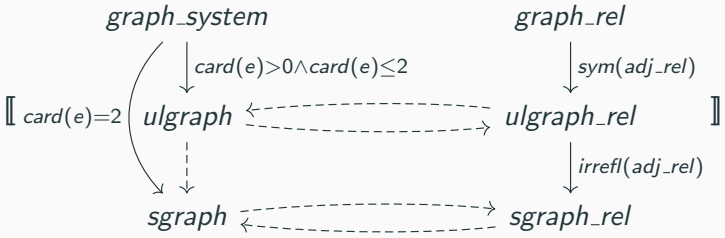
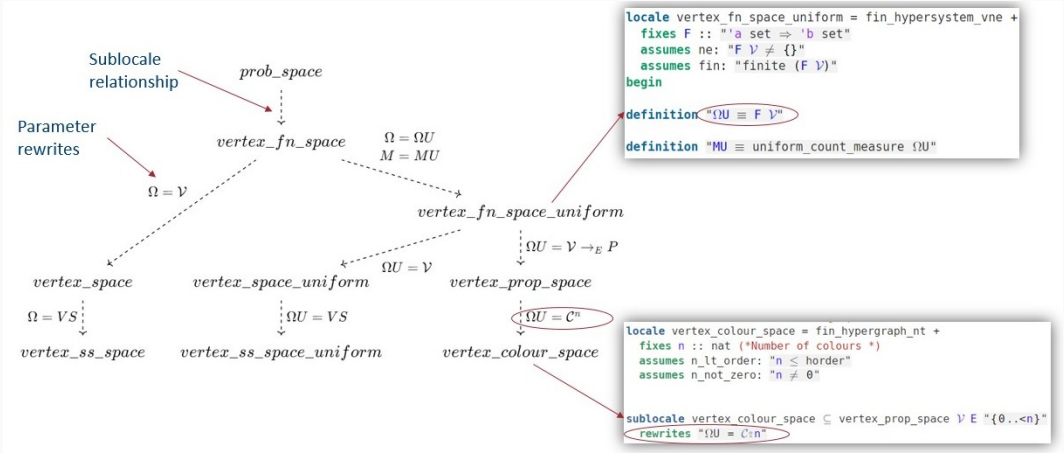


Figure 4: Mutual sublocales: set based and relation based graphs

Easy to build Proof Technique Frameworks

This is a framework for introducing a probability space on combinatorial structures.



Case Study 2: Rely-Guarantee is Coinductive

Proof Engineering for Program Logics

- In mathematics, you often hear people talk about how nice/elegant a proof is.
- Formal proofs of PL concepts can tend to be more repetitive, and have more "grunt work" involved.
- The more straightforward these proofs are, the better.
- In this case we asked - is **coinduction** a more natural, yet equivalent Rely-Guarantee foundation? And can this simplify our mechanised proofs?

4-ways of thinking about Rely-Guarantee

Our work investigated 3 existing frameworks, and introduces 1 new framework:

- Trace-based Semantics (Xu et al, 1997, formalised by Nieto 2003)
- Reachability-based semantics (Coleman-Jones, 1997)
- Inductive-Safety (Vafeiadis & Parkinson, 2007)
- **New: Coinductive-Safety!**

Inductive safety: $\mathsf{safe}_{(R,G,Q)}\ n\ (c, s)$

$$\frac{}{\mathsf{safe}_{(R,G,Q)}\ 0\ (c, s)}\ \textsc{Base}$$

---

1. $\forall s'.\ R\ s\ s' \implies \mathsf{safe}_{(R,G,Q)}\ n\ (c, s')$
2. $\mathsf{final}\ (c, s) \implies Q\ s$
3. $\forall c', s'.\ (c, s) \to (c', s') \implies G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (c', s')$

$$\frac{}{\mathsf{safe}_{(R,G,Q)}\ (n+1)\ (c, s)}\ \textsc{Step}$$

Inductive RG satisfaction: $\forall s \in \mathit{State}.P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c, s')$

Coinductive Safety

Coinductive safety: $\mathsf{safeC}_{(R,G,Q)}\,(c, s)$

$$\frac{\begin{array}{ll} 1. & \forall s'.\ R\ s\ s' \implies \mathsf{safeC}_{(R,G,Q)}\,(c, s') \\ 2. & \mathsf{final}\,(c, s) \implies Q\ s \\ 3. & \forall c', s'.\ (c, s) \to (c', s') \implies G\ s\ s' \wedge \mathsf{safeC}_{(R,G,Q)}\,(c', s') \end{array}}{\mathsf{safeC}_{(R,G,Q)}\,(c, s)}\ \text{StepC}$$

The rule directly defines the infinite safety property (Greatest Fixpoint).
No base case, and removal of the step-counter n,
compared to the inductive (Least Fixpoint) counterpart

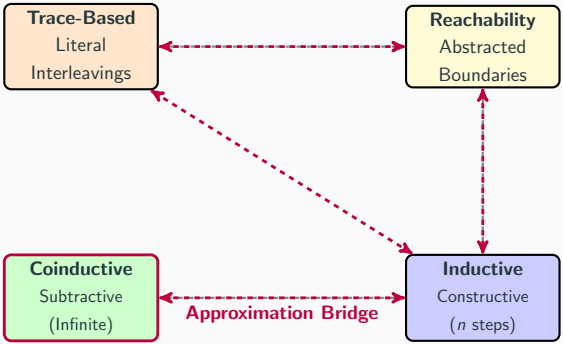
Coinduction from Inductive Approximations

Equivalence Theorem

$$J_F = \bigwedge I_{F^\sharp}(n)$$

- The core of our coinductive definition (J_F) is exactly equal to the limit of the infinite Inductive definition ($\bigwedge I$).
- This result is an immediate consequence of Kleene's theorem under standard continuity assumptions

Equivalence 4-ways



First formal proof of equivalence: Language-independent via Locales

The RG Proof System

An RG Foundation must be sound with respect to these rules

So how difficult are these proofs?

$$\frac{c \vdash (P', R', G', Q') \qquad P \le P' \qquad R \le R' \qquad G' \le G \qquad Q' \le Q}{c \vdash (P, R, G, Q)}\ \text{(Mono)}$$

$$\frac{\mathsf{stable}\ P\ R \qquad \mathsf{stable}\ Q\ R \qquad \mathsf{Im}\ (\mathsf{Rel}\ (\mathsf{evalA}\ a))\ P \le Q \qquad (\mathsf{lift}_1\ P) \sqcap (\mathsf{Rel}\ (\mathsf{evalA}\ a)) \le G}{a \vdash (P, R, G, Q)}\ \text{(AtomRG)}$$

$$\frac{c_1 \vdash (P, R, G, P') \qquad c_2 \vdash (P', R, G, Q) \qquad \mathsf{refl}\ G}{\mathsf{seq}\ c_1\ c_2 \vdash (P, R, G, Q)}\ \text{(SeqRG)}$$

$$\frac{c_1 \vdash (P \sqcap (\mathsf{evalT}\ t), R, G, Q) \qquad c_2 \vdash (P \sqcap (\neg\ (\mathsf{evalT}\ t)), R, G, Q) \qquad \mathsf{stable}\ P\ R \qquad \mathsf{refl}\ G}{\mathsf{if}\ t\ c_1\ c_2 \vdash (P, R, G, Q)}\ \text{(IfRG)}$$

$$\frac{c \vdash (P \sqcap (\mathsf{evalT}\ t), R, G, P) \qquad P \sqcap (\neg\ (\mathsf{evalT}\ t)) \le Q \qquad \mathsf{stable}\ P\ R \qquad \mathsf{stable}\ Q\ R \qquad \mathsf{refl}\ G}{\mathsf{while}\ t\ c \vdash (P, R, G, Q)}\ \text{(WhileRG)}$$

$$\frac{c_1 \vdash (P_1, R_1, G_1, Q_1) \qquad c_2 \vdash (P_2, R_2, G_2, Q_2) \qquad P \le P_1 \sqcap P_2 \qquad R \sqcup G_2 \le R_1 \qquad R \sqcup G_1 \le R_2 \qquad G_1 \sqcup G_2 \le G \qquad Q_1 \sqcap Q_2 \le Q}{\mathsf{par}\ c_1\ c_2 \vdash (P, R, G, Q)}\ \text{(ParRG)}$$

A Proof-Centered Investigation for Soundness

- Based on our definitions, we know in theory inductive proofs worked, but they were simulating coinduction!
- Now to test our theory: are the proofs actually simpler?
- The goal wasn't simply to "redo" soundness proofs using both foundations and compare them afterwards.
- The goal was to carefully engineer proofs according to the underlying fix-point theory that our equivalence theorem came from.
- According to the theory, the coinductive proofs should be more direct!

A Proof Comparison

```
proposition safe_Seq:
assumes safe1: "safe n  (c1, s) R G P'"
and sat2: "⋀ m s' . P' s' ⟹ safe m (c2, s') R G Q"
and Gid: "(=) ≤ G"
shows "safe n (Seq c1 c2,s) R G Q"
  using safe1
proof (induction n arbitrary: s c1)
  case 0
  then show ?case  by (meson Zero)
next
  case (Suc n)
  show ?case
  proof (rule safe_Suc, intro conjI)
```

```
proposition safeC_Seq:
assumes safe1: "safeC R G P' (c1, s)"
and safe2: "⋀ s . P' s ⟹ safeC R G Q (c2, s)"
and Grefl: "(=) ≤ G"
shows "safeC R G Q (Seq c1 c2, s)"
proof -
  define φ where "φ ≡ λ c s . ∃ c1 . c = Seq c1 c2 ∧ safeC R G P' (c1, s)"
  have "φ (c1 $$ c2) s" unfolding φ_def using safe1 by auto
  then show "safeC R G Q (c1 $$ c2, s)"
  proof(coinduct rule: safeC_coinduct')
    case (safeC c s)
    then obtain c1 where eq: "c = (Seq c1 c2)" and safe1: "safeC R G P' (c1, s)"
    show ?case
    proof (intro allI impI conjI)
```

- Coinductive proof removes base case & use of intro rule
- **BUT**, requires some more manual setup in Isabelle
- Regardless, generally adapting coinductive to inductive proofs in the RG context = adding content overall.
- Whereas adapting inductive to coinductive proofs = deleting content overall.

Scaling Up: Proof Enhancements

- Complex concurrent loops can cause state-space explosions in proofs - e.g. the `While` soundness proof

- **Coinductive Approach (Up-to techniques)**:
  - Acts as a "safety net". Instead of proving safety to the end, prove safety until landing in a state *already known to be safe*.

- **Inductive Counterpart (Index-wise Enhancements)**:
  - We identify that up-to coinduction maps to proving approximation-preservation properties for fixed steps n.

- This led to a more general result and insights on up-to coinductive theory!

Main Takeaways

- Rely-guarantee is naturally coinductive
- This simplifies formalised proofs (with the caveat of less automated Isabelle coinduction support currently)
- The formalisation process itself both motivated the investigation, and was key to developing new insights into RG foundations, and also more general theory!

Conclusions

Concluding thoughts

- Two very different case studies, but both demonstrate the value of a proof-engineered approach.
- Modular, general, and more reusable results.
- Leads to a much deeper understanding of the underlying concepts.
- Which in turn leads to new insights into concepts thought to be well known.
- Questions for thought: how does the increasing interest in AI + formalisation impact this process?

Concluding thoughts

- Key takeaway: thinking about how we approach a formalisation can be as valuable as the resulting correctness guarantees.
- Future work: Total correctness for RG, proof engineering across proof assistants.
- Some relevant links:
  - Isabelle AFP formalisations: `https://www.isa-afp.org/authors/edmonds/`
  - Combinatorial Proof Techniques: `https://dl.acm.org/doi/10.1145/3636501.3636946`
  - RG ESOP Paper: `https://dl.acm.org/doi/10.1007/978-3-032-22720-1_9`
  - RG Artifact: `https://doi.org/10.5281/zenodo.18171813`
  - Covert project website: `https://covert-project.github.io`

Contact:

`chelsea.edmonds@uwa.edu.au` | `https://cledmonds.github.io`