FORMALISING COMBINATORIAL MATHEMATICS: A MODULAR APPROACH

CHELSEA EDMONDS | c.l.edmonds@sheffield.ac.uk

Research Associate | University of Sheffield

UnB Summer Workshop in Mathematics

8th of February 2024

PhD Work supported by supervisor: Prof. Lawrence Paulson

PhD Research funded by a joint Cambridge Australia Scholarship and Cambridge Department of Computer Science Studentship

Presentation supported by the EPSRC through the COVERT Project Grant

PRESENTATION OUTLINE

- An introduction to Isabelle/HOL & Locales
- The Motivating Problem
- A Basic Hierarchy – Combinatorial Design Theory
- Locale Reasoning Patterns
  - Locale Interactions
  - Rewriting
  - Mutual & Reverse Sublocales
- Proofs with Locales
  - Using locale structures in proofs
  - Using locales to structure proofs
- Advantages vs Limitations

FORMALISING MATHEMATICS

WHAT IS FORMALISED MATHEMATICS?

- Formal Proofs that are machine checked by an underlying core axiomatic foundation.
- There are many different “proof assistants” that do this kind of work: Isabelle/HOL, HOL Light, Lean, Coq etc.

```
theorem  assumes "prime p"  shows "sqrt p ∉ ℚ"
proof
  from <prime p> have p: "1 < p" by (simp add: prime_def)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
      n: "n ≠ 0" and sqrt_rat: "¦sqrt p¦ = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from n and sqrt_rat have "m = ¦sqrt p¦ * n" by simp
    then show "m² = p * n²"
      by (metis abs_of_nat of_nat_eq_iff of_nat_mult power2_eq_square real_sqrt_abs2 rea
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with <prime p> show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by (auto simp add: power2_eq_square ac_simps)
    with <prime p> show "p dvd n"
      by (metis dvd_triv_left nat_mult_dvd_cancel1 power2_eq_square prime_dvd_power_nat
  qed
  then have "p dvd gcd m n" by simp
  with <coprime m n> have "p = 1" by simp
  with p show False by simp
qed
```

sledgehammer proofs

```
let IRRATIONAL_SQRT_NONSQUARE = prove
 (`!n. rational(sqrt(&n)) ==> ?m. n = m EXP 2`,
  REWRITE_TAC[rational] THEN REPEAT STRIP_TAC THEN
  FIRST_ASSUM(MP_TAC o AP_TERM `\x:real. x pow 2`) THEN
  SIMP_TAC[SQRT_POW_2; REAL_POS] THEN
  ONCE_REWRITE_TAC[GSYM REAL_POW2_ABS] THEN
  REPEAT(FIRST_X_ASSUM(STRIP_ASSUME_TAC o GEN_REWRITE_RULE I [integer])) THEN
  ASM_REWRITE_TAC[REAL_ABS_DIV] THEN DISCH_THEN(MP_TAC o MATCH_MP(REAL_FIELD
   `p = (n / m) pow 2 ==> ~(m = &0) ==> m pow 2 * p = n pow 2`)) THEN
  ANTS_TAC THENL [ASM_MESON_TAC[REAL_ABS_ZERO]; ALL_TAC] THEN
  REWRITE_TAC[REAL_OF_NUM_POW; REAL_OF_NUM_MUL; REAL_OF_NUM_EQ] THEN
  ASM_MESON_TAC[EXP_MULT_EXISTS; REAL_ABS_ZERO; REAL_OF_NUM_EQ]);;

let IRRATIONAL_SQRT_PRIME = prove
 (`!p. prime p ==> ~rational(sqrt(&p))`,
  GEN_TAC THEN ONCE_REWRITE_TAC[GSYM CONTRAPOS_THM] THEN REWRITE_TAC[] THEN
  DISCH_THEN(CHOOSE_THEN SUBST1_TAC o MATCH_MP IRRATIONAL_SQRT_NONSQUARE) THEN
  REWRITE_TAC[PRIME_EXP; ARITH_EQ]);;
```

Isabelle vs HOL Light: Proof of Irrationality

WHY FORMALISE?

ANNALS OF MATHEMATICS
Princeton University & Institute for Advanced Study

About Editorial Board Submission Guidelines Subscriptions Contact

Quasi-projectivity of moduli spaces of polarized varieties

Pages 597-639 from Volume 159 (2004), Issue 2 *by Georg Schumacher, Hajime Tsuji*

Abstract

By means of analytic methods the quasi-projectivity of the moduli space of algebraically polarized varieties with a not necessarily reduced complex structure is proven including the case of nonuniruled polarized varieties.

ANNALS OF MATHEMATICS
Princeton University & Institute for Advanced Study

About Editorial Board Submission Guidelines Subscriptions Contact

Non-quasi-projective moduli spaces

Pages 1077-1096 from Volume 164 (2006), Issue 3 *by János Kollár*

Abstract

We show that every smooth toric variety (and many other algebraic spaces as well) can be realized as a moduli space for smooth, projective, polarized varieties. Some of these are not quasi-projective. This contradicts a recent paper (Quasi-projectivity of moduli spaces of polarized varieties, *Ann. of Math.* **159** (2004) 597–639.).

[1] The result of Problem 11 contradicts the results announced by Levy [1963b]. Unfortunately, the construction presented there cannot be completed.
[2] The transfer to ZF was also claimed by Marek [1966] but the outlined method appears to be unsatisfactory and has not been published.
[3] A contradicting result was announced and later withdrawn by Truss [1970].
[4] The example in Problem 22 is a counterexample to another condition of Mostowski, who conjectured its sufficiency and singled out this example as a test case.
[5] The independence result contradicts the claim of Felgner [1969] that the Cofinality Principle implies the Axiom of Choice. An error has been found by Morris (see Felgner's corrections to [1969]).

*Footnotes on page 118 of Jech's *The Axiom of Choice (1973)*

WHY FORMALISE?

To validate complex proofs

To reveal hidden assumptions, proof steps, and mathematical insights

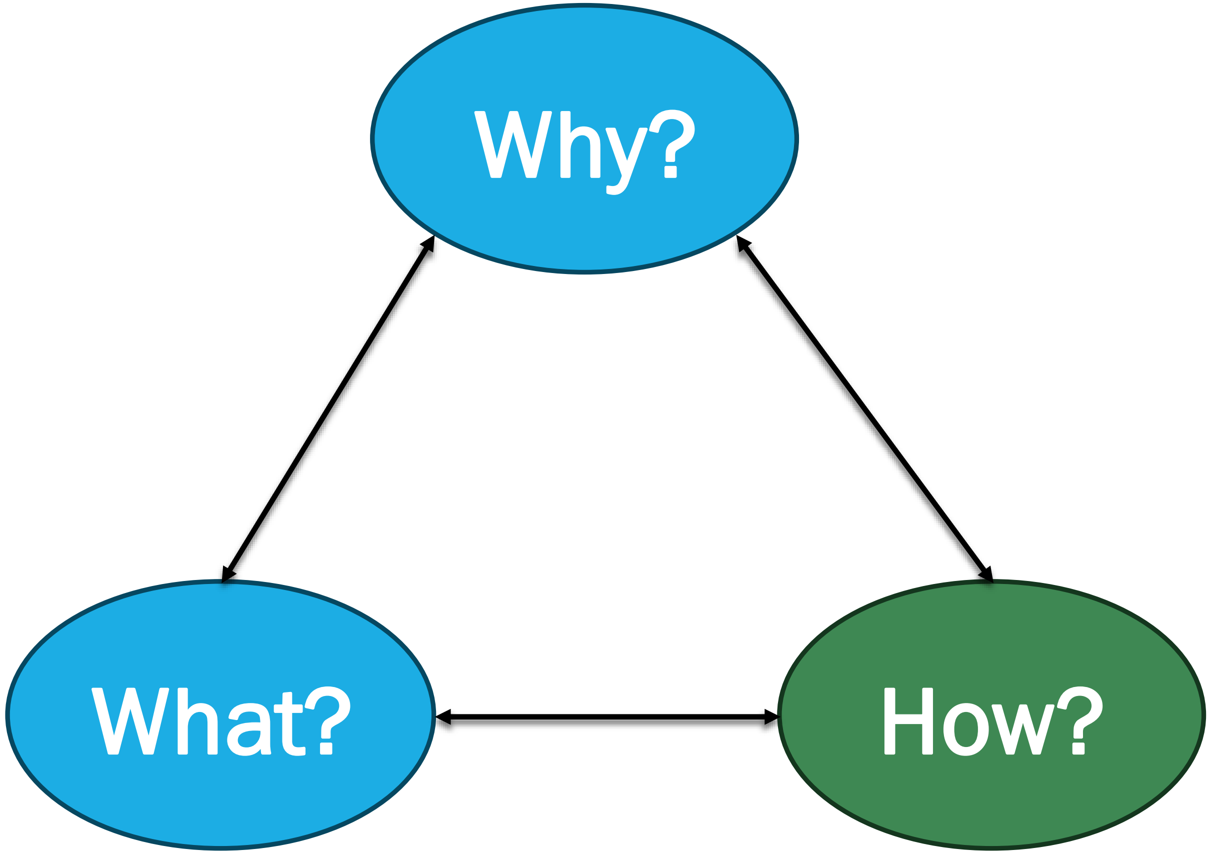
To create central libraries of verified mathematical knowledge

To benefit from advances in automation and technology

Ultimate Goal: Augment Human Intelligence

FORMALISATION CHALLENGES

- Very quickly **growing** libraries
- Lots of **duplication**
- **Theorem specific** libraries
- **Limited reusability** of many results
- Limited ability to **naturally** use mathematical techniques
- Need for **general** techniques & **modular & extensible** libraries



INTRODUCTION TO ISABELLE & LOCALES

ISABELLE/HOL

- Simple type theory
- Sledgehammar – automated proof search.
- Search tools: Query Search, Find Facts, SErAPIS
- The Isar structured proof language
- Interactive Development Environment
- Extensive existing libraries in Maths & Computer Science
- Additional features: Code generation, modularity, polymorphism, documentation generation …

```
theorem  assumes "prime p"  shows "sqrt p ∉ ℚ"
proof
  from ‹prime p› have p: "1 < p" by (simp add: prime_def)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat where
      n: "n ≠ 0" and sqrt_rat: "¦sqrt p¦ = m / n"
    and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from n and sqrt_rat have "m = ¦sqrt p¦ * n" by simp
    then show "m² = p * n²"
      by (metis abs_of_nat of_nat_eq_iff of_nat_mult power2_eq_square real_sqrt_abs2 rea
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with ‹prime p› show "p dvd m" by (rule prime_dvd_power_nat)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by (auto simp add: power2_eq_square ac_simps)
    with ‹prime p› show "p dvd n"
      by (metis dvd_triv_left nat_mult_dvd_cancel1 power2_eq_square prime_dvd_power_nat
  qed
  then have "p dvd gcd m n" by simp
  with ‹coprime m n› have "p = 1" by simp
  with p show False by simp
qed
```

sledgehammer proofs

LOCALE BASICS

- Locales are Isabelle's module system. From a logical perspective, they are simply persistent contexts.

$$\bigwedge x_1 \dots x_n.\, [\![A_1; \dots; A_m]\!] \Rightarrow C.$$

- A simple example (taken from the Locales tutorial):

Notation

Parameters

```
locale partial_order =
  fixes le :: "'a ⇒ 'a ⇒ bool" (infixl "⊑" 50)
  assumes refl [intro, simp]: "x ⊑ x"
    and anti_sym [intro]: "⟦ x ⊑ y; y ⊑ x ⟧ ⟹ x = y"
    and trans [trans]: "⟦ x ⊑ y; y ⊑ z ⟧ ⟹ x ⊑ z"
```

Assumptions

LOCALE BASICS – INHERITANCE & INTERPRETATIONS

- We have direct inheritance

```
locale lattice = partial_order +
  assumes ex_inf: "∃inf. is_inf x y inf"
    and ex_sup: "∃sup. is_sup x y sup"
begin
```

- And indirect inheritance

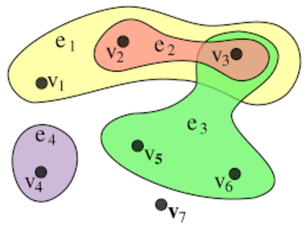
```
sublocale total_order ⊆ lattice
```

- Interpretations (global & local)

```
interpretation int: partial_order "(≤) :: [int, int] ⇒ bool"
  rewrites "int.less x y = (x < y)"
proof -
```

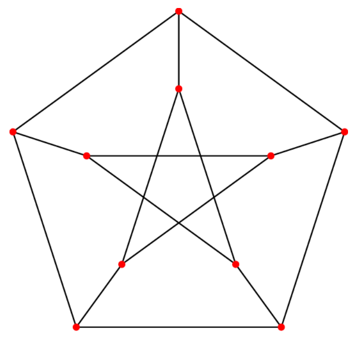
THE MOTIVATING PROBLEM

MOTIVATING PROBLEM – LARGE HIERARCHIES



- K-uniform hypergraphs
- Regular hypergraphs
- Regular k-uniform graphs
- Non-trivial hypergraphs

- Complete Graphs
- Graph Decompositions (Structure)
- Regular graphs
- Cyclic graphs



Hypergraphs

Graphs

Combinatorial Structures

Designs

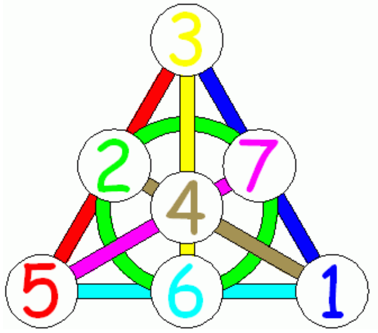
Geometric

+ Many more…

Block 1	(C1, D1)	(C2, D2)	(C3, D3)
Block 2	(C1, D2)	(C2, D3)	(C3, D1)
Block 3	(C1, D3)	(C2, D1)	(C3, D2)
Block 4	(C1, D1)	(C2, D3)	(C3, D2)
Block 5	(C1, D2)	(C2, D1)	(C3, D3)
Block 6	(C1, D3)	(C2, D2)	(C3, D1)

- Block designs
- Balanced Designs
- Group Divisible Designs
- Incomplete designs
- Steiner Systems
- Latin Squares

- Projective planes



THE CHALLENGES

Problem 1:

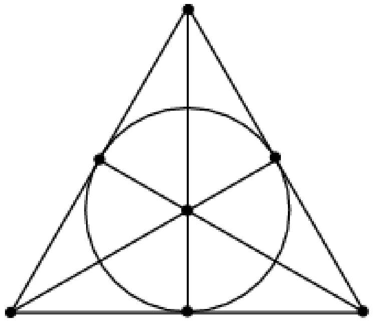
Many variations and definitions (inconsistent)

Problem 2:

Complex inheritance patterns

Problem 3:

Different language… equivalent structures?



The Fano Plane

{0, 1, 2}, {0, 3, 4}, {0, 5, 6}, {1, 3, 5}, {1, 4, 6}, {2, 3, 6}, {2, 4, 5}

Design Rep

FIRST ATTEMPTS…

Approach 1: Typeclasses?

```
class incidence_system_class =
  fixes D :: "'a design"
  assumes wellformed: "b ∈# blocks D ⟹ b ⊆ points D"

record 'a block_design = "'a design" +
  size :: "nat"

record 'a balanced_design = "'a design" +
  balance :: "nat"
  t :: nat

record bibd = "'a block_design" + "'a balanced_design"
(* X Can't combine records *)

class block_design = incidence_system_class +
  fixes k :: "nat"
(* X Can't add new type to class *)
```

Approach 2: Records + Locales?

```
record 'a design =
  points :: "'a set "
  blocks :: "'a set multiset"

locale incidence_system =
  fixes D :: "'a design" (structure)
  assumes wf: "b ∈# blocks D ⟹ b ⊆ points D"
```

Messier notation, less automation.

THE LOCALE-CENTRIC APPROACH

"The software engineering approach to formalising mathematics!"

- Use only locales to model different structures (no complex types/records etc)
- Use *local* definitions inside locale contexts
- Type-synonyms can be used with care to bundle objects
- The "Little Theories" approach for locale definitions
- Avoid duplication at all costs!
- First Introduced by Ballarin in a paper on "Formalising an Abstract Algebra Textbook" (2020)

```
record 'a design =
  points :: "'a set "
  blocks :: "'a set multiset"

locale incidence_system =
  fixes D :: "'a design" (structure)
  assumes wf: "b ∈# blocks D ⟹ b ⊆ points D"
```

```
locale incidence_system =
  fixes point_set :: "'a set" ("𝒱")
  fixes block_collection :: "'a set multiset" ("ℬ")
  assumes wellformed: "b ∈# ℬ ⟹ b ⊆ 𝒱"
begin
locale design = finite_incidence_system +
  assumes blocks_nempty: "bl ∈# ℬ ⟹ bl ≠ {}"
begin
```

A BASIC HIERARCHY

COMBINATORIAL DESIGN THEORY

INTRO TO COMBINATORIAL DESIGNS

- A design is a finite set of points *V* and a collection of subsets of *V*, called blocks *B.*
- Applications range from experimental and algorithm design, to security and communications.
- What makes a design interesting? Properties:
  - The set of block sizes *K*
  - The set of replication numbers *R*
  - The set of t-indices Λ_t
  - The set of intersection numbers *M*
- Language varies: designs, hypergraphs, matrices, geometries, graph decompositions, codes …

Combinatorial Designs/Hypergraphs had not previously been formalised

THE BASIC DEFINITIONS

```
locale incidence_system =
  fixes point_set :: "'a set" ("𝒱")
  fixes block_collection :: "'a set multiset" ("ℬ")
  assumes wellformed: "b ∈# ℬ ⟹ b ⊆ 𝒱"
begin
```

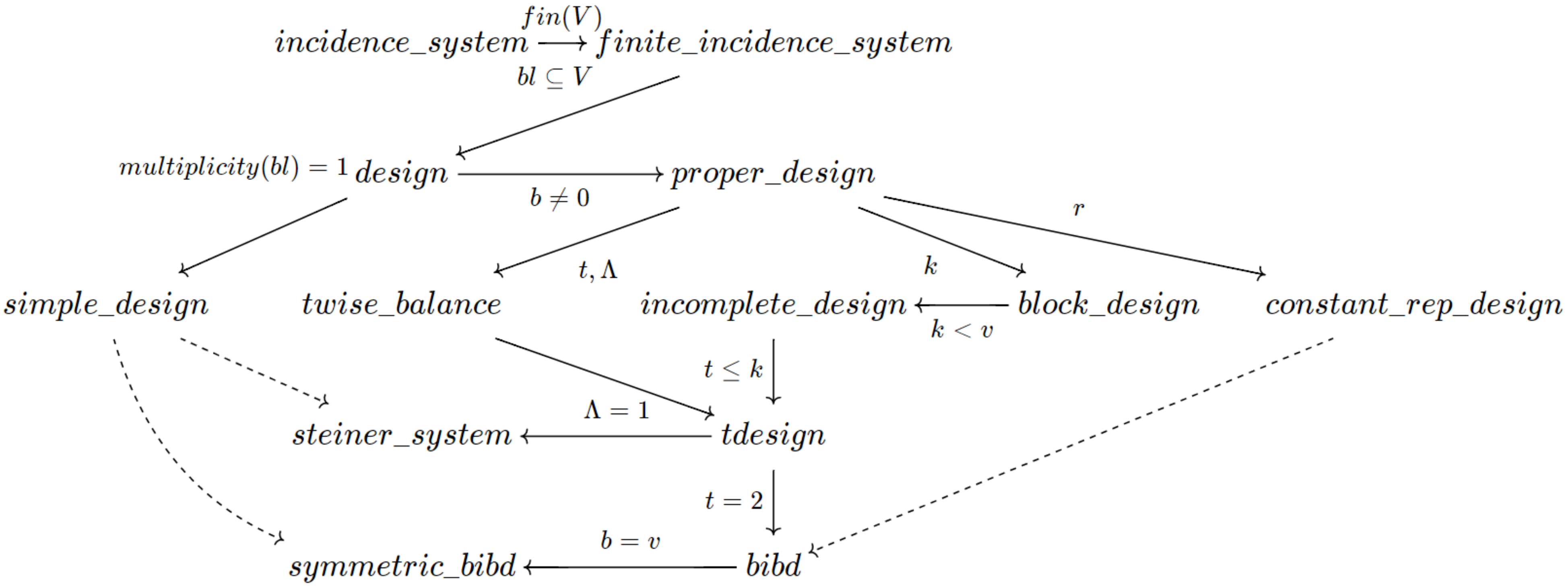
```
locale finite_incidence_system = incidence_system +
  assumes finite_sets: "finite 𝒱"
begin
```

```
locale simple_incidence_system = incidence_system +
  assumes simple [simp]: "bl ∈# ℬ ⟹ multiplicity bl = 1"
```

```
locale design = finite_incidence_system +
  assumes blocks_nempty: "bl ∈# ℬ ⟹ bl ≠ {}"
begin
```

```
locale simple_design = design + simple_incidence_system
```

THE HIERARCHY

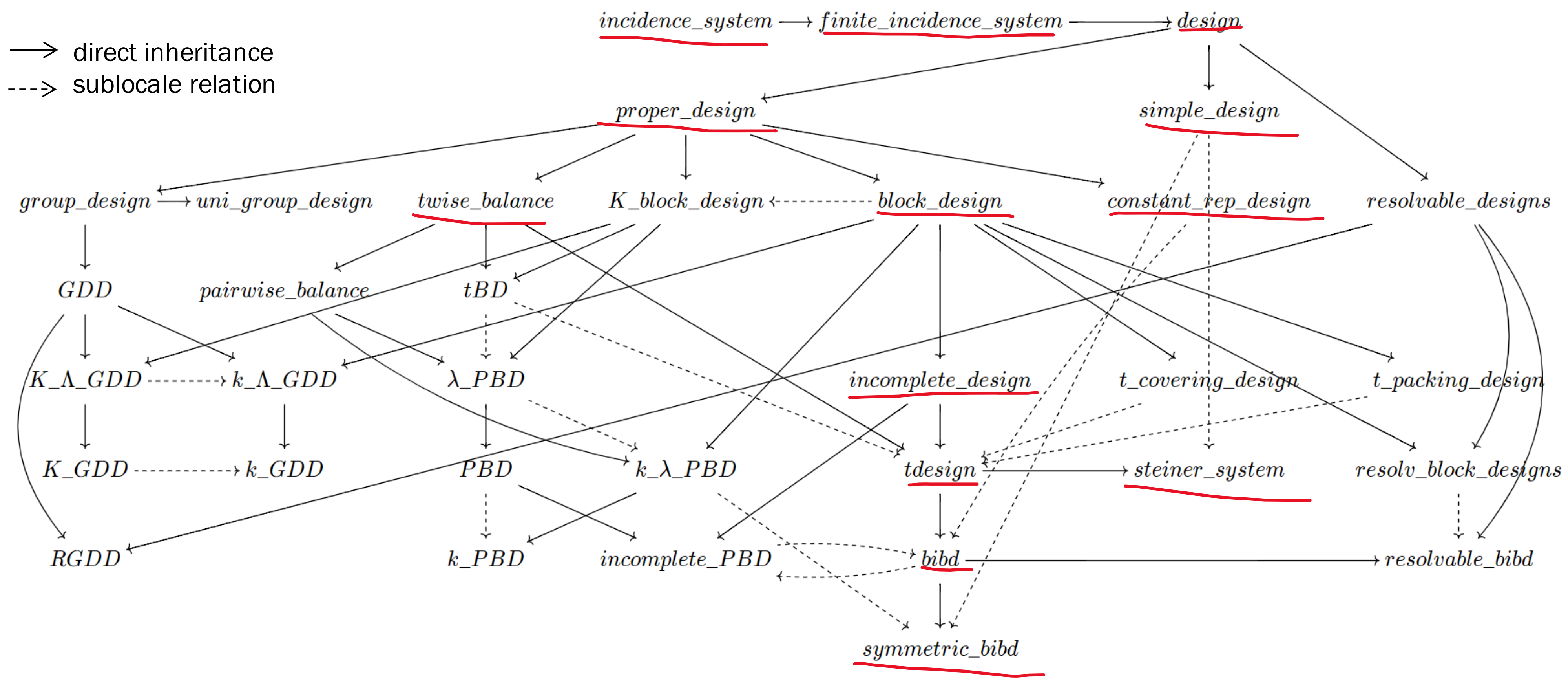



```
locale t_design = incomplete_design + t_wise_balance +
  assumes block_size_t: "t ≤ k"
```

```
locale bibd = t_design 𝒱 ℬ k 2 Λ
  for point_set ("𝒱") and block_collection ("ℬ")
   and u_block_size ("k") and index ("Λ")
```

EXTENDING THE HIERARCHY

Other Design Classes: Group Divisible Designs (GDDs), Pairwise Balanced Designs (PBDs), design isomorphisms
Connections with Graph Theory (Noschinski, 2015)



ANOTHER HIERARCHY – GRAPH THEORY

```
locale graph_system =
  fixes vertices :: "'a set" ("V")
  fixes edges :: "'a edge set" ("E")
  assumes wellformed: "e ∈ E ⟹ e ⊆ V"
```

```
locale ulgraph = graph_system +
  assumes edge_size: "e ∈ E ⟹
     card e > 0 ∧ card e ≤ 2"
```

```
locale sgraph = graph_system +
  assumes two_edges: "e ∈ E ⟹ card e = 2"
```

```
locale bipartite_graph = graph_system +
  fixes X Y :: "'a set"
  assumes partition: "partition_on V {X, Y}"
  assumes ne: "X ≠ Y"
  assumes edge_betw: "e ∈ E ⟹ e ∈ all_bi_edges X Y"
```

Indirect Inheritance via sublocales

```
sublocale bipartite_graph ⊆ sgraph
  using card_edges_two by (unfold_locales)
```

Further extensions done for finite and non-empty properties, as well as connectivity, subgraphs, triangle-free graphs etc. See Archive of Formal Proofs.

WHY GRAPHS AGAIN?

```
type_synonym uvert = nat
type_synonym uedge = "nat set"
type_synonym ugraph = "uvert set × uedge set"
```

Basic Undirected Graphs
(Noschinski)

```
record ('v,'w) graph =
  nodes :: "'v set"
  edges :: "('v × 'w × 'v) set"
```

Graphs "For Purpose"
(Nordhoff & Lammich)

```
record ('a,'b) pre_digraph =
  verts :: "'a set"
  arcs :: "'b set"
  tail :: "'b ⇒ 'a"
  head :: "'b ⇒ 'a"
```

General Digraphs
(Noschinski)

- Existing libraries had notable limitations or were built for purpose
- Notably there was no general library for *undirected graphs* (and digraphs introduce unnecessary complication to formal reasoning)

AND ANOTHER HIERARCHY….? - HYPERGRAPHS

- Realistically, this is just designs… with another language – so we use direct inheritance!

```
locale hypersystem =
  fixes vertices :: "'a set" ("𝒱")
  fixes edges :: "'a hyp_edge multiset" ("E")
  assumes wellformed: "e ∈# E ⟹ e ⊆ 𝒱"
begin
```

```
locale fin_hypersystem = hypersystem + finite_incidence_system 𝒱 E
```

```
locale fin_hypergraph = hypergraph + fin_hypersystem
```

```
locale hypergraph = hypersystem + inf_design 𝒱 E
```

```
locale kuniform_hypergraph = hypergraph +
  fixes k :: nat
  assumes uniform: "⋀ e . e ∈# E ⟹ card e = k"
```

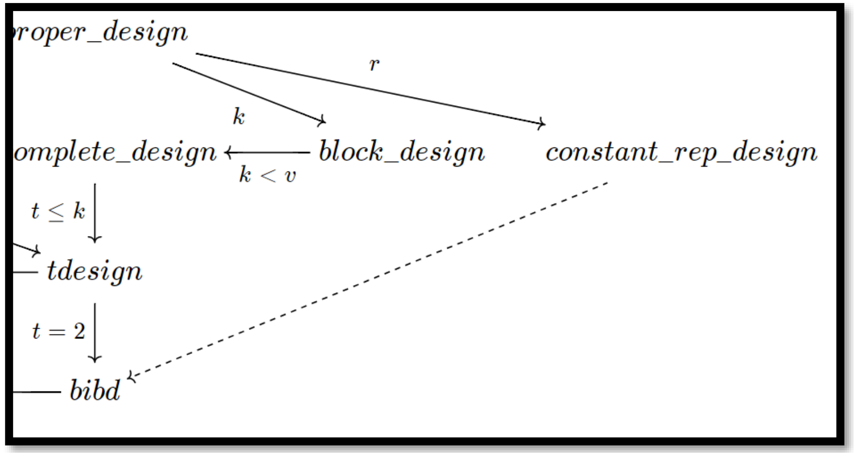
```
locale dregular_hypergraph = hypergraph +
  fixes d
  assumes "⋀ x. x ∈ 𝒱 ⟹ degree x = d"
```

LOCALE REASONING PATTERNS

MODELLING INTERACTIONS

BASIC PROOF TACTICS

- There are two main tactics (currently) for locales: unfold_locales & intro_locales
- This is a proof of an inheritance property: BIBD has a rep number




```
locale constant_rep_design = proper_design +
  fixes design_rep_number :: int ("r")
  assumes rep_number [simp]: "x ∈ 𝒱 ⟹  ℬ rep x = r"
```

```
sublocale bibd ⊆ constant_rep_design 𝒱 ℬ  "(Λ * (v - 1) div (k - 1))"
  apply unfold_locales
```

```
proof (prove)
goal (1 subgoal):
 1. ⋀x. x ∈ 𝒱 ⟹ ℬ rep x = Λ * (v - 1) div (k - 1)
```

Helper lemma:

```
sublocale bibd ⊆ constant_rep_design 𝒱 ℬ  "(Λ * (v - 1) div (k - 1))"
  using r_constant_2 by (unfold_locales) simp_all
```

BASIC PROOF TACTICS

- There are two main tactics (currently) for locales: unfold_locales & intro_locales
- This is a proof using locale constructions

```
definition complement_blocks :: "'a set multiset" ("(ℬᶜ)")where
"complement_blocks ≡ {# blᶜ . bl ∈# ℬ #}"
```

Transformation definition

Local interpretation

Individual proof goals from unfold_locales

```
lemma complement_bibd:
  assumes "k ≤ v - 2"
  shows "bibd 𝒱 (complement_blocks) (v - k) (b + Λ - 2*r)"
proof -
  interpret des: incomplete_design 𝒱 "(complement_blocks)" "(v - k)"
    using assms complement_incomplete by blast
  show ?thesis proof (unfold_locales, simp_all)
    show "2 ≤ des.v" using assms block_size_t by linarith
    show "⋀ps. ps ⊆ 𝒱 ⟹ card ps = 2 ⟹
      points_index (complement_blocks) ps = b + Λ - 2 * (Λ * (des.v - 1) div (k - 1))"
      using complement_bibd_index by simp
    show "2 ≤ des.v - k" using assms block_size_t by linarith
  qed
qed
```

LOCALE INTERACTIONS – COMBINING LOCALES

```
locale incidence_system_isomorphism = source: incidence_system 𝒱 ℬ + target: incidence_system 𝒱' ℬ'
  for "𝒱" and "ℬ" and "𝒱'" and "ℬ'" + fixes bij_map ("π")
  assumes bij: "bij_betw π 𝒱 𝒱'"
  assumes block_img: "image_mset ((`) π) ℬ = ℬ'"
begin
```

```
lemma design_iso_points_indices_imp:
  assumes "x ∈ source.point_indices t"
  shows "x ∈ target.point_indices t"
proof -
  obtain ps where t: "card ps = t" and ss: "ps ⊆ 𝒱" and x: "ℬ index ps = x" using assms
    by (auto simp add: source.point_indices_def)
  then have x_val: "x = ℬ' index (π ` ps)" using design_iso_points_index_eq by auto
  have x_img: " (π ` ps) ⊆ 𝒱'"
    using ss bij iso_points_map by fastforce
  then have "card (π ` ps) = t" using t ss iso_points_ss_card by auto
  then show ?thesis using target.point_indices_elem_in x_img x_val by blast
qed
```

Source and target references to distinguish objects in proofs

Can also work outside of locale context

```
definition isomorphic_designs (infixl "≅D" 50) where
"𝒟 ≅D 𝒟' ⟷ (∃ π . design_isomorphism (fst 𝒟) (snd 𝒟) (fst 𝒟') (snd 𝒟') π)"
```

SUBLOCALE CHAINS

- Introduced by Ballarin as the “functor pattern”.

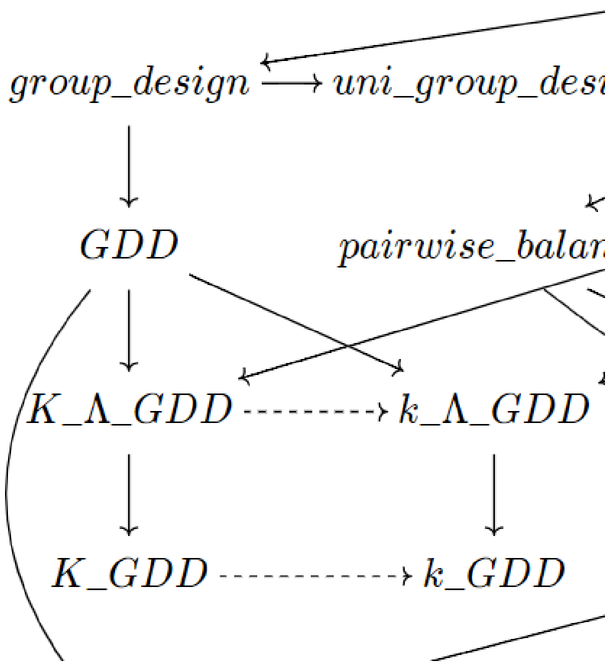
```
locale GDD = group_design +
  fixes index :: int ("Λ")
  assumes index_ge_1: "Λ ≥ 1"
  assumes index_together: "G ∈ 𝒢 ⟹
  assumes index_distinct: "G1 ∈ 𝒢 ⟹
```

```
locale K_Λ_GDD = K_block_design + GDD
```

```
locale k_Λ_GDD = block_design + GDD
```

```
locale K_GDD = K_Λ_GDD 𝒱 ℬ 𝒦 𝒢 1
```

```
locale k_GDD = k_Λ_GDD 𝒱 ℬ k 𝒢 1
```



→ direct inheritance
- - -> sublocale relation

EQUIVALENT STRUCTURES? - REVERSE SUBLOCALES

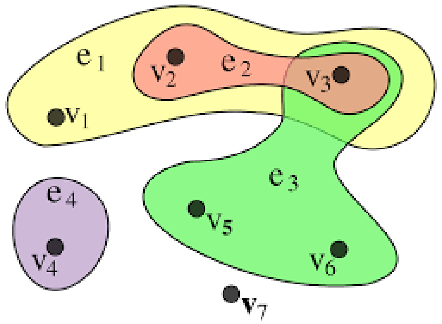
- Reverse sublocales: sublocale in opposite direction of direct inheritance.

```
sublocale fin_hypergraph ⊆ finite_incidence_system 𝒱 E
  rewrites "point_replication_number E v = degree v" and "points_index E vs = degree_set vs"
  by unfold_locales (simp_all add: wellformed finite point_replication_number_def degree_def
      degree_set_def points_index_def)
```

```
locale fin_hypersystem = hypersystem + finite_incidence_system 𝒱 E
```

```
locale hypergraph = hypersystem + inf_design 𝒱 E
```

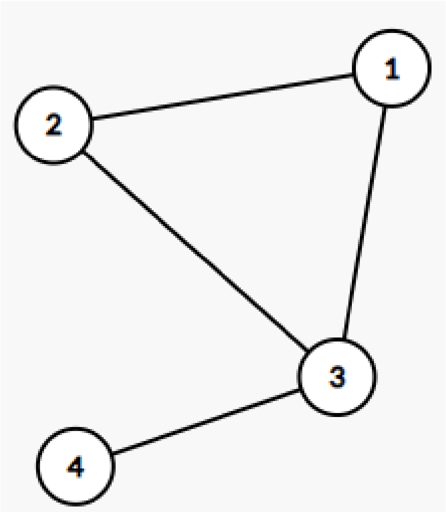
```
sublocale inf_design ⊆ hypergraph 𝒱 ℬ
  by unfold_locales (simp add: wellformed)
```



Block 1	(C1, D1)	(C2, D2)	(C3, D3)
Block 2	(C1, D2)	(C2, D3)	(C3, D1)
Block 3	(C1, D3)	(C2, D1)	(C3, D2)
Block 4	(C1, D1)	(C2, D3)	(C3, D2)
Block 5	(C1, D2)	(C2, D1)	(C3, D3)
Block 6	(C1, D3)	(C2, D2)	(C3, D1)

EQUIVALENT STRUCTURES? - MUTUAL SUBLOCALES

R = {(1, 2), (2, 1), (1, 3), (3, 1), (2, 3), (3, 2), (3, 4), (4, 3)}



E = {{1, 2}, {1, 3}, {2, 3}, {3, 4}}

```
locale graph_rel =
  fixes vertices :: "'a set" ("V")
  fixes adj_rel :: "'a rel"
  assumes wf: "⋀ u v. (u, v) ∈ adj_rel
                 ⟹ u ∈ V ∧ v ∈ V"

 locale ulgraph_rel = graph_rel +
   assumes sym_adj: "sym adj_rel"
```

```
locale ulgraph = graph_system +
  assumes edge_size: "e ∈ E ⟹
     card e > 0 ∧ card e ≤ 2"
```

EQUIVALENT STRUCTURES? - MUTUAL SUBLOCALES

```
text ‹ Temporary interpretation - mutual sublocale setup ›
interpretation ulgraph V edge_set by (rule is_ulgraph)
```

```
interpretation ulgraph_rel V adj_relation by (rule is_ulgraph_rel)
```

+ lemmas on equivalences of definitions ....

```
sublocale ulgraph_rel ⊆ ulgraph "V" "edge_set"
  rewrites "ulgraph.adj_relation edge_set = adj_rel"
  using local.is_ulgraph rel_edges_is by simp_all

sublocale ulgraph ⊆ ulgraph_rel "V" "adj_relation"
  rewrites "ulgraph_rel.edge_set adj_relation = E"
  using is_ulgraph_rel edges_rel_is by simp_all
```

LOCALES IN PROOFS

Locales work for modelling complex hierarchies....

But are they easy to use to formalise mathematical results?

BASICS – PROOFS OF PROPERTIES

- Locales are designed to be a module system – so working inside the context is EASY.

```
lemma necess_cond_1_rhs:
  assumes "x ∈ 𝒱"
  shows "size ({# p ∈# (mset_set (𝒱 - {x}) ×# {# bl ∈# ℬ . x ∈ bl #}). fst p ∈ snd p#}) = Λ * (v- 1)"
```

```
lemma necess_cond_1_lhs:
  assumes "x ∈ 𝒱"
  shows "size ({# p ∈# (mset_set (𝒱 - {x}) ×# {# bl ∈# ℬ . x ∈ bl #}). fst p ∈ snd p#})
       = (ℬ rep x) * (k - 1)"
    (is "size ({# p ∈# (?M ×# ?B). fst p ∈ snd p#}) = (ℬ rep x) * (k - 1) ")
```

```
lemma necessary_condition_one:
  shows "r * (k - 1) = Λ * (v - 1)"
  using necess_cond_1_rhs necess_cond_1_lhs design_points_nempty rep_number by auto
```

Local definitions

Locale parameters

USING SYMMETRIC INSTANCES

```
lemma bipartite_sym: "bipartite_graph V E Y X"
  using partition ne edge_betw all_bi_edges_sym
  by (unfold_locales) (auto simp add: insert_commute)
```

```
lemma edge_size_degree_sumY: "card E = (∑y ∈ Y . degree y)"
proof -
  have "(∑y ∈ Y . degree y) = (∑y ∈ Y . card(neighbors_ss y X))"
    using degree_neighbors_ssY by (simp)
  also have "... = card (all_edges_between X Y)"
    using card_all_edges_betw_neighbor
    by (metis card_all_edges_between_commute partitions_finite(1) partitions_finite(2))
  finally show ?thesis
    by (simp add: card_edges_between_set)
qed

lemma edge_size_degree_sumX: "card E = (∑y ∈ X . degree y)"
proof -
  interpret sym: fin_bipartite_graph V E Y X
    using fin_bipartite_sym by simp
  show ?thesis using sym.edge_size_degree_sumY by simp
qed
```

Interpret for symmetric property. Can be at a local and theory level.

MULTIPLE INSTANCES OF STRUCTURE

```
lemma wilsons_construction_proper:
  assumes "card I = w"
  assumes "w > 0"
  assumes "⋀ n. n ∈ 𝒦' ⟹ n ≥ 2"
  assumes "⋀ B . B ∈# ℬ ⟹ K_GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B }"
  shows "proper_design (𝒱 × I) (∑B ∈# ℬ. (f B))" (is "proper_design ?Y ?B")
proof (unfold_locales, simp_all)
  show "⋀b. ∃x∈#ℬ. b ∈# f x ⟹ b ⊆ 𝒱 × I"
  proof -
    fix b
    assume "∃x∈#ℬ. b ∈# f x"
    then obtain B where "B ∈# ℬ" and "b ∈# (f B)" by auto
    then interpret kgdd: K_GDD "(B × I)" "(f B)" 𝒦' "{{x} × I |x . x ∈ B }" using assms by auto
    show "b ⊆ 𝒱 × I" using kgdd.wellformed
      using ‹B ∈# ℬ› ‹b ∈# f B› wellformed by fastforce
  qed
  show "finite (𝒱 × I)" using finite_sets assms bot_nat_0.not_eq_extremum card.infinite by blast
  show "⋀bl. ∃x∈#ℬ. bl ∈# f x ⟹ bl ≠ {}"
```

Interpret instances from assumption.

This is inside a GDD locale itself!

NOTATION TRICKS – REASONING OUTSIDE OF CONTEXT

```
type_synonym 'a hyp_edge = "'a set"

type_synonym 'a hyp_graph = "('a set) × ('a hyp_edge multiset)"

abbreviation hyp_edges :: "'a hyp_graph ⇒ 'a hyp_edge multiset" where
  "hyp_edges H ≡ snd H"

abbreviation hyp_verts :: "'a hyp_graph ⇒ 'a set" where
  "hyp_verts H ≡ fst H"

locale fin_hypergraph =
  fixes vertices :: "'a set" ("𝒱")
  fixes edges :: "'a hyp_edge multiset" ("E")
  assumes wellformed: "e ∈# E ⟹ e ⊆ 𝒱"
  assumes finite: "finite 𝒱"
```

Bundles hypergraph elements.

Abbreviations for easy access.

```
definition hypergraph_decomposition :: "'a hyp_graph multiset ⇒ bool" where
"hypergraph_decomposition S ≡ (∀ h ∈# S . is_subhypergraph  h) ∧ partition_on_mset E {#hyp_edges h . h ∈# S#}"
```

```
definition is_subhypergraph :: "'a hyp_graph ⇒ bool" where
"is_subhypergraph H ≡ sub_hypergraph (hyp_verts H) (hyp_edges H) 𝒱 E"
```

Within context of fin_hypergraph

APPLYING NOTATION TRICKS – WORKING OUTSIDE A CONTEXT

```
definition not_col_n_uni_hyps:: "nat ⇒ 'a hyp_graph set"
  where "not_col_n_uni_hyps n ≡ { h . fin_kuniform_hypergraph_nt (hyp_verts h) (hyp_edges h) n
    ∧ ¬ (hypergraph.has_property_B (hyp_verts h) (hyp_edges h)) }"
```

Locale & local definition used in theory definition

```
lemma obtains_min_edge_colouring:
  fixes z :: "'a itself"
  assumes "min_edges_colouring n z < x"
  obtains h :: "'a hyp_graph" where "h ∈ not_col_n_uni_hyps n"
    and "enat (size (hyp_edges h)) < x"
proof -
  have "(INF h ∈ ((not_col_n_uni_hyps n) :: 'a hyp_graph set) .
    enat (size (hyp_edges h))) < x"
    using min_edges_colouring_def[of "n" z] assms by auto
  thus ?thesis using
      enat_lt_INF[of "λ h. enat (size (hyp_edges h))" "not_col_n_uni_hyps n" "x"]
    using that by blast
qed
```

Locale “abstracted” away

PROBABILISTIC PROOFS: COMBINING LOCALES ACROSS DISCIPLINES

```
locale dependency_graph = sgraph "V :: 'a set set" E + prob_space "M :: 'a measure" for V E M +
  assumes vin_events: "V ⊆ events"
  assumes mis: "⋀ A. A ∈ V ⟹ mutual_indep_set A (V - ({A} ∪ neighborhood A))"
```

- Locales can be combined no matter their “mathematical” context
- This combines probability with graph theory

PROBABILISTIC PROOFS: “TRANSFERRING” INFORMATION ACROSS LOCALES

- Success story: Undirected Graph Library was easily integrated with formalisation also involving locales from abstract algebra and probability theory
- This formalised the Balog-Szemerédi-Gower’s theorem – a substantial and relatively recent result in Additive combinatorics (joint work with A. Koutsoukou-Argyraki, M. Baksys).

```
interpret P1: prob_space "uniform_count_measure X"
interpret P2: prob_space "uniform_count_measure X2"
interpret P3: prob_space "uniform_count_measure Y"

interpret H: fin_bipartite_graph "(?X1 ∪ Y)" "{e ∈ E. e ⊆ (?X1 ∪ Y)}" "?X1" "Y"
let ?E_loops = "mk_edge ` {(x, x') | x x'. x ∈ X2 ∧ x' ∈ X2 ∧
  (H.codegree_normalized x x' Y) ≥ ?δ ^ 3 / 128}"
interpret Γ: ulgraph "X2" "?E_loops"

have neighborhood_unchanged: "∀ x ∈ ?X1. neighbors_ss x Y = H.neighbors_ss x Y"
  using neighbors_ss_def H.neighbors_ss_def vert_adj_def H.vert_adj_def by auto
then have degree_unchanged: "∀ x ∈ ?X1. degree x = H.degree x"
  using H.degree_neighbors_ssX degree_neighbors_ssX by auto
```

PROBABILISTIC PROOFS: A LOCALE FRAMEWORK

To "introduce randomness" we must define a probability space $(\Omega, \mathcal{F}, P)$ formally

Local definitions?

Define the measure

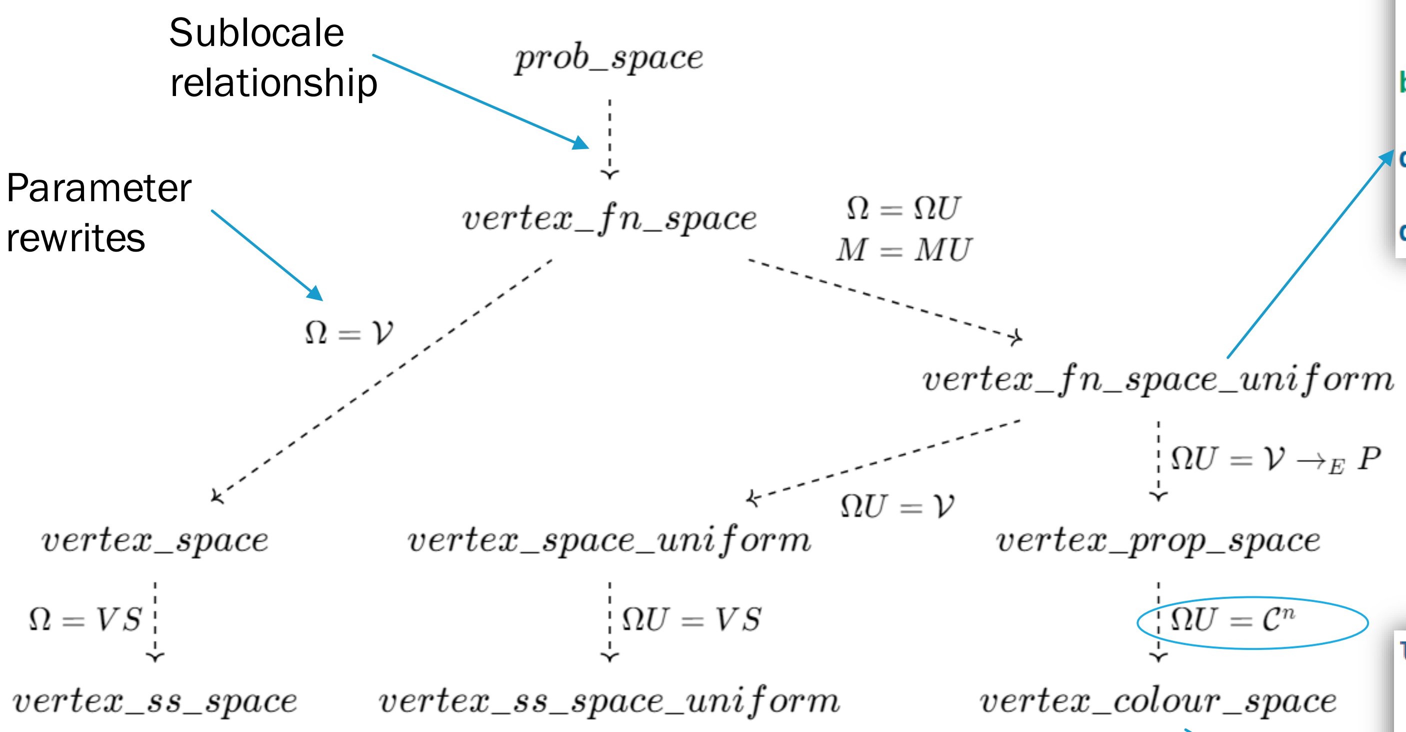
Define the prob space

Useful lemmas

```
define C where  "C = (all_n_vertex_colourings_fun 2)"
let ?M = "uniform_count_measure C"
interpret P: prob_space ?M
  using assms(1) by (intro prob_space_uniform_count_measure)(simp_all add: C_def vertex
have sp: "space ?M = C"
  by (simp add: space_uniform_count_measure)
have sts: "P.events = Pow C" by (simp add: sets_uniform_count_measure)
have finE: "finite (set_mset E)" by simp
have finC: "finite C" using vertex_colourings_fun_fin C_def by simp
have Ccard: "card C = 2 powi (card 𝒱)" using count_vertex_colourings_fun C_def by auto
```

Can we generalise?

PROBABILISTIC PROOFS: A LOCALE FRAMEWORK




```
locale vertex_fn_space_uniform = fin_hypersystem_vne +
  fixes F :: "'a set ⇒ 'b set"
  assumes ne: "F 𝒱 ≠ {}"
  assumes fin: "finite (F 𝒱)"
begin

definition "ΩU ≡ F 𝒱"

definition "MU ≡ uniform_count_measure ΩU"
```

```
locale vertex_colour_space = fin_hypergraph_nt +
  fixes n :: nat (*Number of colours *)
  assumes n_lt_order: "n ≤ horder"
  assumes n_not_zero: "n ≠ 0"


sublocale vertex_colour_space ⊆ vertex_prop_space 𝒱 E "{0..<n}"
  rewrites "ΩU = C↑n"
```

PROBABILISTIC PROOFS: A VERTEX COLOURING SPACE EXAMPLE

```
locale vertex_colour_space = fin_hypergraph_nt +
  fixes n :: nat (*Number of colours *)
  assumes n_lt_order: "n ≤ order"
  assumes n_not_zero: "n ≠ 0"

sublocale vertex_colour_space ⊆ vertex_prop_space 𝒱 E "{0..<n}"
  rewrites "ΩU = 𝒞ⁿ"
proof -
  have "{0..<n} ≠ {}" using n_not_zero by simp
  then interpret vertex_prop_space 𝒱 E "{0..<n}"
    by (unfold_locales) (simp_all)
  show "vertex_prop_space 𝒱 E {0..<n}" by (unfold_locales)
  show "ΩU = 𝒞ⁿ"
    using Ω_def all_n_vertex_colourings_alt by auto
qed
```

Context contains general lemmas on vertex colourings for any future applications of the probabilistic method to colourings!

PROBABILISTIC PROOFS: FRAMEWORK IN ACTION

Proposition 1.3.1 [Erdős (1963a)] *Every n-uniform hypergraph with less than 2^{n-1} edges has property B. Therefore $m(n) \geq 2^{n-1}$.*

Proof. Let $H = (V, E)$ be an n-uniform hypergraph with less than 2^{n-1} edges. Color V randomly by two colors. For each edge $e \in E$, let A_e be the event that e is monochromatic. Clearly $\Pr[A_e] = 2^{1-n}$. Therefore

$$\Pr\left[\bigvee_{e \in E} A_e\right] \leq \sum_{e \in E} \Pr[A_e] < 1$$

and there is a two-coloring without monochromatic edges. ∎

```
context fin_kuniform_hypergraph_nt
begin
proposition erdos_propertyB:
  assumes "size E < (2^(k - 1))"
  assumes "k > 0"
  shows "has_property_B"
proof -
(* (1) Set up the probability space: "Colour V randomly with two colours" *)
  interpret P: vertex_colour_space 𝒱 E 2
    by unfold_locales (auto simp add: order_ge_two)
(* (2) define the event to avoid - monochromatic edges *)
  define A where "A ≡(λ e. {f ∈ 𝒞² . mono_edge f e})"
(* (3) Calculation 1: Clearly Pr[Ae] = 2^(1- n). *)
  have pe: "⋀ e. e ∈ set_mset E ⟹ P.prob {f ∈ 𝒞² . mono_edge f e} = 2 powi (1 - int k)"
    using P.prob_monochromatic_edge uniform assms(1) by fastforce
(* (3) Calculation 2: Have Pr (of Ae for any e) ≤ Sum over e (Pr (A e)) < 1 *)
  have "(∑e ∈ set_mset E. P.prob (A e)) < 1"
  proof -
    have "int k - 1 = int (k - 1)" using assms by linarith
    then have "card (set_mset E) < 2 powi (int k - 1)" using card_size_set_mset[of E] assms by simp
    then have "(∑e ∈ (set_mset E). P.prob (A e)) < 2 powi (int k - 1) * 2 powi (1 - int k)"
      unfolding A_def using pe by simp
    moreover have "((2 :: real) powi ((int k) - 1)) * (2 powi (1 - (int k))) = 1"
      using power_int_add[of 2 "int k - 1" "1- int k"] by force
    ultimately show ?thesis using power_int_add[of 2 "int k - 1" "1- int k"] by simp
  qed
  moreover have "A ` (set_mset E) ⊆ P.events" unfolding A_def P.sets_eq by blast
(* (4) obtain a colouring avoiding bad events *)
  ultimately obtain f where "f ∈ 𝒞²" and "f ∉ ⋃(A `(set_mset E))"
    using P.Union_bound_obtain_fun[of "set_mset E" A] finite_set_mset P.space_eq by auto
  thus ?thesis using event_is_proper_colouring A_def is_n_colourable_def by auto
qed
```

PROBABILISTIC PROOFS: FRAMEWORK IN ACTION

Proposition 1.3.1 [Erdős (1963a)] *Every n-uniform hypergraph with less than 2^{n-1} edges has property B. Therefore $m(n) \geq 2^{n-1}$.*

Proof. Let $H = (V, E)$ be an n-uniform hypergraph with less than 2^{n-1} edges. Color V randomly by two colors. For each edge $e \in E$, let A_e be the event that e is monochromatic. Clearly $\Pr[A_e] = 2^{1-n}$. Therefore

$$\Pr\left[\bigvee_{e\in E} A_e\right] \leq \sum_{e\in E} \Pr[A_e] < 1$$

and there is a two-coloring without monochromatic edges. ■

```
context fin_kuniform_hypergraph_nt
begin
proposition erdos_propertyB:
  assumes "size E < (2^(k - 1))"
  assumes "k > 0"
  shows "has_property_B"
proof -
(* (1) Set up the probability space: "Colour V randomly with two colours" *)
  interpret P: vertex_colour_space 𝒱 E 2
    by unfold_locales (auto simp add: order_ge_two)
(* (2) define the event to avoid - monochromatic edges *)
  define A where "A ≡(λ e. {f ∈ 𝒞² . mono_edge f e})"
(* (3) Calculation 1: Clearly Pr[Ae] = 2^(1- n). *)
  have pe: "⋀ e. e ∈ set_mset E ⟹ P.prob {f ∈ 𝒞² . mono_edge f e} = 2 powi (1 - int k)"
    using P.prob_monochromatic_edge uniform assms(1) by fastforce
(* (3) Calculation 2: Have Pr (of Ae for any e) ≤ Sum over e (Pr (A e)) < 1 *)
  have "(∑e ∈ set_mset E. P.prob (A e)) < 1"
  proof -
    have "int k - 1 = int (k - 1)" using assms by linarith
    then have "card (set_mset E) < 2 powi (int k - 1)" using card_size_set_mset[of E] assms by simp
    then have "(∑e ∈ (set_mset E). P.prob (A e)) < 2 powi (int k - 1) * 2 powi (1 - int k)"
      unfolding A_def using pe by simp
    moreover have "((2 :: real) powi ((int k) - 1)) * (2 powi (1 - (int k))) = 1"
      using power_int_add[of 2 "int k - 1" "1- int k"] by force
    ultimately show ?thesis using power_int_add[of 2 "int k - 1" "1- int k"] by simp
  qed
  moreover have "A ` (set_mset E) ⊆ P.events" unfolding A_def P.sets_eq by blast
(* (4) obtain a colouring avoiding bad events *)
  ultimately obtain f where "f ∈ 𝒞²" and "f ∉ ⋃(A `(set_mset E))"
    using P.Union_bound_obtain_fun[of "set_mset E" A] finite_set_mset P.space_eq by auto
  thus ?thesis using event_is_proper_colouring A_def is_n_colourable_def by auto
qed
```

LOCALES: ADVANTAGES VS LIMITATIONS

OVERVIEW: ADVANTAGES & LIMITATIONS

Advantages

- Facilitates a “little theories” approach
- Removes duplication
- Increases flexibility and extensibility.
- Easy hierarchy manipulation
- Significant notational benefits.
- Proofs became much neater.
- Transfer of properties
- More modular proofs & proof techniques

Limitations

- Lack of Automation
- Increasingly complex locale hierarchy, where sublocale relationships must be maintained.
- Using locale specifications outside of a locale context lacks support (Notational etc)
- Can’t naturally define definitions involving multiple instances of structures

KEY SUCCESSES SO FAR

- This work in combinatorial structure hierarchies
- Extensions on this work to create a modular proof framework for the probabilistic method.
- The original fundamental work by Ballarin on Algebra (https://dl.acm.org/doi/abs/10.1007/s10817-019-09537-9
- Work on formalising Schemes in Simple Type Theory by Bordg, Paulson, & Li (https://arxiv.org/abs/2104.09366)
- Work on formalising omega categories (Bordg & Mateo)
  https://dl.acm.org/doi/abs/10.1145/3573105.3575679

RESULTS PROVED USING LOCALE-CENTRIC STRUCTURE

- Design Properties
  - Necessary conditions/basic constructions (BIBD’s, symmetric, derived, residual)
  - Symmetric Intersection Theorem
  - Wilson’s construction
  - Bose’s inequality
  - Fisher’s Inequality (& many variations)
- Szemerédi’s Regularity Lemma/Roth’s Theorem (alteration from published version)
- Balog-Szemerédi-Gowers theorem
- Lovász Local Lemma
- Bounds on vertex colouring properties of hypergraphs.

(and more…)

CONCLUDING THOUGHTS

CONTACT ME!

C.L.EDMONDS@SHEFFIELD.AC.UK

- Locales have a lot of potential to be the new "go-to" in Isabelle for large hierarchies relying flexibility, modularity, and transference of data
  - Not limited to mathematical hierarchies!
- Next steps
  - Increase automation
  - More natural ways to work with locales outside contexts.
  - More specific tactics, tools, and tutorials.
- Relevant Papers:
  - A Modular First Formalisation of Combinatorial Design Theory (with L. Paulson)
  - A Formalisation of the Balog-Szemerédi-Gowers Theorem in Isabelle/HOL (with A. Koutsoukou-Argyraki, M. Baksys, E.)
  - Formal Probabilistic Methods for Combinatorial Structures using the Lovász Local Lemma (with L. Paulson)
  - To come: paper on overall approach!