

# Design Theory

Chelsea Edmonds and Lawrence Paulson

February 6, 2026

## Abstract

Combinatorial design theory studies incidence set systems with certain balance and symmetry properties. It is closely related to hypergraph theory. This formalisation presents a general library for formal reasoning on incidence set systems, designs and their applications, including formal definitions and proofs for many key properties, operations, and theorems on the construction and existence of designs. Notably, this includes formalising  $t$ -designs, balanced incomplete block designs (BIBD), group divisible designs (GDD), pairwise balanced designs (PBD), design isomorphisms, and the relationship between graphs and designs. A locale-centric approach has been used to manage the relationships between the many different types of designs. Theorems of particular interest include the necessary conditions for existence of a BIBD, Wilson's construction on GDDs, and Bose's inequality on resolvable designs. This formalisation is partly presented in the paper "A Modular First Formalisation of Combinatorial Design Theory", presented at CICM 2021.

## Contents

<b>1</b>	<b>Micellaneous Helper Functions on Sets and Multisets</b>	<b>3</b>
1.1	Set Theory Extras . . . . .	3
1.2	Multiset Helpers . . . . .	7
1.3	Partitions on Multisets . . . . .	15
<b>2</b>	<b>Design Theory Basics</b>	<b>21</b>
2.1	Initial setup . . . . .	22
2.2	Incidence System . . . . .	22
2.3	Finite Incidence Systems . . . . .	23
2.4	Designs . . . . .	23
2.5	Core Property Definitions . . . . .	25
2.5.1	Replication Number . . . . .	25
2.5.2	Point Index . . . . .	26
2.5.3	Intersection Number . . . . .	29
2.6	Incidence System Set Property Definitions . . . . .	30

2.7	Basic Constructions on designs . . . . .	32
2.7.1	Design Complements . . . . .	32
2.7.2	Multiples . . . . .	35
2.8	Simple Designs . . . . .	37
2.9	Proper Designs . . . . .	38
<b>3</b>	<b>Design Operations</b>	<b>39</b>
3.1	Incidence system definitions . . . . .	39
3.2	Incidence System Interpretations . . . . .	45
3.3	Operation Closure for Designs . . . . .	45
3.4	Combining Set Systems . . . . .	49
<b>4</b>	<b>Block and Balanced Designs</b>	<b>51</b>
4.1	Block Designs . . . . .	51
4.1.1	K Block Designs . . . . .	51
4.1.2	Uniform Block Design . . . . .	51
4.1.3	Incomplete Designs . . . . .	53
4.2	Balanced Designs . . . . .	54
4.2.1	Sub-types of t-wise balance . . . . .	55
4.2.2	Covering and Packing Designs . . . . .	56
4.3	Constant Replication Design . . . . .	58
4.4	T-designs . . . . .	59
4.5	Steiner Systems . . . . .	61
4.6	Combining block designs . . . . .	61
<b>5</b>	<b>BIBD's</b>	<b>63</b>
5.1	BIBD Basics . . . . .	63
5.2	Necessary Conditions for Existence . . . . .	64
5.2.1	BIBD Param Relationships . . . . .	67
5.3	Constructing New bibd's . . . . .	68
5.3.1	BIBD Complement, Multiple, Combine . . . . .	68
5.3.2	Derived Designs . . . . .	70
5.3.3	Residual Designs . . . . .	71
5.4	Symmetric BIBD's . . . . .	73
5.4.1	Intersection Property on Symmetric BIBDs . . . . .	74
5.4.2	Symmetric BIBD is Simple . . . . .	81
5.4.3	Residual/Derived Sym BIBD Constructions . . . . .	82
5.5	BIBD's and Other Block Designs . . . . .	84
<b>6</b>	<b>Resolvable Designs</b>	<b>85</b>
6.1	Resolutions and Resolution Classes . . . . .	85
6.2	Resolvable Design Locale . . . . .	86
6.3	Resolvable Block Designs . . . . .	87
6.3.1	Bose's Inequality . . . . .	89

<b>7</b>	<b>Group Divisible Designs</b>	<b>90</b>
7.1	Group design . . . . .	90
7.1.1	Group Type . . . . .	92
7.1.2	Uniform Group designs . . . . .	95
7.2	GDD . . . . .	96
7.2.1	Sub types of GDD's . . . . .	99
7.3	GDD and PBD Constructions . . . . .	99
7.3.1	GDD Delete Point construction . . . . .	99
7.3.2	PBD construction from GDD . . . . .	101
7.3.3	Wilson's Construction . . . . .	105
<b>8</b>	<b>Graphs and Designs</b>	<b>115</b>
8.1	Non-empty digraphs . . . . .	115
8.2	Arcs to Blocks . . . . .	115
8.3	Graphs are designs . . . . .	117
8.4	R-regular graphs . . . . .	118
<b>9</b>	<b>Sub-designs</b>	<b>122</b>
9.1	Sub-system and Sub-design Locales . . . . .	122
9.2	Reasoning on Sub-designs . . . . .	124
9.2.1	Reasoning on Incidence Sys property relationships . . . . .	124
9.2.2	Reasoning on Incidence Sys/Design operations . . . . .	124
<b>10</b>	<b>Design Isomorphisms</b>	<b>126</b>
10.1	Images of Set Systems . . . . .	126
10.2	Incidence System Isomorphisms . . . . .	127
10.3	Design Isomorphisms . . . . .	130
10.3.1	Isomorphism Operation . . . . .	130
10.3.2	Design Properties/Operations under Isomorphism . . . . .	131

# 1 Micellaneous Helper Functions on Sets and Multisets

```

theory Multisets-Extras imports
  HOL-Library.Multiset
  Card-Partitions.Set-Partition
  Nested-Multisets-Ordinals.Multiset-More
  Nested-Multisets-Ordinals.Duplicate-Free-Multiset
  HOL-Library.Disjoint-Sets
begin

```

## 1.1 Set Theory Extras

A number of extra helper lemmas for reasoning on sets (finite) required for Design Theory proofs

```

lemma card-Pow-filter-one:
  assumes finite A
  shows  $\text{card } \{x \in \text{Pow } A . \text{card } x = 1\} = \text{card } (A)$ 
  using assms
proof (induct rule: finite-induct)
  case empty
  then show ?case by auto
next
  case (insert x F)
  have  $\text{Pow } (\text{insert } x \text{ ' } F) = \text{Pow } F \cup \text{insert } x \text{ ' } \text{Pow } F$ 
  by (simp add: Pow-insert)
  then have split:  $\{y \in \text{Pow } (\text{insert } x \text{ ' } F) . \text{card } y = 1\} =$ 
     $\{y \in (\text{Pow } F) . \text{card } y = 1\} \cup \{y \in (\text{insert } x \text{ ' } \text{Pow } F) . \text{card } y = 1\}$ 
  by blast
  have  $\bigwedge y . y \in (\text{insert } x \text{ ' } \text{Pow } F) \implies \text{finite } y$ 
  using finite-subset insert.hyps(1) by fastforce
  then have single:  $\bigwedge y . y \in (\text{insert } x \text{ ' } \text{Pow } F) \implies \text{card } y = 1 \implies y = \{x\}$ 
  by (metis card-1-singletonE empty-iff image-iff insertCI insertE)
  then have  $\text{card } \{y \in (\text{insert } x \text{ ' } \text{Pow } F) . \text{card } y = 1\} = 1$ 
  using empty-iff imageI is-singletonI is-singletonI' is-singleton-altdef
  by (metis (full-types, lifting) Collect-empty-eq-bot Pow-bottom bot-empty-eq
mem-Collect-eq)
  then have  $\{y \in (\text{insert } x \text{ ' } \text{Pow } F) . \text{card } y = 1\} = \{\{x\}\}$ 
  using single card-1-singletonE card-eq-0-iff
  by (smt (verit) empty-Collect-eq mem-Collect-eq singletonD zero-neq-one)
  then have split2:  $\{y \in \text{Pow } (\text{insert } x \text{ ' } F) . \text{card } y = 1\} = \{y \in (\text{Pow } F) . \text{card } y$ 
     $= 1\} \cup \{\{x\}\}$ 
  using split by simp
  then show ?case
  proof (cases x ∈ F)
  case True
  then show ?thesis using insert.hyps(2) by auto
next
  case False
  then have  $\{y \in (\text{Pow } F) . \text{card } y = 1\} \cap \{\{x\}\} = \{\}$  by blast
  then have fact:  $\text{card } \{y \in \text{Pow } (\text{insert } x \text{ ' } F) . \text{card } y = 1\} =$ 
     $\text{card } \{y \in (\text{Pow } F) . \text{card } y = 1\} + \text{card } \{\{x\}\}$ 
  using split2 card-Un-disjoint insert.hyps(1) by auto
  have  $\text{card } (\text{insert } x \text{ ' } F) = \text{card } F + 1$ 
  using False card-insert-disjoint by (metis Suc-eq-plus1 insert.hyps(1))
  then show ?thesis using fact insert.hyps(3) by auto
qed
qed

```

```

lemma elem-exists-non-empty-set:
  assumes  $\text{card } A > 0$ 
  obtains x where  $x \in A$ 
  using assms card-gt-0-iff by fastforce

```

**lemma** *set-self-img-compr*:  $\{a \mid a . a \in A\} = A$   
**by** *blast*

**lemma** *card-subset-not-gt-card*:  $\text{finite } A \implies \text{card } ps > \text{card } A \implies \neg (ps \subseteq A)$   
**using** *card-mono leD* **by** *auto*

**lemma** *card-inter-lt-single*:  $\text{finite } A \implies \text{finite } B \implies \text{card } (A \cap B) \leq \text{card } A$   
**by** (*simp add: card-mono*)

**lemma** *set-diff-non-empty-not-subset*:

**assumes**  $A \subseteq (B - C)$

**assumes**  $C \neq \{\}$

**assumes**  $A \neq \{\}$

**assumes**  $B \neq \{\}$

**shows**  $\neg (A \subseteq C)$

**proof** (*rule ccontr*)

**assume**  $\neg \neg (A \subseteq C)$

**then have**  $a: \bigwedge x . x \in A \implies x \in C$  **by** *blast*

**thus** *False* **using**  $a$  *assms* **by** *blast*

**qed**

**lemma** *set-card-diff-ge-zero*:  $\text{finite } A \implies \text{finite } B \implies A \neq B \implies \text{card } A = \text{card } B \implies$

$\text{card } (A - B) > 0$

**by** (*meson Diff-eq-empty-iff card-0-eq card-subset-eq finite-Diff neq0-conv*)

**lemma** *set-filter-diff*:  $\{a \in A . P a\} - \{x\} = \{a \in (A - \{x\}) . (P a)\}$   
**by** (*auto*)

**lemma** *set-filter-diff-card*:  $\text{card } (\{a \in A . P a\} - \{x\}) = \text{card } \{a \in (A - \{x\}) . (P a)\}$

**by** (*simp add: set-filter-diff*)

**lemma** *obtain-subset-with-card-int-n*:

**assumes**  $(n :: \text{int}) \leq \text{of-nat } (\text{card } S)$

**assumes**  $(n :: \text{int}) \geq 0$

**obtains**  $T$  **where**  $T \subseteq S$  *of-nat*  $(\text{card } T) = (n :: \text{int})$  *finite*  $T$

**using** *obtain-subset-with-card-n assms*

**by** (*metis nonneg-int-cases of-nat-le-iff*)

**lemma** *transform-filter-img-empty-rm*:

**assumes**  $\bigwedge g . g \in G \implies g \neq \{\}$

**shows**  $\{g - \{x\} \mid g . g \in G \wedge g \neq \{x\}\} = \{g - \{x\} \mid g . g \in G\} - \{\{\}\}$

**proof** –

**let**  $?f = \lambda g . g - \{x\}$

**have**  $\bigwedge g . g \in G \implies g \neq \{x\} \longleftrightarrow ?f g \neq \{\}$  **using** *assms*

**by** (*metis Diff-cancel Diff-empty Diff-insert0 insert-Diff*)

**thus** *?thesis* **by** *auto*

**qed**

**lemma** *bij-betw-inter-subsets*:  $\text{bij-betw } f \ A \ B \implies a1 \subseteq A \implies a2 \subseteq A$   
 $\implies f' (a1 \cap a2) = (f' a1) \cap (f' a2)$   
**by** (*meson bij-betw-imp-inj-on inj-on-image-Int*)

Partition related set theory lemmas

**lemma** *partition-on-remove-pt*:

**assumes** *partition-on*  $A \ G$

**shows** *partition-on*  $(A - \{x\}) \ \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\}$

**proof** (*intro partition-onI*)

**show**  $\bigwedge p. p \in \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\} \implies p \neq \{\}$

**using** *assms partition-onD3 subset-singletonD* **by** *force*

**let**  $?f = (\lambda g. g - \{x\})$

**have** *un-img*:  $\bigcup (\{?f \ g \mid g. g \in G\}) = ?f (\bigcup G)$  **by** *blast*

**have** *empty*:  $\bigcup \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\} = \bigcup (\{g - \{x\} \mid g. g \in G\} - \{\{\}\})$

**by** *blast*

**then have**  $\bigcup (\{g - \{x\} \mid g. g \in G\} - \{\{\}\}) = \bigcup (\{g - \{x\} \mid g. g \in G\})$  **by** *blast*

**then show**  $\bigcup \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\} = A - \{x\}$  **using** *partition-onD1* *assms un-img*

**by** (*metis empty*)

**then show**  $\bigwedge p \ p'$ .

$p \in \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\} \implies$

$p' \in \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\} \implies p \neq p' \implies p \cap p' = \{\}$

**proof** -

**fix**  $p1 \ p2$

**assume**  $p1$ :  $p1 \in \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\}$

**and**  $p2$ :  $p2 \in \{g - \{x\} \mid g. g \in G \wedge g \neq \{x\}\}$

**and**  $ne$ :  $p1 \neq p2$

**obtain**  $p1' \ p2'$  **where** *orig1*:  $p1 = p1' - \{x\}$  **and** *orig2*:  $p2 = p2' - \{x\}$

**and** *origne*:  $p1' \neq p2'$  **and** *ne1*:  $p1' \neq \{x\}$  **and** *ne2*:  $p2' \neq \{x\}$  **and** *ing1*:

$p1' \in G$

**and** *ing2*:  $p2' \in G$

**using**  $p1 \ p2$  **using** *mem-Collect-eq ne* **by** *blast*

**then have**  $p1' \cap p2' = \{\}$  **using** *assms partition-onD2 ing1 ing2 origne disjointD* **by** *blast*

**thus**  $p1 \cap p2 = \{\}$  **using** *orig1 orig2* **by** *blast*

**qed**

**qed**

**lemma** *partition-on-cart-prod*:

**assumes**  $\text{card } I > 0$

**assumes**  $A \neq \{\}$

**assumes**  $G \neq \{\}$

**assumes** *partition-on*  $A \ G$

**shows** *partition-on*  $(A \times I) \ \{g \times I \mid g. g \in G\}$

**proof** (*intro partition-onI*)

**show**  $\bigwedge p. p \in \{g \times I \mid g. g \in G\} \implies p \neq \{\}$

**using** *assms(1) assms(4) partition-onD3* **by** *fastforce*  
**show**  $\bigcup \{g \times I \mid g. g \in G\} = A \times I$   
**by** (*metis Setcompr-eq-image Sigma-Union assms(4) partition-onD1*)  
**show**  $\bigwedge p p'. p \in \{g \times I \mid g. g \in G\} \implies p' \in \{g \times I \mid g. g \in G\} \implies p \neq p' \implies p \cap p' = \{\}$   
**by** (*smt (verit, best) Sigma-Int-distrib1 Sigma-empty1 assms(4) mem-Collect-eq partition-onE*)  
**qed**

## 1.2 Multiset Helpers

Generic Size, count and card helpers

**lemma** *count-size-set-repr*:  $size \{\# x \in \# A . x = g\# \} = count A g$   
**by** (*simp add: filter-eq-replicate-mset*)

**lemma** *mset-nempty-set-nempty*:  $A \neq \{\#\} \longleftrightarrow (set-mset A) \neq \{\}$   
**by** *simp*

**lemma** *mset-size-ne0-set-card*:  $size A > 0 \implies card (set-mset A) > 0$   
**using** *mset-nempty-set-nempty* **by** *fastforce*

**lemma** *set-count-size-min*:  $count A a \geq n \implies size A \geq n$   
**by** (*metis (full-types) count-le-replicate-mset-subset-eq size-mset-mono size-replicate-mset*)

**lemma** *card-size-filter-eq*:  $finite A \implies card \{a \in A . P a\} = size \{\# a \in \# mset-set A . P a\# \}$   
**by** *simp*

**lemma** *size-multiset-set-mset-const-count*:

**assumes**  $card (set-mset A) = ca$   
**assumes**  $\bigwedge p. p \in \# A \implies count A p = ca2$   
**shows**  $size A = (ca * ca2)$

**proof** –

**have**  $size A = (\sum p \in (set-mset A) . count A p)$  **using** *size-multiset-overloaded-eq*  
**by** *auto*

**then have**  $size A = (\sum p \in (set-mset A) . ca2)$  **using** *assms* **by** *simp*

**thus** *?thesis* **using** *assms(1)* **by** *auto*

**qed**

**lemma** *size-multiset-int-count*:

**assumes**  $of-nat (card (set-mset A)) = (ca :: int)$   
**assumes**  $\bigwedge p. p \in \# A \implies of-nat (count A p) = (ca2 :: int)$   
**shows**  $of-nat (size A) = ((ca :: int) * ca2)$

**proof** –

**have**  $size A = (\sum p \in (set-mset A) . count A p)$  **using** *size-multiset-overloaded-eq*  
**by** *auto*

**then have**  $of-nat (size A) = (\sum p \in (set-mset A) . ca2)$  **using** *assms* **by** *simp*

**thus** *?thesis* **using** *assms(1)* **by** *auto*

**qed**

**lemma** *mset-union-size*:  $\text{size } (A \cup\# B) = \text{size } (A) + \text{size } (B - A)$   
**by** (*simp add: union-mset-def*)

**lemma** *mset-union-size-inter*:  $\text{size } (A \cup\# B) = \text{size } (A) + \text{size } B - \text{size } (A \cap\# B)$   
**by** (*metis diff-add-inverse2 size-Un-Int*)

Lemmas for `repeat_mset`

**lemma** *repeat-mset-size* [*simp*]:  $\text{size } (\text{repeat-mset } n A) = n * \text{size } A$   
**by** (*induction n*) *auto*

**lemma** *repeat-mset-subset-in*:  
**assumes**  $\bigwedge a . a \in\# A \implies a \subseteq B$   
**assumes**  $X \in\# \text{repeat-mset } n A$   
**assumes**  $x \in X$   
**shows**  $x \in B$   
**using** *assms* **by** (*induction n*) *auto*

**lemma** *repeat-mset-not-empty*:  $n > 0 \implies A \neq \{\#\} \implies \text{repeat-mset } n A \neq \{\#\}$   
**by** (*induction n*) *auto*

**lemma** *elem-in-repeat-in-original*:  $a \in\# \text{repeat-mset } n A \implies a \in\# A$   
**by** (*metis count-inI count-repeat-mset in-countE mult.commute mult-zero-left nat.distinct(1)*)

**lemma** *elem-in-original-in-repeat*:  $n > 0 \implies a \in\# A \implies a \in\# \text{repeat-mset } n A$   
**by** (*metis count-greater-zero-iff count-repeat-mset nat-0-less-mult-iff*)

Lemmas on image and filter for multisets

**lemma** *multiset-add-filter-size*:  $\text{size } \{\# a \in\# (A1 + A2) . P a \#\} = \text{size } \{\# a \in\# A1 . P a \#\} +$   
 $\text{size } \{\# a \in\# A2 . P a \#\}$   
**by** *simp*

**lemma** *size-filter-neg*:  $\text{size } \{\# a \in\# A . P a \#\} = \text{size } A - \text{size } \{\# a \in\# A . \neg P a \#\}$   
**using** *size-filter-mset-lesseq size-union union-filter-mset-complement*  
**by** (*metis ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add*)

**lemma** *filter-filter-mset-cond-simp*:  
**assumes**  $\bigwedge a . P a \implies Q a$   
**shows**  $\text{filter-mset } P A = \text{filter-mset } P (\text{filter-mset } Q A)$   
**proof** –  
**have**  $\text{filter-mset } P (\text{filter-mset } Q A) = \text{filter-mset } (\lambda a . Q a \wedge P a) A$   
**by** (*simp add: filter-filter-mset*)  
**thus** *?thesis* **using** *assms*  
**by** (*metis (mono-tags, lifting) filter-mset-cong*)  
**qed**

**lemma** *filter-filter-mset-ss-member*:  $filter\text{-}mset (\lambda a . \{x, y\} \subseteq a) A =$   
 $filter\text{-}mset (\lambda a . \{x, y\} \subseteq a) (filter\text{-}mset (\lambda a . x \in a) A)$

**proof** –

**have** *filter*:  $filter\text{-}mset (\lambda a . \{x, y\} \subseteq a) (filter\text{-}mset (\lambda a . x \in a) A) =$   
 $filter\text{-}mset (\lambda a . x \in a \wedge \{x, y\} \subseteq a) A$  **by** (*simp add: filter-filter-mset*)

**have**  $\bigwedge a. \{x, y\} \subseteq a \implies x \in a$  **by** *simp*

**thus** *?thesis using filter by auto*

**qed**

**lemma** *mset-image-do-nothing*:  $(\bigwedge x . x \in\# A \implies f x = x) \implies image\text{-}mset f$   
 $A = A$

**by** (*induct A auto*)

**lemma** *set-mset-filter*:  $set\text{-}mset \{\# f a . a \in\# A \#\} = \{f a \mid a. a \in\# A\}$   
**by** (*simp add: Setcompr-eq-image*)

**lemma** *mset-exists-imply*:  $x \in\# \{\# f a . a \in\# A \#\} \implies \exists y \in\# A . x = f y$   
**by** *auto*

**lemma** *filter-mset-image-mset*:

$filter\text{-}mset P (image\text{-}mset f A) = image\text{-}mset f (filter\text{-}mset (\lambda x. P (f x)) A)$   
**by** (*induction A auto*)

**lemma** *mset-bunion-filter*:  $\{\# a \in\# A . P a \vee Q a \#\} = \{\# a \in\# A . P a \#\} \cup$   
 $\{\# a \in\# A . Q a \#\}$

**by** (*rule multiset-eqI simp*)

**lemma** *mset-inter-filter*:  $\{\# a \in\# A . P a \wedge Q a \#\} = \{\# a \in\# A . P a \#\} \cap$   
 $\{\# a \in\# A . Q a \#\}$

**by** (*rule multiset-eqI simp*)

**lemma** *image-image-mset*:  $image\text{-}mset (\lambda x . f x) (image\text{-}mset (\lambda y . g y) A) =$   
 $image\text{-}mset (\lambda x . f (g x)) A$

**by** *simp*

Big Union over multiset helpers

**lemma** *mset-big-union-obtain*:

**assumes**  $x \in\# \sum\# A$

**obtains**  $a$  **where**  $a \in\# A$  **and**  $x \in\# a$

**using** *assms by blast*

**lemma** *size-big-union-sum*:  $size (\sum\# (M :: 'a \text{ multiset multiset})) = (\sum x \in\# M .$   
 $size x)$

**by** (*induct M auto*)

Cartesian Product on Multisets

**lemma** *size-cartesian-product-singleton* [*simp*]:  $size (\{\# a \#\} \times\# B) = size B$

**by** (*simp add: Times-mset-single-left*)

**lemma** *size-cartesian-product-singleton-right* [*simp*]:  $\text{size } (A \times\# \{\#b\}) = \text{size } A$

**by** (*simp add: Times-mset-single-right*)

**lemma** *size-cartesian-product-empty* [*simp*]:  $\text{size } (A \times\# \{\#\}) = 0$

**by** *simp*

**lemma** *size-add-elem-step-eq*:

**assumes**  $\text{size } (A \times\# B) = \text{size } A * \text{size } B$

**shows**  $\text{size } (\text{add-mset } x A \times\# B) = \text{size } (\text{add-mset } x A) * \text{size } B$

**proof** –

**have**  $(\text{add-mset } x A \times\# B) = A \times\# B + \{\#x\} \times\# B$

**by** (*metis Sigma-mset-plus-distrib1 add-mset-add-single*)

**then have**  $\text{size } (\text{add-mset } x A \times\# B) = \text{size } (A \times\# B) + \text{size } B$  **by** *auto*

**also have**  $\dots = \text{size } A * \text{size } B + \text{size } B$

**by** (*simp add: assms*)

**finally have**  $\text{size } (\text{add-mset } x A \times\# B) = (\text{size } A + 1) * \text{size } B$

**by** *auto*

**thus** *?thesis* **by** *simp*

**qed**

**lemma** *size-cartesian-product*:  $\text{size } (A \times\# B) = \text{size } A * \text{size } B$

**by** (*induct A*) (*simp-all add: size-add-elem-step-eq*)

**lemma** *cart-prod-distinct-mset*:

**assumes** *assm1*: *distinct-mset A*

**assumes** *assm2*: *distinct-mset B*

**shows** *distinct-mset*  $(A \times\# B)$

**unfolding** *distinct-mset-count-less-1*

**proof** (*rule allI*)

**fix** *x*

**have** *count-mult*:  $\text{count } (A \times\# B) x = \text{count } A (\text{fst } x) * \text{count } B (\text{snd } x)$

**using** *count-Sigma-mset* **by** (*metis prod.exhaust-sel*)

**then have**  $\text{count } A (\text{fst } x) * \text{count } B (\text{snd } x) \leq 1$  **using** *assm1 assm2*

**unfolding** *distinct-mset-count-less-1* **using** *mult-le-one* **by** *blast*

**thus**  $\text{count } (A \times\# B) x \leq 1$  **using** *count-mult* **by** *simp*

**qed**

**lemma** *cart-product-single-intersect*:  $x1 \neq x2 \implies (\{\#x1\} \times\# A) \cap\# (\{\#x2\} \times\# B) = \{\#\}$

**using** *multiset-inter-single* **by** *fastforce*

**lemma** *size-union-distinct-cart-prod*:  $x1 \neq x2 \implies \text{size } ((\{\#x1\} \times\# A) \cup\# (\{\#x2\} \times\# B)) =$

$\text{size } (\{\#x1\} \times\# A) + \text{size } (\{\#x2\} \times\# B)$

**by** (*simp add: cart-product-single-intersect size-Un-disjoint*)

**lemma** *size-Union-distinct-cart-prod*: *distinct-mset M*  $\implies$

$size (\sum p \in \#M. (\{ \#p \# \} \times \# B)) = size (M) * size (B)$   
**by** (*induction M*) *auto*

**lemma** *size-Union-distinct-cart-prod-filter: distinct-mset M*  $\implies$   
 $(\bigwedge p . p \in \# M \implies size (\{ \# b \in \# B . P p b \# \}) = c) \implies$   
 $size (\sum p \in \#M. (\{ \#p \# \} \times \# \{ \# b \in \# B . P p b \# \})) = size (M) * c$   
**by** (*induction M*) *auto*

**lemma** *size-Union-distinct-cart-prod-filter2: distinct-mset V*  $\implies$   
 $(\bigwedge b . b \in \# B \implies size (\{ \# v \in \# V . P v b \# \}) = c) \implies$   
 $size (\sum b \in \#B. (\{ \# v \in \# V . P v b \# \} \times \# \{ \#b \# \})) = size (B) * c$   
**by** (*induction B*) *auto*

**lemma** *cart-product-add-1: (add-mset a A)  $\times$  # B = {#a#}  $\times$  # B + (A  $\times$  # B)*  
**by** (*metis Sigma-mset-plus-distrib1 add-mset-add-single union-commute*)

**lemma** *cart-product-add-1-filter: {#m  $\in$  # ((add-mset a M)  $\times$  # N) . P m #} = {#m  $\in$  # (M  $\times$  # N) . P m #} + {#m  $\in$  # ({#a#}  $\times$  # N) . P m #}*  
**unfolding** *add-mset-add-single [of a M] Sigma-mset-plus-distrib1*  
**by** (*simp add: Times-mset-single-left*)

**lemma** *cart-product-add-1-filter2: {#m  $\in$  # (M  $\times$  # (add-mset b N)) . P m #} = {#m  $\in$  # (M  $\times$  # N) . P m #} + {#m  $\in$  # (M  $\times$  # {#b#}) . P m #}*  
**unfolding** *add-mset-add-single [of b N] Sigma-mset-plus-distrib1*  
**by** (*metis Times-insert-left Times-mset-single-right add-mset-add-single filter-union-mset*)

**lemma** *cart-prod-singleton-right-gen:*  
**assumes**  $\bigwedge x . x \in \# (A \times \# \{ \#b \# \}) \implies P x \longleftrightarrow Q (fst x)$   
**shows**  $\{ \#x \in \# (A \times \# \{ \#b \# \}). P x \# \} = \{ \# a \in \# A . Q a \# \} \times \# \{ \#b \# \}$   
**using** *assms*  
**proof** (*induction A*)  
**case** *empty*  
**then show** *?case by simp*  
**next**  
**case** (*add x A*)  
**have**  $add\text{-}mset\ x\ A\ \times\ \#\ \{ \#b \# \} = add\text{-}mset\ (x, b)\ (A\ \times\ \#\ \{ \#b \# \})$   
**by** (*simp add: Times-mset-single-right*)  
**then have** *lhs: filter-mset P (add-mset x A  $\times$  # {#b#}) = filter-mset P (A  $\times$  # {#b#}) + filter-mset P {#(x, b)#} by simp*  
**have** *rhs: filter-mset Q (add-mset x A)  $\times$  # {#b#} = filter-mset Q A  $\times$  # {#b#}*  
**+**  
 $filter\text{-}mset\ Q\ \{ \#x \# \} \times \# \{ \#b \# \}$   
**by** (*metis Sigma-mset-plus-distrib1 add-mset-add-single filter-union-mset*)  
**have**  $filter\text{-}mset\ P\ \{ \#(x, b) \# \} = filter\text{-}mset\ Q\ \{ \#x \# \} \times \# \{ \#b \# \}$   
**using** *add.premys by fastforce*  
**then show** *?case using lhs rhs add.IH add.premys by force*

qed

**lemma** *cart-prod-singleton-left-gen*:

**assumes**  $\bigwedge x . x \in \# (\{ \#a \# \} \times \# B) \implies P x \longleftrightarrow Q (\text{snd } x)$

**shows**  $\{ \#x \in \# (\{ \#a \# \} \times \# B) . P x \# \} = \{ \#a \# \} \times \# \{ \#b \in \# B . Q b \# \}$

**using** *assms*

**proof** (*induction B*)

**case** *empty*

**then show** *?case* **by** *simp*

**next**

**case** (*add x B*)

**have** *lhs*:  $\text{filter-mset } P (\{ \#a \# \} \times \# \text{add-mset } x B) = \text{filter-mset } P (\{ \#a \# \} \times \# B) +$

$\text{filter-mset } P \{ \#(a, x) \# \}$

**by** (*simp add: cart-product-add-1-filter2*)

**have** *rhs*:  $\{ \#a \# \} \times \# \text{filter-mset } Q (\text{add-mset } x B) = \{ \#a \# \} \times \# \text{filter-mset } Q B +$

$\{ \#a \# \} \times \# \text{filter-mset } Q \{ \#x \# \}$

**using** *add-mset-add-single filter-union-mset* **by** (*metis Times-mset-single-left image-mset-union*)

**have**  $\text{filter-mset } P \{ \#(a, x) \# \} = \{ \#a \# \} \times \# \text{filter-mset } Q \{ \#x \# \}$

**using** *add.prem*s **by** *fastforce*

**then show** *?case* **using** *lhs rhs add.IH add.prem*s **by** *force*

qed

**lemma** *cart-product-singleton-left*:  $\{ \#m \in \# (\{ \#a \# \} \times \# N) . \text{fst } m \in \text{snd } m \# \} =$

$\{ \#a \# \} \times \# \{ \#n \in \# N . a \in n \# \}$  (**is** *?A = ?B*)

**proof** –

**have** *stmt*:  $\bigwedge m . m \in \# (\{ \#a \# \} \times \# N) \implies \text{fst } m \in \text{snd } m \longleftrightarrow a \in \text{snd } m$

**by** (*simp add: mem-Times-mset-iff*)

**thus** *?thesis* **by** (*metis (no-types, lifting) Sigma-mset-cong stmt cart-prod-singleton-left-gen*)

qed

**lemma** *cart-product-singleton-right*:  $\{ \#m \in \# (N \times \# \{ \#b \# \}) . \text{fst } m \in \text{snd } m \# \} =$

$\{ \#n \in \# N . n \in b \# \} \times \# \{ \#b \# \}$  (**is** *?A = ?B*)

**proof** –

**have** *stmt*:  $\bigwedge m . m \in \# (N \times \# \{ \#b \# \}) \implies \text{fst } m \in \text{snd } m \longleftrightarrow \text{fst } m \in b$

**by** (*simp add: mem-Times-mset-iff*)

**thus** *?thesis* **by** (*metis (no-types, lifting) Sigma-mset-cong stmt cart-prod-singleton-right-gen*)

qed

**lemma** *cart-product-add-1-filter-eq*:  $\{ \#m \in \# ((\text{add-mset } a M) \times \# N) . (\text{fst } m \in \text{snd } m) \# \} =$

$\{ \#m \in \# (M \times \# N) . (\text{fst } m \in \text{snd } m) \# \} + (\{ \#a \# \} \times \# \{ \#n \in \# N . a \in n \# \})$

**unfolding** *add-mset-add-single* [*of a M*] *Sigma-mset-plus-distrib1*

**using** *cart-product-singleton-left cart-product-add-1-filter* **by** *fastforce*

**lemma** *cart-product-add-1-filter-eq-mirror*:  $\{\#m \in\# M \times\# (add\text{-}mset\ b\ N) . (fst\ m \in\ snd\ m)\ \#\} =$   
 $\{\#m \in\# (M \times\# N) . (fst\ m \in\ snd\ m)\ \#\} + (\{\#n \in\# M . n \in\ b\ \#\} \times\#$   
 $\{\#b\#\})$   
**unfolding** *add-mset-add-single [of b N] Sigma-mset-plus-distrib1*  
**by** (*metis (no-types) add-mset-add-single cart-product-add-1-filter<sup>2</sup> cart-product-singleton-right*)

**lemma** *set-break-down-left*:  
**shows**  $\{\#m \in\# (M \times\# N) . (fst\ m) \in\ (snd\ m)\ \#\} = (\sum\ m \in\# M . (\{\#m\#\}$   
 $\times\# \{\#n \in\# N . m \in\ n\#\}))$   
**by** (*induction M*) (*auto simp add: cart-product-add-1-filter-eq*)

**lemma** *set-break-down-right*:  
**shows**  $\{\#x \in\# M \times\# N . (fst\ x) \in\ (snd\ x)\ \#\} = (\sum\ n \in\# N . (\{\#m \in\# M .$   
 $m \in\ n\#\} \times\# \{\#n\#\}))$   
**by** (*induction N*) (*auto simp add: cart-product-add-1-filter-eq-mirror*)

Reasoning on sums of elements over multisets

**lemma** *sum-over-fun-eq*:  
**assumes**  $\bigwedge x . x \in\# A \implies f\ x = g\ x$   
**shows**  $(\sum\ x \in\# A . f(x)) = (\sum\ x \in\# A . g(x))$   
**using** *assms by auto*

**lemma** *sum-mset-add-diff-nat*:  
**fixes**  $x :: 'a$  **and**  $f\ g :: 'a \Rightarrow nat$   
**assumes**  $\bigwedge x . x \in\# A \implies f\ x \geq g\ x$   
**shows**  $(\sum\ x \in\# A . f\ x - g\ x) = (\sum\ x \in\# A . f\ x) - (\sum\ x \in\# A . g\ x)$   
**using** *assms by (induction A) (simp-all add: sum-mset-mono)*

**lemma** *sum-mset-add-diff-int*:  
**fixes**  $x :: 'a$  **and**  $f\ g :: 'a \Rightarrow int$   
**shows**  $(\sum\ x \in\# A . f\ x - g\ x) = (\sum\ x \in\# A . f\ x) - (\sum\ x \in\# A . g\ x)$   
**by** (*induction A*) (*simp-all add: sum-mset-mono*)

**context** *ring-1*  
**begin**

**lemma** *sum-mset-add-diff*:  $(\sum\ x \in\# A . f\ x - g\ x) = (\sum\ x \in\# A . f\ x) - (\sum$   
 $x \in\# A . g\ x)$   
**by** (*induction A*) (*auto simp add: algebra-simps*)

**end**

**context** *ordered-semiring*  
**begin**

**lemma** *sum-mset-ge0*:  $(\bigwedge x . f\ x \geq 0) \implies (\sum\ x \in\# A . f\ x) \geq 0$

**proof** (*induction A*)  
**case** *empty*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*add x A*)  
**then have** *hyp2: 0 ≤ sum-mset (image-mset f A)* **by** *blast*  
**then have** *sum-mset (image-mset f (add-mset x A)) = sum-mset (image-mset f A) + f x*  
**by** (*simp add: add-commute*)  
**then show** *?case*  
**by** (*simp add: add.IH add.prem*)  
**qed**

**lemma** *sum-order-add-mset: (∧ x . f x ≥ 0) ⇒ (∑ x ∈# A. f x) ≤ (∑ x ∈# add-mset a A. f x)*  
**by** (*simp add: local.add-increasing*)

**lemma** *sum-mset-0-left: (∧ x . f x ≥ 0) ⇒ (∑ x ∈# A. f x) = 0 ⇒ (∀ x ∈# A . f x = 0)*  
**apply** (*induction A*)  
**apply** *auto*  
**using** *local.add-nonneg-eq-0-iff sum-mset-ge0* **apply** *blast*  
**using** *local.add-nonneg-eq-0-iff sum-mset-ge0* **by** *blast*

**lemma** *sum-mset-0-iff-ge-0:*  
**assumes** ( $\bigwedge x . f x \geq 0$ )  
**shows** ( $\sum x \in\# A. f x = 0 \iff (\forall x \in \text{set-mset } A . f x = 0)$ )  
**using** *sum-mset-0-left assms* **by** *auto*

**end**

**lemma** *mset-set-size-card-count: (∑ x ∈# A. x) = (∑ x ∈ set-mset A . x \* (count A x))*

**proof** (*induction A*)  
**case** *empty*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*add y A*)  
**have** *lhs: (∑ x ∈# add-mset y A. x) = (∑ x ∈# A. x) + y* **by** *simp*  
**have** *rhs: (∑ x ∈ set-mset (add-mset y A). x \* count (add-mset y A) x) = (∑ x ∈ (insert y (set-mset A)). x \* count (add-mset y A) x)*  
**by** *simp*  
**then show** *?case*  
**proof** (*cases y ∈# A*)  
**case** *True*  
**have** *x-val: ∧ x . x ∈ (insert y (set-mset A)) ⇒ x ≠ y ⇒ x \* count (add-mset y A) x = x \* (count A x)*  
**by** *auto*  
**have** *y-count: count (add-mset y A) y = 1 + count A y*

```

    using True count-inI by fastforce
  then have  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
     $(y * (\text{count } (\text{add-mset } y \ A) \ y)) + (\sum x \in (\text{set-mset } A) - \{y\}. x * \text{count } A \ x)$ 
    using x-val finite-set-mset sum.cong sum.insert rhs
    by (smt (verit) DiffD1 Diff-insert-absorb insert-absorb mk-disjoint-insert
sum.insert-remove)
  then have s1:  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
     $y + y * (\text{count } A \ y) + (\sum x \in (\text{set-mset } A) - \{y\}. x * \text{count } A \ x)$ 
    using y-count by simp
  then have  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
     $y + (\sum x \in \text{insert } y \ ((\text{set-mset } A) - \{y\}) . x * \text{count } A \ x)$ 
    by (simp add: sum.insert-remove)
  then have  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
     $y + (\sum x \in (\text{set-mset } A) . x * \text{count } A \ x)$ 
    by (simp add: True insert-absorb)
  then show ?thesis using lhs add.IH
    by linarith
next
case False
have x-val:  $\bigwedge x . x \in \text{set-mset } A \implies x * \text{count } (\text{add-mset } y \ A) \ x = x * (\text{count}$ 
A x)
  using False by auto
have y-count:  $\text{count } (\text{add-mset } y \ A) \ y = 1$  using False count-inI by fastforce
have lhs:  $(\sum x \in \# \text{add-mset } y \ A. x) = (\sum x \in \# A. x) + y$  by simp
have  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
 $(y * \text{count } (\text{add-mset } y \ A) \ y) + (\sum x \in \text{set-mset } A. x * \text{count } A \ x)$ 
  using x-val rhs by (metis (no-types, lifting) False finite-set-mset sum.cong
sum.insert)
  then have  $(\sum x \in \text{set-mset } (\text{add-mset } y \ A). x * \text{count } (\text{add-mset } y \ A) \ x) =$ 
 $y + (\sum x \in \text{set-mset } A. x * \text{count } A \ x)$ 
  using y-count by simp
  then show ?thesis using lhs add.IH by linarith
qed
qed

```

### 1.3 Partitions on Multisets

A partition on a multiset A is a multiset of multisets, where the sum over P equals A and the empty multiset is not in the partition. Based off set partition definition. We note that unlike set partitions, there is no requirement for elements in the multisets to be distinct due to the definition of union on multisets [1]

**lemma** *mset-size-partition-dep*:  $\text{size } \{\# a \in \# A . P \ a \ \vee \ Q \ a \ \#\} =$   
 $\text{size } \{\# a \in \# A . P \ a \ \#\} + \text{size } \{\# a \in \# A . Q \ a \ \#\} - \text{size } \{\# a \in \# A .$   
 $P \ a \ \wedge \ Q \ a \ \#\}$   
 by (simp add: mset-bunion-filter mset-inter-filter mset-union-size-inter)

**definition** *partition-on-mset* :: 'a multiset  $\implies$  'a multiset multiset  $\implies$  bool **where**

*partition-on-mset*  $A P \longleftrightarrow \sum \#P = A \wedge \{\#\} \notin \# P$

**lemma** *partition-on-msetI* [intro]:  $\sum \#P = A \implies \{\#\} \notin \# P \implies \text{partition-on-mset } A P$   
**by** (*simp add: partition-on-mset-def*)

**lemma** *partition-on-msetD1*:  $\text{partition-on-mset } A P \implies \sum \#P = A$   
**by** (*simp add: partition-on-mset-def*)

**lemma** *partition-on-msetD2*:  $\text{partition-on-mset } A P \implies \{\#\} \notin \# P$   
**by** (*simp add: partition-on-mset-def*)

**lemma** *partition-on-mset-empty*:  $\text{partition-on-mset } \{\#\} P \longleftrightarrow P = \{\#\}$   
**unfolding** *partition-on-mset-def*  
**using** *multiset-nonemptyE* **by** *fastforce*

**lemma** *partition-on-mset-all*:  $A \neq \{\#\} \implies \text{partition-on-mset } A \{\#A \#\}$   
**by** (*simp add: partition-on-mset-def*)

**lemma** *partition-on-mset-singletons*:  $\text{partition-on-mset } A (\text{image-mset } (\lambda x . \{\#x\#}) A)$   
**by** (*auto simp: partition-on-mset-def*)

**lemma** *partition-on-mset-not-empty*:  $A \neq \{\#\} \implies \text{partition-on-mset } A P \implies P \neq \{\#\}$   
**by** (*auto simp: partition-on-mset-def*)

**lemma** *partition-on-msetI2*:  $\sum \#P = A \implies (\bigwedge p . p \in \# P \implies p \neq \{\#\}) \implies \text{partition-on-mset } A P$   
**by** (*auto simp: partition-on-mset-def*)

**lemma** *partition-on-mset-elems*:  $\text{partition-on-mset } A P \implies p1 \in \# P \implies x \in \# p1 \implies x \in \# A$   
**by** (*auto simp: partition-on-mset-def*)

**lemma** *partition-on-mset-sum-size-eq*:  $\text{partition-on-mset } A P \implies (\sum x \in \# P . \text{size } x) = \text{size } A$   
**by** (*metis partition-on-msetD1 size-big-union-sum*)

**lemma** *partition-on-mset-card*: **assumes**  $\text{partition-on-mset } A P$  **shows**  $\text{size } P \leq \text{size } A$   
**proof** (*rule ccontr*)  
**assume**  $\neg \text{size } P \leq \text{size } A$   
**then have**  $a: \text{size } P > \text{size } A$  **by** *simp*  
**have**  $\bigwedge x . x \in \# P \implies \text{size } x > 0$  **using** *partition-on-msetD2*  
**using** *assms nonempty-has-size* **by** *auto*  
**then have**  $(\sum x \in \# P . \text{size } x) \geq \text{size } P$   
**by** (*metis leI less-one not-less-zero size-eq-sum-mset sum-mset-mono*)  
**thus** *False* **using** *a partition-on-mset-sum-size-eq*

**using** *assms* **by** *fastforce*  
**qed**

**lemma** *partition-on-mset-count-eq*: *partition-on-mset*  $A P \implies a \in\# A \implies$   
 $(\sum x \in\# P. \text{count } x a) = \text{count } A a$   
**by** (*metis count-sum-mset partition-on-msetD1*)

**lemma** *partition-on-mset-subsets*: *partition-on-mset*  $A P \implies x \in\# P \implies x \subseteq\#$   
 $A$   
**by** (*auto simp add: partition-on-mset-def*)

**lemma** *partition-on-mset-distinct*:  
**assumes** *partition-on-mset*  $A P$   
**assumes** *distinct-mset*  $A$   
**shows** *distinct-mset*  $P$   
**proof** (*rule ccontr*)  
**assume**  $\neg$  *distinct-mset*  $P$   
**then obtain**  $p1$  **where** *count*: *count*  $P p1 \geq 2$   
**by** (*metis Suc-1 distinct-mset-count-less-1 less-Suc-eq-le not-less-eq*)  
**then have** *cge*:  $\bigwedge x . x \in\# p1 \implies (\sum p \in\# P. \text{count } p x) \geq 2$   
**by** (*smt (verit) count-greater-eq-one-iff count-sum-mset-if-1-0 dual-order.trans sum-mset-mono zero-le*)  
**have** *elem-in*:  $\bigwedge x . x \in\# p1 \implies x \in\# A$  **using** *partition-on-mset-elems*  
**by** (*metis count assms(1) count-eq-zero-iff not-numeral-le-zero*)  
**have**  $\bigwedge x . x \in\# A \implies \text{count } A x = 1$  **using** *assms*  
**by** (*simp add: distinct-mset-def*)  
**thus** *False*  
**using** *assms partition-on-mset-count-eq cge elem-in count-inI local.count multiset-nonemptyE*  
**by** (*metis (mono-tags) not-numeral-le-zero numeral-One numeral-le-iff partition-on-mset-def semiring-norm(69)*)  
**qed**

**lemma** *partition-on-mset-distinct-disjoint*:  
**assumes** *partition-on-mset*  $A P$   
**assumes** *distinct-mset*  $A$   
**assumes**  $p1 \in\# P$   
**assumes**  $p2 \in\# P - \{\#p1\#}$   
**shows**  $p1 \cap\# p2 = \{\#\}$   
**using** *Diff-eq-empty-iff-mset assms diff-add-zero distinct-mset-add multiset-inter-assoc sum-mset.remove*  
**by** (*smt (verit) partition-on-msetD1 subset-mset.inf.absorb-iff2 subset-mset.le-add-same-cancel1 subset-mset.le-iff-inf*)

**lemma** *partition-on-mset-diff*:  
**assumes** *partition-on-mset*  $A P$   
**assumes**  $Q \subseteq\# P$   
**shows** *partition-on-mset*  $(A - \sum\# Q) (P - Q)$   
**using** *assms partition-on-mset-def*

by (smt (verit) diff-union-cancelL subset-mset.add-diff-inverse sum-mset.union union-iff)

**lemma** *sigma-over-set-partition-count*:

assumes *finite A*  
 assumes *partition-on A P*  
 assumes  $x \in \# \sum \# (mset-set (mset-set ' P))$   
 shows  $count (\sum \# (mset-set (mset-set ' P))) x = 1$   
**proof** –  
 have *disj: disjoint P* using *assms partition-onD2* by *auto*  
 then obtain *p* where *pin: p ∈ # mset-set (mset-set ' P)* and *xin: x ∈ # p*  
 using *assms* by *blast*  
 then have  $count (mset-set (mset-set ' P)) p = 1$   
 by (*meson count-eq-zero-iff count-mset-set'*)  
 then have *filter:  $\bigwedge p'. p' \in \# ((mset-set (mset-set ' P)) - \{\#p\}) \implies p \neq p'$*   
 using *count-eq-zero-iff count-single* by *fastforce*  
 have *zero:  $\bigwedge p'. p' \in \# mset-set (mset-set ' P) \implies p' \neq p \implies count p' x = 0$*   
**proof** (*rule ccontr*)  
 fix *p'*  
 assume *assm: p' ∈ # mset-set (mset-set ' P)* and *ne: p' ≠ p* and *n0: count p' x ≠ 0*  
 then have *xin2: x ∈ # p'* by *auto*  
 obtain *p1 p2* where *p1in: p1 ∈ P* and *p2in: p2 ∈ P* and *p1eq: mset-set p1 = p*  
 and *p2eq: mset-set p2 = p'* using *assm assms(1) assms(2) pin*  
 by (*metis (no-types, lifting) elem-mset-set finite-elements finite-imageI image-iff*)  
 have *origne: p1 ≠ p2* using *ne p1eq p2eq* by *auto*  
 have *p1 = p2* using *partition-onD4 xin xin2*  
 by (*metis assms(2) count-eq-zero-iff count-mset-set' p1eq p1in p2eq p2in*)  
 then show *False* using *origne* by *simp*  
**qed**  
 have *one: count p x = 1* using *pin xin assms count-eq-zero-iff count-greater-eq-one-iff*  
 by (*metis count-mset-set(3) count-mset-set-le-one image-iff le-antisym*)  
 then have  $count (\sum \# (mset-set (mset-set ' P))) x =$   
 $(\sum p' \in \# (mset-set (mset-set ' P)) . count p' x)$   
 using *count-sum-mset* by *auto*  
 also have  $\dots = (count p x) + (\sum p' \in \# ((mset-set (mset-set ' P)) - \{\#p\}) .$   
*count p' x)*  
 by (*metis (mono-tags, lifting) insert-DiffM pin sum-mset.insert*)  
 also have  $\dots = 1 + (\sum p' \in \# ((mset-set (mset-set ' P)) - \{\#p\}) . count p'$   
*x)*  
 using *one* by *presburger*  
 finally have  $count (\sum \# (mset-set (mset-set ' P))) x =$   
 $1 + (\sum p' \in \# ((mset-set (mset-set ' P)) - \{\#p\}) . 0)$   
 using *zero filter* by (*metis (mono-tags, lifting) in-diffD sum-over-fun-eq*)  
 then show  $count (\sum \# (mset-set (mset-set ' P))) x = 1$  by *simp*  
**qed**

```

lemma partition-on-mset-set:
  assumes finite A
  assumes partition-on A P
  shows partition-on-mset (mset-set A) (mset-set (image (λ x. mset-set x) P))
proof (intro partition-on-msetI)
  have partd1: ∪ P = A using assms partition-onD1 by auto
  have imp: ∧x. x ∈# ∑# (mset-set (mset-set ' P)) ⇒ x ∈# mset-set A
  proof -
    fix x
    assume x ∈# ∑# (mset-set (mset-set ' P))
    then obtain p where p ∈ (mset-set ' P) and xin: x ∈# p
    by (metis elem-mset-set equals0D infinite-set-mset-mset-set mset-big-union-obtain)

    then have set-mset p ∈ P
    by (metis empty-iff finite-set-mset-mset-set image-iff infinite-set-mset-mset-set)

    then show x ∈# mset-set A
      using partd1 xin assms(1) by auto
  qed
  have imp2: ∧x. x ∈# mset-set A ⇒ x ∈# ∑# (mset-set (mset-set ' P))
  proof -
    fix x
    assume x ∈# mset-set A
    then have x ∈ A by (simp add: assms(1))
    then obtain p where p ∈ P and x ∈ p using assms(2) using partd1 by
blast
    then obtain p' where p' ∈ (mset-set ' P) and p' = mset-set p by blast
    thus x ∈# ∑# (mset-set (mset-set ' P)) using assms ⟨p ∈ P⟩ ⟨x ∈ p⟩
finite-elements partd1
    by (metis Sup-upper finite-imageI finite-set-mset-mset-set in-Union-mset-iff
rev-finite-subset)
  qed
  have a1: ∧ x. x ∈# mset-set A ⇒ count (mset-set A) x = 1
    using assms(1) by fastforce
  then show ∑# (mset-set (mset-set ' P)) = mset-set A using imp imp2 a1
    by (metis assms(1) assms(2) count-eq-zero-iff multiset-eqI sigma-over-set-partition-count)
  have ∧ p. p ∈ P ⇒ p ≠ {} using assms partition-onD3 by auto
  then have ∧ p. p ∈ P ⇒ mset-set p ≠ {#} using mset-set-empty-iff
    by (metis Union-upper assms(1) partd1 rev-finite-subset)
  then show {#} ∉# mset-set (mset-set ' P)
    by (metis elem-mset-set equals0D image-iff infinite-set-mset-mset-set)
qed

```

```

lemma partition-on-mset-distinct-inter:
  assumes partition-on-mset A P
  assumes distinct-mset A
  assumes p1 ∈# P and p2 ∈# P and p1 ≠ p2
  shows p1 ∩# p2 = {#}
  by (metis assms in-remove1-mset-neq partition-on-mset-distinct-disjoint)

```

**lemma** *partition-on-set-mset-distinct*:  
**assumes** *partition-on-mset A P*  
**assumes** *distinct-mset A*  
**assumes**  $p \in\# \text{image-mset set-mset } P$   
**assumes**  $p' \in\# \text{image-mset set-mset } P$   
**assumes**  $p \neq p'$   
**shows**  $p \cap p' = \{\}$   
**proof** –  
**obtain**  $p1$  **where**  $p1in: p1 \in\# P$  **and**  $p1eq: \text{set-mset } p1 = p$  **using** *assms(3)*  
**by** *blast*  
**obtain**  $p2$  **where**  $p2in: p2 \in\# P$  **and**  $p2eq: \text{set-mset } p2 = p'$  **using** *assms(4)*  
**by** *blast*  
**have** *distinct-mset P* **using** *assms partition-on-mset-distinct* **by** *blast*  
**then have**  $p1 \neq p2$  **using** *assms* **using**  $p1eq$   $p2eq$  **by** *fastforce*  
**then have**  $p1 \cap\# p2 = \{\#\}$  **using** *partition-on-mset-distinct-inter*  
**using** *assms(1) assms(2) p1in p2in* **by** *auto*  
**thus** *?thesis* **using**  $p1eq$   $p2eq$   
**by** (*metis set-mset-empty set-mset-inter*)  
**qed**

**lemma** *partition-on-set-mset*:  
**assumes** *partition-on-mset A P*  
**assumes** *distinct-mset A*  
**shows** *partition-on (set-mset A) (set-mset (image-mset set-mset P))*  
**proof** (*intro partition-onI*)  
**show**  $\bigwedge p. p \in\# \text{image-mset set-mset } P \implies p \neq \{\}$   
**using** *assms(1) partition-on-msetD2* **by** *fastforce*  
**next**  
**have**  $\bigwedge x. x \in \text{set-mset } A \implies x \in \bigcup (\text{set-mset } (\text{image-mset set-mset } P))$   
**by** (*metis Union-iff assms(1) image-eqI mset-big-union-obtain partition-on-msetD1*  
*set-image-mset*)  
**then show**  $\bigcup (\text{set-mset } (\text{image-mset set-mset } P)) = \text{set-mset } A$   
**using** *set-eqI' partition-on-mset-elems assms* **by** *auto*  
**show**  $\bigwedge p p'. p \in\# \text{image-mset set-mset } P \implies p' \in\# \text{image-mset set-mset } P \implies$   

$$p \neq p' \implies p \cap p' = \{\}$$
**using** *partition-on-set-mset-distinct assms* **by** *blast*  
**qed**

**lemma** *partition-on-mset-eq-imp-eq-carrier*:  
**assumes** *partition-on-mset A P*  
**assumes** *partition-on-mset B P*  
**shows**  $A = B$   
**using** *assms partition-on-msetD1* **by** *auto*

**lemma** *partition-on-mset-add-single*:  
**assumes** *partition-on-mset A P*  
**shows** *partition-on-mset (add-mset a A) (add-mset {\#a\#} P)*

```

using assms by (auto simp: partition-on-mset-def)

lemma partition-on-mset-add-part:
  assumes partition-on-mset A P
  assumes  $X \neq \{\#\}$ 
  assumes  $A + X = A'$ 
  shows partition-on-mset A' (add-mset X P)
  using assms by (auto simp: partition-on-mset-def)

lemma partition-on-mset-add:
  assumes partition-on-mset A P
  assumes  $X \in\# P$ 
  assumes add-mset a X = X'
  shows partition-on-mset (add-mset a A) (add-mset X' (P - {\#X\#}))
  using add-mset-add-single assms empty-not-add-mset mset-subset-eq-single partition-on-mset-all
  by (smt (verit) partition-on-mset-def subset-mset.add-diff-inverse sum-mset.add-mset sum-mset.remove union-iff union-mset-add-mset-left)

lemma partition-on-mset-elem-exists-part:
  assumes partition-on-mset A P
  assumes  $x \in\# A$ 
  obtains p where  $p \in\# P$  and  $x \in\# p$ 
  using assms in-Union-mset-iff partition-on-msetD2 partition-on-msetI
  by (metis partition-on-mset-eq-imp-eq-carrier)

lemma partition-on-mset-combine:
  assumes partition-on-mset A P
  assumes partition-on-mset B Q
  shows partition-on-mset (A + B) (P + Q)
  unfolding partition-on-mset-def
  using assms partition-on-msetD1 partition-on-msetD2 by auto

lemma partition-on-mset-split:
  assumes partition-on-mset A (P + Q)
  shows partition-on-mset ( $\sum \#P$ ) P
  using partition-on-mset-def partition-on-msetD2 assms by fastforce
end
theory Design-Basics imports Main Multisets-Extras HOL-Library.Disjoint-Sets
begin

```

## 2 Design Theory Basics

All definitions in this section reference the handbook of combinatorial designs [3]

## 2.1 Initial setup

Enable coercion of nats to ints to aid with reasoning on design properties

```
declare [[coercion-enabled]]
declare [[coercion of-nat :: nat  $\Rightarrow$  int]]
```

## 2.2 Incidence System

An incidence system is defined to be a wellformed set system. i.e. each block is a subset of the base point set. Alternatively, an incidence system can be looked at as the point set and an incidence relation which indicates if they are in the same block

```
locale incidence-system =
  fixes point-set :: 'a set ( $\langle \mathcal{V} \rangle$ )
  fixes block-collection :: 'a set multiset ( $\langle \mathcal{B} \rangle$ )
  assumes wellformed:  $b \in \# \mathcal{B} \implies b \subseteq \mathcal{V}$ 
begin
```

```
definition  $\mathcal{I} \equiv \{ (x, b) . b \in \# \mathcal{B} \wedge x \in b \}$ 
```

```
definition incident :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  bool where
incident  $p$   $b \equiv (p, b) \in \mathcal{I}$ 
```

Defines common notation used to indicate number of points ( $v$ ) and number of blocks ( $b$ )

```
abbreviation  $v \equiv \text{card } \mathcal{V}$ 
```

```
abbreviation  $b \equiv \text{size } \mathcal{B}$ 
```

Basic incidence lemmas

```
lemma incidence-alt-def:
  assumes  $p \in \mathcal{V}$ 
  assumes  $b \in \# \mathcal{B}$ 
  shows incident  $p$   $b \iff p \in b$ 
  by (auto simp add: incident-def I-def assms)
```

```
lemma wf-invalid-point:  $x \notin \mathcal{V} \implies b \in \# \mathcal{B} \implies x \notin b$ 
using wellformed by auto
```

```
lemma block-set-nempty-imp-block-ex:  $\mathcal{B} \neq \{\#\} \implies \exists bl . bl \in \# \mathcal{B}$ 
by auto
```

Abbreviations for all incidence systems

```
abbreviation multiplicity :: 'a set  $\Rightarrow$  nat where
multiplicity  $b \equiv \text{count } \mathcal{B} b$ 
```

```
abbreviation incomplete-block :: 'a set  $\Rightarrow$  bool where
incomplete-block  $bl \equiv \text{card } bl < \text{card } \mathcal{V} \wedge bl \in \# \mathcal{B}$ 
```

**lemma** *incomplete-alt-size*: *incomplete-block*  $bl \implies \text{card } bl < v$   
**by** *simp*

**lemma** *incomplete-alt-in*: *incomplete-block*  $bl \implies bl \in\# \mathcal{B}$   
**by** *simp*

**lemma** *incomplete-alt-imp[intro]*:  $\text{card } bl < v \implies bl \in\# \mathcal{B} \implies \text{incomplete-block } bl$   
**by** *simp*

**definition** *design-support* :: 'a set set **where**  
*design-support*  $\equiv \text{set-mset } \mathcal{B}$

**end**

## 2.3 Finite Incidence Systems

These simply require the point set to be finite. As multisets are only defined to be finite, it is implied that the block set must be finite already

**locale** *finite-incidence-system* = *incidence-system* +  
**assumes** *finite-sets*: *finite*  $\mathcal{V}$   
**begin**

**lemma** *finite-blocks*:  $b \in\# \mathcal{B} \implies \text{finite } b$   
**using** *wellformed finite-sets finite-subset* **by** *blast*

**lemma** *mset-points-distinct*: *distinct-mset* (*mset-set*  $\mathcal{V}$ )  
**using** *finite-sets* **by** (*simp add: distinct-mset-def*)

**lemma** *mset-points-distinct-diff-one*: *distinct-mset* (*mset-set* ( $\mathcal{V} - \{x\}$ ))  
**by** (*meson count-mset-set-le-one distinct-mset-count-less-1*)

**lemma** *finite-design-support*: *finite* (*design-support*)  
**using** *design-support-def* **by** *auto*

**lemma** *block-size-lt-order*:  $bl \in\# \mathcal{B} \implies \text{card } bl \leq \text{card } \mathcal{V}$   
**using** *wellformed* **by** (*simp add: card-mono finite-sets*)

**end**

## 2.4 Designs

There are many varied definitions of a design in literature. However, the most commonly accepted definition is a finite point set,  $V$  and collection of blocks  $B$ , where no block in  $B$  can be empty

**locale** *design* = *finite-incidence-system* +  
**assumes** *blocks-nempty*:  $bl \in\# \mathcal{B} \implies bl \neq \{\}$

**begin**

**lemma** *wf-design*:  $design \mathcal{V} \mathcal{B}$  **by** *intro-locales*

**lemma** *wf-design-iff*:  $bl \in \# \mathcal{B} \implies design \mathcal{V} \mathcal{B} \iff (bl \subseteq \mathcal{V} \wedge finite \mathcal{V} \wedge bl \neq \{\})$   
**using** *blocks-nempty wellformed finite-sets*  
**by** (*simp add: wf-design*)

Reasoning on non empty properties and non zero parameters

**lemma** *blocks-nempty-alt*:  $\forall bl \in \# \mathcal{B}. bl \neq \{\}$   
**using** *blocks-nempty* **by** *auto*

**lemma** *block-set-nempty-imp-points*:  $\mathcal{B} \neq \{\#\} \implies \mathcal{V} \neq \{\}$   
**using** *wf-design wf-design-iff* **by** *auto*

**lemma** *b-non-zero-imp-v-non-zero*:  $b > 0 \implies v > 0$   
**using** *block-set-nempty-imp-points finite-sets* **by** *fastforce*

**lemma** *v-eq0-imp-b-eq0*:  $v = 0 \implies b = 0$   
**using** *b-non-zero-imp-v-non-zero* **by** *auto*

Size lemmas

**lemma** *block-size-lt-v*:  $bl \in \# \mathcal{B} \implies card \ bl \leq v$   
**by** (*simp add: card-mono finite-sets wellformed*)

**lemma** *block-size-gt-0*:  $bl \in \# \mathcal{B} \implies card \ bl > 0$   
**using** *finite-sets blocks-nempty finite-blocks* **by** *fastforce*

**lemma** *design-cart-product-size*:  $size \ ((mset-set \ \mathcal{V}) \times \# \ \mathcal{B}) = v * b$   
**by** (*simp add: size-cartesian-product*)

**end**

Intro rules for design locale

**lemma** *wf-design-implies*:  
**assumes**  $(\bigwedge b . b \in \# \mathcal{B} \implies b \subseteq V)$   
**assumes**  $\bigwedge b . b \in \# \mathcal{B} \implies b \neq \{\}$   
**assumes** *finite V*  
**assumes**  $\mathcal{B} \neq \{\#\}$   
**assumes**  $V \neq \{\}$   
**shows** *design V B*  
**using** *assms* **by** (*unfold-locales simp-all*)

**lemma** (**in** *incidence-system*) *finite-sysI[intro]*:  $finite \ \mathcal{V} \implies finite-incidence-system \ \mathcal{V} \ \mathcal{B}$   
**by** (*unfold-locales simp-all*)

**lemma** (**in** *finite-incidence-system*) *designI[intro]*:  $(\bigwedge b . b \in \# \mathcal{B} \implies b \neq \{\}) \implies \mathcal{B} \neq \{\#\}$

$\implies \mathcal{V} \neq \{\} \implies \text{design } \mathcal{V} \mathcal{B}$   
**by** (*unfold-locales*) *simp-all*

## 2.5 Core Property Definitions

### 2.5.1 Replication Number

The replication number for a point is the number of blocks that point is incident with

**definition** *point-replication-number* :: 'a set multiset  $\Rightarrow$  'a  $\Rightarrow$  nat (**infix** <rep> 75)  
**where**  
 $B \text{ rep } x \equiv \text{size } \{\#b \in \# B . x \in b\# \}$

**lemma** *max-point-rep*:  $B \text{ rep } x \leq \text{size } B$   
**using** *size-filter-mset-lesseq* **by** (*simp add: point-replication-number-def*)

**lemma** *rep-number-g0-exists*:  
**assumes**  $B \text{ rep } x > 0$   
**obtains**  $b$  **where**  $b \in \# B$  **and**  $x \in b$   
**proof** –  
**have**  $\text{size } \{\#b \in \# B . x \in b\# \} > 0$  **using** *assms point-replication-number-def*  
**by** *metis*  
**thus** *?thesis*  
**by** (*metis filter-mset-empty-conv nonempty-has-size that*)  
**qed**

**lemma** *rep-number-on-set-def*:  $\text{finite } B \implies (\text{mset-set } B) \text{ rep } x = \text{card } \{b \in B . x \in b\}$   
**by** (*simp add: point-replication-number-def*)

**lemma** *point-rep-number-split*[*simp*]:  $(A + B) \text{ rep } x = A \text{ rep } x + B \text{ rep } x$   
**by** (*simp add: point-replication-number-def*)

**lemma** *point-rep-singleton-val* [*simp*]:  $x \in b \implies \{\#b\# \} \text{ rep } x = 1$   
**by** (*simp add: point-replication-number-def*)

**lemma** *point-rep-singleton-ival* [*simp*]:  $x \notin b \implies \{\#b\# \} \text{ rep } x = 0$   
**by** (*simp add: point-replication-number-def*)

**context** *incidence-system*  
**begin**

**lemma** *point-rep-number-alt-def*:  $\mathcal{B} \text{ rep } x = \text{size } \{\# b \in \# \mathcal{B} . x \in b\# \}$   
**by** (*simp add: point-replication-number-def*)

**lemma** *rep-number-non-zero-system-point*:  $\mathcal{B} \text{ rep } x > 0 \implies x \in \mathcal{V}$   
**using** *rep-number-g0-exists wellformed*  
**by** (*metis wf-invalid-point*)

**lemma** *point-rep-non-existance* [simp]:  $x \notin \mathcal{V} \implies \mathcal{B} \text{ rep } x = 0$   
**using** *wf-invalid-point* **by** (*simp add: point-replication-number-def filter-mset-empty-conv*)

**lemma** *point-rep-number-inv*:  $\text{size } \{\# b \in \# \mathcal{B} . x \notin b \#\} = b - (\mathcal{B} \text{ rep } x)$

**proof** –

**have**  $b = \text{size } \{\# b \in \# \mathcal{B} . x \notin b \#\} + \text{size } \{\# b \in \# \mathcal{B} . x \in b \#\}$

**using** *multiset-partition* **by** (*metis add.commute size-union*)

**thus** *?thesis* **by** (*simp add: point-replication-number-def*)

**qed**

**lemma** *point-rep-num-inv-non-empty*:  $(\mathcal{B} \text{ rep } x) < b \implies \mathcal{B} \neq \{\#\} \implies \{\# b \in \# \mathcal{B} . x \notin b \#\} \neq \{\#\}$

**by** (*metis diff-zero point-replication-number-def size-empty size-filter-neg verit-comp-simplify1 (1)*)

**end**

## 2.5.2 Point Index

The point index of a subset of points in a design, is the number of times those points occur together in a block of the design

**definition** *points-index* :: 'a set multiset  $\Rightarrow$  'a set  $\Rightarrow$  nat (**infix**  $\langle \text{index} \rangle$  75) **where**  
 $B \text{ index } ps \equiv \text{size } \{\# b \in \# B . ps \subseteq b \#\}$

**lemma** *points-index-empty* [simp]:  $\{\#\} \text{ index } ps = 0$

**by** (*simp add: points-index-def*)

**lemma** *point-index-distrib*:  $(B1 + B2) \text{ index } ps = B1 \text{ index } ps + B2 \text{ index } ps$

**by** (*simp add: points-index-def*)

**lemma** *point-index-diff*:  $B1 \text{ index } ps = (B1 + B2) \text{ index } ps - B2 \text{ index } ps$

**by** (*simp add: points-index-def*)

**lemma** *points-index-singleton*:  $\{\# b \#\} \text{ index } ps = 1 \iff ps \subseteq b$

**by** (*simp add: points-index-def*)

**lemma** *points-index-singleton-zero*:  $\neg (ps \subseteq b) \implies \{\# b \#\} \text{ index } ps = 0$

**by** (*simp add: points-index-def*)

**lemma** *points-index-sum*:  $(\sum \# B) \text{ index } ps = (\sum b \in \# B . (b \text{ index } ps))$

**using** *points-index-empty* **by** (*induction B*) (*auto simp add: point-index-distrib*)

**lemma** *points-index-block-image-add-eq*:

**assumes**  $x \notin ps$

**assumes**  $B \text{ index } ps = l$

**shows**  $\{\# \text{insert } x b . b \in \# B \#\} \text{ index } ps = l$

**using** *points-index-def* **by** (*metis (no-types, lifting) assms filter-mset-cong image-mset-filter-swap2 points-index-def size-image-mset subset-insert*)

**lemma** *points-index-on-set-def* [*simp*]:  
**assumes** *finite B*  
**shows**  $(\text{mset-set } B) \text{ index } ps = \text{card } \{b \in B. ps \subseteq b\}$   
**by** (*simp add: points-index-def assms*)

**lemma** *points-index-single-rep-num*:  $B \text{ index } \{x\} = B \text{ rep } x$   
**by** (*simp add: points-index-def point-replication-number-def*)

**lemma** *points-index-pair-rep-num*:  
**assumes**  $\bigwedge b. b \in\# B \implies x \in b$   
**shows**  $B \text{ index } \{x, y\} = B \text{ rep } y$   
**using** *point-replication-number-def points-index-def*  
**by** (*metis assms empty-subsetI filter-mset-cong insert-subset*)

**lemma** *points-index-0-left-imp*:  
**assumes**  $B \text{ index } ps = 0$   
**assumes**  $b \in\# B$   
**shows**  $\neg (ps \subseteq b)$   
**proof** (*rule ccontr*)  
**assume**  $\neg \neg ps \subseteq b$   
**then have**  $a: ps \subseteq b$  **by** *auto*  
**then have**  $b \in\# \{\#bl \in\# B . ps \subseteq bl\# \}$  **by** (*simp add: assms(2)*)  
**thus** *False* **by** (*metis assms(1) count-greater-eq-Suc-zero-iff count-size-set-repr not-less-eq-eq points-index-def size-filter-mset-lesseq*)

**qed**

**lemma** *points-index-0-right-imp*:  
**assumes**  $\bigwedge b . b \in\# B \implies (\neg ps \subseteq b)$   
**shows**  $B \text{ index } ps = 0$   
**using** *assms* **by** (*simp add: filter-mset-empty-conv points-index-def*)

**lemma** *points-index-0-iff*:  $B \text{ index } ps = 0 \iff (\forall b. b \in\# B \longrightarrow (\neg ps \subseteq b))$   
**using** *points-index-0-left-imp points-index-0-right-imp* **by** *metis*

**lemma** *points-index-gt0-impl-existance*:  
**assumes**  $B \text{ index } ps > 0$   
**shows**  $(\exists bl . (bl \in\# B \wedge ps \subseteq bl))$   
**proof** –  
**have**  $\text{size } \{\#bl \in\# B . ps \subseteq bl\# \} > 0$   
**by** (*metis assms points-index-def*)  
**then obtain** *bl* **where**  $bl \in\# B$  **and**  $ps \subseteq bl$   
**by** (*metis filter-mset-empty-conv nonempty-has-size*)  
**thus** *?thesis* **by** *auto*

**qed**

**lemma** *points-index-one-unique*:  
**assumes**  $B \text{ index } ps = 1$   
**assumes**  $bl \in\# B$  **and**  $ps \subseteq bl$  **and**  $bl' \in\# B$  **and**  $ps \subseteq bl'$

**shows**  $bl = bl'$   
**proof** (*rule ccontr*)  
**assume**  $assm: bl \neq bl'$   
**then have**  $bl1: bl \in \# \{ \#bl \in \# B . ps \subseteq bl \# \}$  **using** *assms* **by** *simp*  
**then have**  $bl2: bl' \in \# \{ \#bl \in \# B . ps \subseteq bl \# \}$  **using** *assms* **by** *simp*  
**then have**  $\{ \#bl, bl' \# \} \subseteq \# \{ \#bl \in \# B . ps \subseteq bl \# \}$  **using** *assms* **by** (*metis bl1 bl2 points-index-def*)  
 $add-mset-subseteq-single-iff\ assm\ mset-subset-eq-single\ size-single\ subseteq-mset-size-eql$   
  
**then have**  $size \{ \#bl \in \# B . ps \subseteq bl \# \} \geq 2$  **using** *size-mset-mono* **by** *fastforce*  
  
**thus** *False* **using** *assms* **by** (*metis numeral-le-one-iff points-index-def semiring-norm(69)*)  
**qed**

**lemma** *points-index-one-unique-block*:  
**assumes**  $B\ index\ ps = 1$   
**shows**  $\exists! bl . (bl \in \# B \wedge ps \subseteq bl)$   
**using** *assms points-index-gt0-impl-existence points-index-one-unique*  
**by** (*metis zero-less-one*)

**lemma** *points-index-one-not-unique-block*:  
**assumes**  $B\ index\ ps = 1$   
**assumes**  $ps \subseteq bl$   
**assumes**  $bl \in \# B$   
**assumes**  $bl' \in \# B - \{ \#bl \# \}$   
**shows**  $\neg ps \subseteq bl'$   
**proof** –  
**have**  $B = (B - \{ \#bl \# \}) + \{ \#bl \# \}$  **by** (*simp add: assms(3)*)  
**then have**  $(B - \{ \#bl \# \})\ index\ ps = B\ index\ ps - \{ \#bl \# \}\ index\ ps$   
**by** (*metis point-index-diff*)  
**then have**  $(B - \{ \#bl \# \})\ index\ ps = 0$  **using** *assms points-index-singleton*  
**by** (*metis diff-self-eq-0*)  
**thus** *?thesis* **using** *assms(4) points-index-0-left-imp* **by** *auto*  
**qed**

**lemma** (*in incidence-system*) *points-index-alt-def*:  $\mathcal{B}\ index\ ps = size \{ \#b \in \# \mathcal{B} . ps \subseteq b \# \}$   
**by** (*simp add: points-index-def*)

**lemma** (*in incidence-system*) *points-index-ps-nin*:  $\neg (ps \subseteq \mathcal{V}) \implies \mathcal{B}\ index\ ps = 0$   
**using** *points-index-alt-def filter-mset-empty-conv in-mono size-empty subsetI wf-invalid-point*  
**by** *metis*

**lemma** (*in incidence-system*) *points-index-count-bl*:  
 $multiplicity\ bl \geq n \implies ps \subseteq bl \implies count \{ \#bl \in \# \mathcal{B} . ps \subseteq bl \# \} bl \geq n$   
**by** *simp*

**lemma** (*in finite-incidence-system*) *points-index-zero*:

**assumes**  $\text{card } ps > \text{card } \mathcal{V}$   
**shows**  $\mathcal{B} \text{ index } ps = 0$   
**proof** –  
**have**  $\bigwedge b. b \in \# \mathcal{B} \implies \text{card } ps > \text{card } b$   
**using** *block-size-lt-order card-subset-not-gt-card finite-sets assms* **by** *fastforce*  
**then have**  $\{\#b \in \# \mathcal{B} . ps \subseteq b\# \} = \{\#\}$   
**by** (*simp add: card-subset-not-gt-card filter-mset-empty-conv finite-blocks*)  
**thus** *?thesis* **using** *points-index-alt-def* **by** *simp*  
**qed**

**lemma** (*in design*) *points-index-subset*:  
 $x \subseteq \# \{\#bl \in \# \mathcal{B} . ps \subseteq bl\# \} \implies ps \subseteq \mathcal{V} \implies (\mathcal{B} \text{ index } ps) \geq (\text{size } x)$   
**by** (*simp add: points-index-def size-mset-mono*)

**lemma** (*in design*) *points-index-count-min: multiplicity*  $bl \geq n \implies ps \subseteq bl \implies \mathcal{B} \text{ index } ps \geq n$   
**using** *points-index-alt-def set-count-size-min* **by** (*metis filter-mset.rep-eq*)

### 2.5.3 Intersection Number

The intersection number of two blocks is the size of the intersection of those blocks. i.e. the number of points which occur in both blocks

**definition** *intersection-number* :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  nat (**infix**  $\langle |\cap| \rangle$  70) **where**  
 $b1 \text{ } |\cap| \text{ } b2 \equiv \text{card } (b1 \cap b2)$

**lemma** *intersection-num-non-neg*:  $b1 \text{ } |\cap| \text{ } b2 \geq 0$   
**by** (*simp add: intersection-number-def*)

**lemma** *intersection-number-empty-iff*:  
**assumes** *finite*  $b1$   
**shows**  $b1 \cap b2 = \{\} \iff b1 \text{ } |\cap| \text{ } b2 = 0$   
**by** (*simp add: intersection-number-def assms*)

**lemma** *intersect-num-commute*:  $b1 \text{ } |\cap| \text{ } b2 = b2 \text{ } |\cap| \text{ } b1$   
**by** (*simp add: inf-commute intersection-number-def*)

**definition** *n-intersect-number* :: 'a set  $\Rightarrow$  nat  $\Rightarrow$  'a set  $\Rightarrow$  nat **where**  
 $n\text{-intersect-number } b1 \ n \ b2 \equiv \text{card } \{ x \in \text{Pow } (b1 \cap b2) . \text{card } x = n \}$

**notation** *n-intersect-number* ( $\langle (- \text{ } |\cap| \text{ } -) \rangle$  [52, 51, 52] 50)

**lemma** *n-intersect-num-subset-def*:  $b1 \text{ } |\cap|_n \text{ } b2 = \text{card } \{ x . x \subseteq b1 \cap b2 \wedge \text{card } x = n \}$   
**using** *n-intersect-number-def* **by** *auto*

**lemma** *n-inter-num-one*:  $\text{finite } b1 \implies \text{finite } b2 \implies b1 \text{ } |\cap|_1 \text{ } b2 = b1 \text{ } |\cap| \text{ } b2$   
**using** *n-intersect-number-def intersection-number-def card-Pow-filter-one*  
**by** (*metis (full-types) finite-Int*)

**lemma** *n-inter-num-choose*:  $finite\ b1 \implies finite\ b2 \implies b1 \mid\cap\mid_n\ b2 = (card\ (b1 \cap b2)\ choose\ n)$

**using** *n-subsets n-intersect-num-subset-def*  
**by** (*metis (full-types) finite-Int*)

**lemma** *set-filter-single*:  $x \in A \implies \{a \in A . a = x\} = \{x\}$   
**by** *auto*

**lemma** (*in design*) *n-inter-num-zero*:

**assumes**  $b1 \in\#\ \mathcal{B}$  **and**  $b2 \in\#\ \mathcal{B}$

**shows**  $b1 \mid\cap\mid_0\ b2 = 1$

**proof** –

**have** *empty*:  $\bigwedge x . finite\ x \implies card\ x = 0 \implies x = \{\}$

**by** *simp*

**have** *empt-in*:  $\{\} \in Pow\ (b1 \cap b2)$  **by** *simp*

**have** *finite*  $(b1 \cap b2)$  **using** *finite-blocks assms* **by** *simp*

**then have**  $\bigwedge x . x \in Pow\ (b1 \cap b2) \implies finite\ x$  **by** (*meson PowD finite-subset*)

**then have**  $\{x \in Pow\ (b1 \cap b2) . card\ x = 0\} = \{x \in Pow\ (b1 \cap b2) . x = \{\}\}$

**using** *empty* **by** (*metis card.empty*)

**then have**  $\{x \in Pow\ (b1 \cap b2) . card\ x = 0\} = \{\{\}\}$

**by** (*simp add: empt-in set-filter-single Collect-conv-if*)

**thus** *?thesis* **by** (*simp add: n-intersect-number-def*)

**qed**

**lemma** (*in design*) *n-inter-num-choose-design*:  $b1 \in\#\ \mathcal{B} \implies b2 \in\#\ \mathcal{B}$

$\implies b1 \mid\cap\mid_n\ b2 = (card\ (b1 \cap b2)\ choose\ n)$

**using** *finite-blocks* **by** (*simp add: n-inter-num-choose*)

**lemma** (*in design*) *n-inter-num-choose-design-inter*:  $b1 \in\#\ \mathcal{B} \implies b2 \in\#\ \mathcal{B}$

$\implies b1 \mid\cap\mid_n\ b2 = (nat\ (b1 \mid\cap\mid\ b2)\ choose\ n)$

**using** *finite-blocks* **by** (*simp add: n-inter-num-choose intersection-number-def*)

## 2.6 Incidence System Set Property Definitions

**context** *incidence-system*

**begin**

The set of replication numbers for all points of design

**definition** *replication-numbers* :: *nat set* **where**

*replication-numbers*  $\equiv \{\mathcal{B}\ rep\ x \mid x . x \in \mathcal{V}\}$

**lemma** *replication-numbers-non-empty*:

**assumes**  $\mathcal{V} \neq \{\}$

**shows** *replication-numbers*  $\neq \{\}$

**by** (*simp add: assms replication-numbers-def*)

**lemma** *obtain-point-with-rep*:  $r \in replication\ numbers \implies \exists x . x \in \mathcal{V} \wedge \mathcal{B}\ rep\ x = r$

**using** *replication-numbers-def* **by** *auto*

**lemma** *point-rep-number-in-set*:  $x \in \mathcal{V} \implies (\mathcal{B} \text{ rep } x) \in \text{replication-numbers}$   
**by** (*auto simp add: replication-numbers-def*)

**lemma** (*in finite-incidence-system*) *replication-numbers-finite*: *finite replication-numbers*  
**using** *finite-sets* **by** (*simp add: replication-numbers-def*)

The set of all block sizes in a system

**definition** *sys-block-sizes* :: *nat set* **where**  
*sys-block-sizes*  $\equiv \{ \text{card } bl \mid bl. bl \in \# \mathcal{B} \}$

**lemma** *block-sizes-non-empty-set*:  
**assumes**  $\mathcal{B} \neq \{\#\}$   
**shows** *sys-block-sizes*  $\neq \{\}$   
**by** (*simp add: sys-block-sizes-def assms*)

**lemma** *finite-block-sizes*: *finite (sys-block-sizes)*  
**by** (*simp add: sys-block-sizes-def*)

**lemma** *block-sizes-non-empty*:  
**assumes**  $\mathcal{B} \neq \{\#\}$   
**shows**  $\text{card } (\text{sys-block-sizes}) > 0$   
**using** *finite-block-sizes block-sizes-non-empty-set*  
**by** (*simp add: assms card-gt-0-iff*)

**lemma** *sys-block-sizes-in*:  $bl \in \# \mathcal{B} \implies \text{card } bl \in \text{sys-block-sizes}$   
**unfolding** *sys-block-sizes-def* **by** *auto*

**lemma** *sys-block-sizes-obtain-bl*:  $x \in \text{sys-block-sizes} \implies (\exists bl \in \# \mathcal{B}. \text{card } bl = x)$   
**by** (*auto simp add: sys-block-sizes-def*)

The set of all possible intersection numbers in a system.

**definition** *intersection-numbers* :: *nat set* **where**  
*intersection-numbers*  $\equiv \{ b1 \mid \cap \mid b2 \mid b1 \ b2 . b1 \in \# \mathcal{B} \wedge b2 \in \# (\mathcal{B} - \{\#b1\#\}) \}$

**lemma** *obtain-blocks-intersect-num*:  $n \in \text{intersection-numbers} \implies$   
 $\exists b1 \ b2. b1 \in \# \mathcal{B} \wedge b2 \in \# (\mathcal{B} - \{\#b1\#\}) \wedge b1 \mid \cap \mid b2 = n$   
**by** (*auto simp add: intersection-numbers-def*)

**lemma** *intersect-num-in-set*:  $b1 \in \# \mathcal{B} \implies b2 \in \# (\mathcal{B} - \{\#b1\#\}) \implies b1 \mid \cap \mid b2$   
 $\in \text{intersection-numbers}$   
**by** (*auto simp add: intersection-numbers-def*)

The set of all possible point indices

**definition** *point-indices* :: *nat*  $\Rightarrow$  *nat set* **where**  
*point-indices*  $t \equiv \{ \mathcal{B} \text{ index } ps \mid ps. \text{card } ps = t \wedge ps \subseteq \mathcal{V} \}$

**lemma** *point-indices-elem-in*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps \in \text{point-indices } t$

**by** (*auto simp add: point-indices-def*)

**lemma** *point-indices-alt-def*: *point-indices*  $t = \{ \mathcal{B} \text{ index } ps \mid ps. \text{card } ps = t \wedge ps \subseteq \mathcal{V} \}$

**by** (*simp add: point-indices-def*)

**end**

## 2.7 Basic Constructions on designs

This section defines some of the most common universal constructions found in design theory involving only a single design

### 2.7.1 Design Complements

**context** *incidence-system*

**begin**

The complement of a block are all the points in the design not in that block. The complement of a design is therefore the original point sets, and set of all block complements

**definition** *block-complement*:: '*a set*  $\Rightarrow$  '*a set* ( $\langle \cdot^c \rangle$  [56] 55) **where** *block-complement*  $b \equiv \mathcal{V} - b$

**definition** *complement-blocks* :: '*a set multiset* ( $\langle (\mathcal{B}^C) \rangle$ ) **where** *complement-blocks*  $\equiv \{ \# \text{ } bl^c . bl \in \# \mathcal{B} \# \}$

**lemma** *block-complement-elem-iff*:

**assumes**  $ps \subseteq \mathcal{V}$

**shows**  $ps \subseteq bl^c \iff (\forall x \in ps. x \notin bl)$

**using** *assms block-complement-def* **by** (*auto*)

**lemma** *block-complement-inter-empty*:  $bl1^c = bl2 \implies bl1 \cap bl2 = \{ \}$

**using** *block-complement-def* **by** *auto*

**lemma** *block-complement-inv*:

**assumes**  $bl \in \# \mathcal{B}$

**assumes**  $bl^c = bl2$

**shows**  $bl2^c = bl$

**by** (*metis Diff-Diff-Int assms(1) assms(2) block-complement-def inf.absorb-iff2 wellformed*)

**lemma** *block-complement-subset-points*:  $ps \subseteq (bl^c) \implies ps \subseteq \mathcal{V}$

**using** *block-complement-def* **by** *blast*

**lemma** *obtain-comp-block-orig*:

**assumes**  $bl1 \in \# \mathcal{B}^C$

**obtains**  $bl2$  **where**  $bl2 \in \# \mathcal{B}$  **and**  $bl1 = bl2^c$

**using** *wellformed assms* **by** (*auto simp add: complement-blocks-def*)

**lemma** *complement-same-b* [simp]:  $\text{size } \mathcal{B}^C = \text{size } \mathcal{B}$   
**by** (*simp add: complement-blocks-def*)

**lemma** *block-comp-elem-alt-left*:  $x \in \text{bl} \implies \text{ps} \subseteq \text{bl}^c \implies x \notin \text{ps}$   
**by** (*auto simp add: block-complement-def block-complement-elem-iff*)

**lemma** *block-comp-elem-alt-right*:  $\text{ps} \subseteq \mathcal{V} \implies (\bigwedge x . x \in \text{ps} \implies x \notin \text{bl}) \implies \text{ps} \subseteq \text{bl}^c$   
**by** (*auto simp add: block-complement-elem-iff*)

**lemma** *complement-index*:  
**assumes**  $\text{ps} \subseteq \mathcal{V}$   
**shows**  $\mathcal{B}^C \text{ index ps} = \text{size } \{\# b \in \# \mathcal{B} . (\forall x \in \text{ps} . x \notin b) \#\}$   
**proof** –  
**have**  $\mathcal{B}^C \text{ index ps} = \text{size } \{\# b \in \# \{\# \text{bl}^c . \text{bl} \in \# \mathcal{B} \#\} . \text{ps} \subseteq b \#\}$   
**by** (*simp add: complement-blocks-def points-index-def*)  
**then have**  $\mathcal{B}^C \text{ index ps} = \text{size } \{\# \text{bl}^c \mid \text{bl} \in \# \mathcal{B} . \text{ps} \subseteq \text{bl}^c \#\}$   
**by** (*metis image-mset-filter-swap*)  
**thus** *?thesis using assms by* (*simp add: block-complement-elem-iff*)  
**qed**

**lemma** *complement-index-2*:  
**assumes**  $\{x, y\} \subseteq \mathcal{V}$   
**shows**  $\mathcal{B}^C \text{ index } \{x, y\} = \text{size } \{\# b \in \# \mathcal{B} . x \notin b \wedge y \notin b \#\}$   
**proof** –  
**have**  $a: \bigwedge b . b \in \# \mathcal{B} \implies \forall x' \in \{x, y\} . x' \notin b \implies x \notin b \wedge y \notin b$   
**by** *simp*  
**have**  $\bigwedge b . b \in \# \mathcal{B} \implies x \notin b \wedge y \notin b \implies \forall x' \in \{x, y\} . x' \notin b$   
**by** *simp*  
**thus** *?thesis using assms a complement-index*  
**by** (*smt (verit) filter-mset-cong*)  
**qed**

**lemma** *complement-rep-number*:  
**assumes**  $x \in \mathcal{V}$  **and**  $\mathcal{B} \text{ rep } x = r$   
**shows**  $\mathcal{B}^C \text{ rep } x = \text{b} - r$   
**proof** –  
**have**  $r: \text{size } \{\# b \in \# \mathcal{B} . x \in b \#\} = r$  **using** *assms by* (*simp add: point-replication-number-def*)  
**then have**  $a: \bigwedge b . b \in \# \mathcal{B} \implies x \in b \implies x \notin b^c$   
**by** (*simp add: block-complement-def*)  
**have**  $\bigwedge b . b \in \# \mathcal{B} \implies x \notin b \implies x \in b^c$   
**by** (*simp add: assms(1) block-complement-def*)  
**then have** *alt: (image-mset block-complement  $\mathcal{B}$ ) rep x = size  $\{\# b \in \# \mathcal{B} . x \notin b \#\}$*   
**using** *a filter-mset-cong image-mset-filter-swap2 point-replication-number-def*  
**by** (*smt (verit, ccfv-SIG) size-image-mset*)  
**have**  $\text{b} = \text{size } \{\# b \in \# \mathcal{B} . x \in b \#\} + \text{size } \{\# b \in \# \mathcal{B} . x \notin b \#\}$   
**by** (*metis multiset-partition size-union*)

```

thus ?thesis using alt
  by (simp add: r complement-blocks-def)
qed

lemma complement-blocks-wf:  $bl \in \# \mathcal{B}^C \implies bl \subseteq \mathcal{V}$ 
  by (auto simp add: complement-blocks-def block-complement-def)

lemma complement-wf [intro]: incidence-system  $\mathcal{V} \mathcal{B}^C$ 
  using complement-blocks-wf by (unfold-locales)

interpretation sys-complement: incidence-system  $\mathcal{V} \mathcal{B}^C$ 
  using complement-wf by simp
end

context finite-incidence-system
begin
lemma block-complement-size:  $b \subseteq \mathcal{V} \implies \text{card } (b^c) = \text{card } \mathcal{V} - \text{card } b$ 
  by (simp add: block-complement-def card-Diff-subset finite-subset card-mono of-nat-diff finite-sets)

lemma block-comp-incomplete: incomplete-block  $bl \implies \text{card } (bl^c) > 0$ 
  using block-complement-size by (simp add: wellformed)

lemma block-comp-incomplete-nempty: incomplete-block  $bl \implies bl^c \neq \{\}$ 
  using wellformed block-complement-def finite-blocks
  by (auto simp add: block-complement-size block-comp-incomplete card-subset-not-gt-card)

lemma incomplete-block-proper-subset: incomplete-block  $bl \implies bl \subset \mathcal{V}$ 
  using wellformed by fastforce

lemma complement-finite: finite-incidence-system  $\mathcal{V} \mathcal{B}^C$ 
  using complement-wf finite-sets by (simp add: incidence-system.finite-sysI)

interpretation comp-fin: finite-incidence-system  $\mathcal{V} \mathcal{B}^C$ 
  using complement-finite by simp

end

context design
begin
lemma (in design) complement-design:
  assumes  $\bigwedge bl . bl \in \# \mathcal{B} \implies \text{incomplete-block } bl$ 
  shows design  $\mathcal{V} (\mathcal{B}^C)$ 
proof –
  interpret fin: finite-incidence-system  $\mathcal{V} \mathcal{B}^C$  using complement-finite by simp
  show ?thesis using assms block-comp-incomplete-nempty wellformed
  by (unfold-locales) (auto simp add: complement-blocks-def)
qed

```

**end**

## 2.7.2 Multiples

An easy way to construct new set systems is to simply multiply the block collection by some constant

**context** *incidence-system*

**begin**

**abbreviation** *multiple-blocks* :: *nat*  $\Rightarrow$  'a set multiset **where**

*multiple-blocks* *n*  $\equiv$  *repeat-mset* *n*  $\mathcal{B}$

**lemma** *multiple-block-in-original*:  $b \in\# \text{multiple-blocks } n \implies b \in\# \mathcal{B}$

**by** (*simp add: elem-in-repeat-in-original*)

**lemma** *multiple-block-in*:  $n > 0 \implies b \in\# \mathcal{B} \implies b \in\# \text{multiple-blocks } n$

**by** (*simp add: elem-in-original-in-repeat*)

**lemma** *multiple-blocks-gt*:  $n > 0 \implies \text{size } (\text{multiple-blocks } n) \geq \text{size } \mathcal{B}$

**by** (*simp*)

**lemma** *block-original-count-le*:  $n > 0 \implies \text{count } \mathcal{B} \ b \leq \text{count } (\text{multiple-blocks } n) \ b$

**using** *count-repeat-mset* **by** *simp*

**lemma** *multiple-blocks-sub*:  $n > 0 \implies \mathcal{B} \subseteq\# (\text{multiple-blocks } n)$

**by** (*simp add: mset-subset-eqI block-original-count-le*)

**lemma** *multiple-1-same*: *multiple-blocks* 1 =  $\mathcal{B}$

**by** *simp*

**lemma** *multiple-unfold-1*: *multiple-blocks* (*Suc* *n*) = (*multiple-blocks* *n*) +  $\mathcal{B}$

**by** *simp*

**lemma** *multiple-point-rep-num*: (*multiple-blocks* *n*) *rep* *x* = ( $\mathcal{B}$  *rep* *x*) \* *n*

**proof** (*induction* *n*)

**case** 0

**then show** ?*case* **by** (*simp add: point-replication-number-def*)

**next**

**case** (*Suc* *n*)

**then have** *multiple-blocks* (*Suc* *n*) *rep* *x* =  $\mathcal{B}$  *rep* *x* \* *n* + ( $\mathcal{B}$  *rep* *x*)

**using** *Suc.IH* *Suc.prem*s **by** (*simp add: union-commute point-replication-number-def*)

**then show** ?*case*

**by** (*simp*)

**qed**

**lemma** *multiple-point-index*: (*multiple-blocks* *n*) *index* *ps* = ( $\mathcal{B}$  *index* *ps*) \* *n*

**by** (*induction* *n*) (*auto simp add: points-index-def*)

**lemma** *repeat-mset-block-point-rel*:  $\bigwedge b x. b \in \# \text{ multiple-blocks } n \implies x \in b \implies x \in \mathcal{V}$

**by** (*induction n*) (*auto, meson subset-iff wellformed*)

**lemma** *multiple-is-wellformed*: *incidence-system*  $\mathcal{V}$  (*multiple-blocks n*)

**using** *repeat-mset-subset-in wellformed repeat-mset-block-point-rel* **by** (*unfold-locales*) (*auto*)

**lemma** *multiple-blocks-num* [*simp*]: *size* (*multiple-blocks n*) =  $n * b$

**by** *simp*

**interpretation** *mult-sys*: *incidence-system*  $\mathcal{V}$  (*multiple-blocks n*)

**by** (*simp add: multiple-is-wellformed*)

**lemma** *multiple-block-multiplicity* [*simp*]: *mult-sys.multiplicity n bl* = (*multiplicity bl*) \*  $n$

**by** (*simp*)

**lemma** *multiple-block-sizes-same*:

**assumes**  $n > 0$

**shows** *sys-block-sizes* = *mult-sys.sys-block-sizes n*

**proof** –

**have** *def*: *mult-sys.sys-block-sizes n* = {*card bl* | *bl. bl*  $\in \#$  (*multiple-blocks n*)}

**by** (*simp add: mult-sys.sys-block-sizes-def*)

**then have** *eq*:  $\bigwedge bl. bl \in \#$  (*multiple-blocks n*)  $\longleftrightarrow bl \in \# \mathcal{B}$

**using** *assms multiple-block-in multiple-block-in-original* **by** *blast*

**thus** *?thesis* **using** *def* **by** (*simp add: sys-block-sizes-def eq*)

**qed**

**end**

**context** *finite-incidence-system*

**begin**

**lemma** *multiple-is-finite*: *finite-incidence-system*  $\mathcal{V}$  (*multiple-blocks n*)

**using** *multiple-is-wellformed finite-sets* **by** (*unfold-locales*) (*auto simp add: incidence-system-def*)

**end**

**context** *design*

**begin**

**lemma** *multiple-is-design*: *design*  $\mathcal{V}$  (*multiple-blocks n*)

**proof** –

**interpret** *fis*: *finite-incidence-system*  $\mathcal{V}$  *multiple-blocks n* **using** *multiple-is-finite*

**by** *simp*

**show** *?thesis* **using** *blocks-empty*

**by** (*unfold-locales*) (*auto simp add: elem-in-repeat-in-original repeat-mset-not-empty*)

qed

end

## 2.8 Simple Designs

Simple designs are those in which the multiplicity of each block is at most one. In other words, the block collection is a set. This can significantly ease reasoning.

**locale** *simple-incidence-system* = *incidence-system* +  
  **assumes** *simple* [*simp*]:  $bl \in \# \mathcal{B} \implies \text{multiplicity } bl = 1$

**begin**

**lemma** *simple-alt-def-all*:  $\forall bl \in \# \mathcal{B} . \text{multiplicity } bl = 1$   
  **using** *simple* **by** *auto*

**lemma** *simple-blocks-eq-sup*:  $\text{mset-set } (\text{design-support}) = \mathcal{B}$   
  **using** *distinct-mset-def simple design-support-def* **by** (*metis distinct-mset-set-mset-ident*)

**lemma** *simple-block-size-eq-card*:  $b = \text{card } (\text{design-support})$   
  **by** (*metis simple-blocks-eq-sup size-mset-set*)

**lemma** *points-index-simple-def*:  $\mathcal{B} \text{ index } ps = \text{card } \{b \in \text{design-support} . ps \subseteq b\}$   
  **using** *design-support-def points-index-def card-size-filter-eq simple-blocks-eq-sup*  
  **by** (*metis finite-set-mset*)

**lemma** *replication-num-simple-def*:  $\mathcal{B} \text{ rep } x = \text{card } \{b \in \text{design-support} . x \in b\}$   
  **using** *design-support-def point-replication-number-def card-size-filter-eq simple-blocks-eq-sup*  
  **by** (*metis finite-set-mset*)

**end**

**locale** *simple-design* = *design* + *simple-incidence-system*

  Additional reasoning about when something is not simple

**context** *incidence-system*

**begin**

**lemma** *simple-not-multiplicity*:  $b \in \# \mathcal{B} \implies \text{multiplicity } b > 1 \implies \neg \text{simple-incidence-system} \vee \mathcal{B}$

**using** *simple-incidence-system-def simple-incidence-system-axioms-def* **by** (*metis nat-neq-iff*)

**lemma** *multiple-not-simple*:

**assumes**  $n > 1$

**assumes**  $\mathcal{B} \neq \{\#\}$

**shows**  $\neg \text{simple-incidence-system} \vee (\text{multiple-blocks } n)$

**proof** (*rule ccontr, simp*)

```

assume simple-incidence-system  $\mathcal{V}$  (multiple-blocks  $n$ )
then have  $\bigwedge bl. bl \in \# \mathcal{B} \implies \text{count } (\text{multiple-blocks } n) \text{ } bl = 1$ 
  using assms(1) elem-in-original-in-repeat
  by (metis not-gr-zero not-less-zero simple-incidence-system.simple)
thus False using assms by auto
qed

end

```

## 2.9 Proper Designs

Many types of designs rely on parameter conditions that only make sense for non-empty designs. i.e. designs with at least one block, and therefore given well-formed condition, at least one point. To this end we define the notion of a "proper" design

```

locale proper-design = design +
  assumes b-non-zero:  $b \neq 0$ 
begin

```

```

lemma is-proper: proper-design  $\mathcal{V}$   $\mathcal{B}$  by intro-locales

```

```

lemma v-non-zero:  $v > 0$ 
  using b-non-zero v-eq0-imp-b-eq-0 by auto

```

```

lemma b-positive:  $b > 0$  using b-non-zero
  by (simp add: nonempty-has-size)

```

```

lemma design-points-nonempty:  $\mathcal{V} \neq \{\}$ 
  using v-non-zero by auto

```

```

lemma design-blocks-nonempty:  $\mathcal{B} \neq \{\#\}$ 
  using b-non-zero by auto

```

```

end

```

Intro rules for a proper design

```

lemma (in design) proper-designI[intro]:  $b \neq 0 \implies \text{proper-design } \mathcal{V} \mathcal{B}$ 
  by (unfold-locales) simp

```

```

lemma proper-designII[intro]:
  assumes design  $V B$  and  $B \neq \{\#\}$ 
  shows proper-design  $V B$ 

```

```

proof –
  interpret des: design  $V B$  using assms by simp
  show ?thesis using assms by unfold-locales simp
qed

```

Reasoning on construction closure for proper designs

```

context proper-design

```

**begin**

**lemma** *multiple-proper-design*:

**assumes**  $n > 0$

**shows** *proper-design*  $\mathcal{V}$  (*multiple-blocks*  $n$ )

**using** *multiple-is-design* *assms* *design-blocks-nempty* *multiple-block-in*

**by** (*metis* *block-set-nempty-imp-block-ex* *empty-iff* *proper-designII* *set-mset-empty*)

**lemma** *complement-proper-design*:

**assumes**  $\bigwedge bl . bl \in \# \mathcal{B} \implies$  *incomplete-block*  $bl$

**shows** *proper-design*  $\mathcal{V} \mathcal{B}^C$

**proof** –

**interpret** *des*: *design*  $\mathcal{V} \mathcal{B}^C$

**by** (*simp* *add*: *assms* *complement-design*)

**show** *?thesis* **using** *b-non-zero* **by** (*unfold-locales*) *auto*

**qed**

**end**

**end**

**theory** *Design-Operations* **imports** *Design-Basics*

**begin**

### 3 Design Operations

Incidence systems have a number of very typical computational operations which can be used for constructions in design theory. Definitions in this section are based off the handbook of combinatorial designs, hypergraph theory [2], and the GAP design theory library [5]

#### 3.1 Incidence system definitions

**context** *incidence-system*

**begin**

The basic add point operation only affects the point set of a design

**definition** *add-point* ::  $'a \Rightarrow 'a$  set **where**

*add-point*  $p \equiv$  *insert*  $p \mathcal{V}$

**lemma** *add-existing-point* [*simp*]:  $p \in \mathcal{V} \implies$  *add-point*  $p = \mathcal{V}$

**using** *add-point-def* **by** *fastforce*

**lemma** *add-point-wf*: *incidence-system* (*add-point*  $p$ )  $\mathcal{B}$

**using** *wf-invalid-point* *add-point-def* **by** (*unfold-locales*) (*auto*)

An extension of the basic add point operation also adds the point to a given set of blocks

**definition** *add-point-to-blocks* ::  $'a \Rightarrow 'a$  set  $set \Rightarrow 'a$  set *multiset* **where**

$add\text{-}point\text{-}to\text{-}blocks\ p\ bs \equiv \{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\} + \{\# b \in \# \mathcal{B} . b \notin bs\#\}$

**lemma** *add-point-blocks-blocks-alt*:  $add\text{-}point\text{-}to\text{-}blocks\ p\ bs = image\text{-}mset\ (insert\ p)\ (filter\text{-}mset\ (\lambda\ b . b \in bs)\ \mathcal{B}) + (filter\text{-}mset\ (\lambda\ b . b \notin bs)\ \mathcal{B})$   
**using** *add-point-to-blocks-def* **by** *simp*

**lemma** *add-point-existing-blocks*:  
**assumes**  $(\bigwedge\ bl . bl \in bs \implies p \in bl)$   
**shows**  $add\text{-}point\text{-}to\text{-}blocks\ p\ bs = \mathcal{B}$   
**proof** (*simp add: add-point-to-blocks-def*)  
**have**  $image\text{-}mset\ (insert\ p)\ \{\#b \in \# \mathcal{B} . b \in bs\#\} = \{\#b \in \# \mathcal{B} . b \in bs\#\}$  **using** *assms*  
**by** (*simp add: image-filter-cong insert-absorb*)  
**thus**  $image\text{-}mset\ (insert\ p)\ \{\#b \in \# \mathcal{B} . b \in bs\#\} + \{\#b \in \# \mathcal{B} . b \notin bs\#\} = \mathcal{B}$   
**using** *multiset-partition* **by** *simp*  
**qed**

**lemma** *add-new-point-rep-number*:  
**assumes**  $p \notin \mathcal{V}$   
**shows**  $(add\text{-}point\text{-}to\text{-}blocks\ p\ bs)\ rep\ p = size\ \{\#b \in \# \mathcal{B} . b \in bs\#\}$   
**proof** –  
**have**  $\bigwedge\ b . b \in \# \mathcal{B} \implies b \notin bs \implies p \notin b$   
**by** (*simp add: assms wf-invalid-point*)  
**then have**  $zero: \{\# b \in \# \mathcal{B} . b \notin bs\#\}\ rep\ p = 0$   
**by** (*simp add: filter-mset-empty-conv point-replication-number-def*)  
**have**  $(add\text{-}point\text{-}to\text{-}blocks\ p\ bs)\ rep\ p = \{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\}\ rep\ p + \{\# b \in \# \mathcal{B} . b \notin bs\#\}\ rep\ p$   
**by** (*simp add: add-point-to-blocks-def*)  
**then have**  $eq: (add\text{-}point\text{-}to\text{-}blocks\ p\ bs)\ rep\ p = \{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\}\ rep\ p$   
**using** *zero* **by** *simp*  
**have**  $\bigwedge\ bl . bl \in \# \{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\} \implies p \in bl$  **by** *auto*  
**then have**  $\{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\}\ rep\ p = size\ \{\# (insert\ p\ b) \mid b \in \# \mathcal{B} . b \in bs\#\}$   
**using** *point-replication-number-def* **by** (*metis filter-mset-True filter-mset-cong*)  
**thus** *?thesis* **by** (*simp add: eq*)  
**qed**

**lemma** *add-point-blocks-wf*:  $incidence\text{-}system\ (add\text{-}point\ p)\ (add\text{-}point\text{-}to\text{-}blocks\ p\ bs)$   
**by** (*unfold-locales*) (*auto simp add: add-point-def wf-invalid-point add-point-to-blocks-def*)

Basic (weak) delete point operation removes a point from both the point set and from any blocks that contain it to maintain wellformed property

**definition** *del-point* ::  $'a \Rightarrow 'a\ set$  **where**  
 $del\text{-}point\ p \equiv \mathcal{V} - \{p\}$

**definition** *del-point-blocks*:: 'a  $\Rightarrow$  'a set multiset **where**

*del-point-blocks*  $p \equiv \{\# (bl - \{p\}) \cdot bl \in \# \mathcal{B} \#\}$

**lemma** *del-point-block-count*:  $size (del-point-blocks p) = size \mathcal{B}$

**by** (*simp add: del-point-blocks-def*)

**lemma** *remove-invalid-point-block*:  $p \notin \mathcal{V} \Longrightarrow bl \in \# \mathcal{B} \Longrightarrow bl - \{p\} = bl$

**using** *wf-invalid-point* **by** *blast*

**lemma** *del-invalid-point*:  $p \notin \mathcal{V} \Longrightarrow (del-point p) = \mathcal{V}$

**by** (*simp add: del-point-def*)

**lemma** *del-invalid-point-blocks*:  $p \notin \mathcal{V} \Longrightarrow (del-point-blocks p) = \mathcal{B}$

**using** *del-invalid-point*

**by** (*auto simp add: remove-invalid-point-block del-point-blocks-def*)

**lemma** *delete-point-p-not-in-bl-blocks*:  $(\bigwedge bl. bl \in \# \mathcal{B} \Longrightarrow p \notin bl) \Longrightarrow (del-point-blocks p) = \mathcal{B}$

**by** (*simp add: del-point-blocks-def*)

**lemma** *delete-point-blocks-wf*:  $b \in \# (del-point-blocks p) \Longrightarrow b \subseteq \mathcal{V} - \{p\}$

**unfolding** *del-point-blocks-def* **using** *wellformed* **by** *auto*

**lemma** *delete-point-blocks-sub*:

**assumes**  $b \in \# (del-point-blocks p)$

**obtains**  $bl$  **where**  $bl \in \# \mathcal{B} \wedge b \subseteq bl$

**using** *assms* **by** (*auto simp add: del-point-blocks-def*)

**lemma** *delete-point-split-blocks*:  $del-point-blocks p =$

$\{\# bl \in \# \mathcal{B} \cdot p \notin bl \#\} + \{\# bl - \{p\} \mid bl \in \# \mathcal{B} \cdot p \in bl \#\}$

**proof** –

**have**  $sm: \bigwedge bl. p \notin bl \Longrightarrow bl - \{p\} = bl$  **by** *simp*

**have**  $del-point-blocks p = \{\# (bl - \{p\}) \cdot bl \in \# \mathcal{B} \#\}$  **by** (*simp add: del-point-blocks-def*)

**also have**  $\dots = \{\# (bl - \{p\}) \mid bl \in \# \mathcal{B} \cdot p \notin bl \#\} + \{\# (bl - \{p\}) \mid bl \in \# \mathcal{B} \cdot p \in bl \#\}$

**using** *multiset-partition* **by** (*metis image-mset-union union-commute*)

**finally have**  $del-point-blocks p = \{\# bl \mid bl \in \# \mathcal{B} \cdot p \notin bl \#\} +$

$\{\# (bl - \{p\}) \mid bl \in \# \mathcal{B} \cdot p \in bl \#\}$

**using** *sm mem-Collect-eq*

**by** (*metis (mono-tags, lifting) Multiset.set-mset-filter multiset.map-cong*)

**thus** *?thesis* **by** *simp*

**qed**

**lemma** *delete-point-index-eq*:

**assumes**  $ps \subseteq (del-point p)$

**shows**  $(del-point-blocks p) index ps = \mathcal{B} index ps$

**proof** –

**have**  $eq: filter-mset ((\subseteq) ps) \{\# bl - \{p\} \cdot bl \in \# \mathcal{B} \#\} =$

```

      image-mset (λ b . b - {p}) (filter-mset (λ b. ps ⊆ b - {p}) B)
    using filter-mset-image-mset by blast
    have p ∉ ps using assms del-point-def by blast
    then have ∧ bl . ps ⊆ bl ↔ ps ⊆ bl - {p} by blast
    then have ((filter-mset (λ b. ps ⊆ b - {p}) B)) = (filter-mset (λ b . ps ⊆ b) B)
  by auto
  then have size (image-mset (λ b . b - {p}) (filter-mset (λ b. ps ⊆ b - {p}) B))

    = B index ps
    by (simp add: points-index-def)
  thus ?thesis using eq
    by (simp add: del-point-blocks-def points-index-def)
qed

```

**lemma** *delete-point-wf: incidence-system (del-point p) (del-point-blocks p)*  
 using *delete-point-blocks-wf del-point-def* by (*unfold-locales*) auto

The concept of a strong delete point comes from hypergraph theory. When a point is deleted, any blocks containing it are also deleted

**definition** *str-del-point-blocks* :: 'a ⇒ 'a set multiset **where**  
*str-del-point-blocks p* ≡ {# bl ∈# B . p ∉ bl#}

**lemma** *str-del-point-blocks-alt: str-del-point-blocks p = B - {# bl ∈# B . p ∈ bl#}*  
 using *add-diff-cancel-left' multiset-partition* by (*metis str-del-point-blocks-def*)

**lemma** *delete-point-strong-block-in: p ∉ bl ⇒ bl ∈# B ⇒ bl ∈# str-del-point-blocks p*  
 by (*simp add: str-del-point-blocks-def*)

**lemma** *delete-point-strong-block-not-in: p ∈ bl ⇒ bl ∉# (str-del-point-blocks) p*  
 by (*simp add: str-del-point-blocks-def*)

**lemma** *delete-point-strong-block-in-iff: bl ∈# B ⇒ bl ∈# str-del-point-blocks p*  
 $\longleftrightarrow p \notin bl$   
 using *delete-point-strong-block-in delete-point-strong-block-not-in*  
 by (*simp add: str-del-point-blocks-def*)

**lemma** *delete-point-strong-block-subset: str-del-point-blocks p ⊆# B*  
 by (*simp add: str-del-point-blocks-def*)

**lemma** *delete-point-strong-block-in-orig: bl ∈# str-del-point-blocks p ⇒ bl ∈# B*  
 using *delete-point-strong-block-subset* by (*metis mset-subset-eqD*)

**lemma** *delete-invalid-pt-strong-eq: p ∉ V ⇒ B = str-del-point-blocks p*  
 unfolding *str-del-point-blocks-def* using *wf-invalid-point empty-iff*  
 by (*metis Multiset.diff-cancel filter-mset-eq-conv set-mset-empty subset-mset.order-refl*)

**lemma** *strong-del-point-index-alt:*

**assumes**  $ps \subseteq (\text{del-point } p)$   
**shows**  $(\text{str-del-point-blocks } p) \text{ index } ps =$   
 $\mathcal{B} \text{ index } ps - \{\# \text{ bl} \in \# \mathcal{B} . p \in \text{bl}\# \} \text{ index } ps$   
**using** *str-del-point-blocks-alt points-index-def*  
**by** (*metis filter-diff-mset multiset-filter-mono multiset-filter-subset size-Diff-submset*)

**lemma** *strong-del-point-incidence-wf: incidence-system (del-point p) (str-del-point-blocks p)*  
**using** *wellformed str-del-point-blocks-def del-point-def* **by** (*unfold-locales*) *auto*

Add block operation

**definition** *add-block* :: 'a set  $\Rightarrow$  'a set multiset **where**  
*add-block*  $b \equiv \mathcal{B} + \{\#b\# \}$

**lemma** *add-block-alt: add-block b = add-mset b B*  
**by** (*simp add: add-block-def*)

**lemma** *add-block-rep-number-in:*  
**assumes**  $x \in b$   
**shows**  $(\text{add-block } b) \text{ rep } x = \mathcal{B} \text{ rep } x + 1$   
**proof** –  
**have**  $(\text{add-block } b) = \{\#b\# \} + \mathcal{B}$  **by** (*simp add: add-block-def*)  
**then have** *split:*  $(\text{add-block } b) \text{ rep } x = \{\#b\# \} \text{ rep } x + \mathcal{B} \text{ rep } x$   
**by** (*metis point-rep-number-split*)  
**have**  $\{\#b\# \} \text{ rep } x = 1$  **using** *assms* **by** *simp*  
**thus** *?thesis* **using** *split* **by** *auto*  
**qed**

**lemma** *add-block-rep-number-not-in:  $x \notin b \implies (\text{add-block } b) \text{ rep } x = \mathcal{B} \text{ rep } x$*   
**using** *point-rep-number-split add-block-alt point-rep-singleton-inval*  
**by** (*metis add.right-neutral union-mset-add-mset-right*)

**lemma** *add-block-index-in:*  
**assumes**  $ps \subseteq b$   
**shows**  $(\text{add-block } b) \text{ index } ps = \mathcal{B} \text{ index } ps + 1$   
**proof** –  
**have**  $(\text{add-block } b) = \{\#b\# \} + \mathcal{B}$  **by** (*simp add: add-block-def*)  
**then have** *split:*  $(\text{add-block } b) \text{ index } ps = \{\#b\# \} \text{ index } ps + \mathcal{B} \text{ index } ps$   
**by** (*metis point-index-distrib*)  
**have**  $\{\#b\# \} \text{ index } ps = 1$  **using** *assms* *points-index-singleton* **by** *auto*  
**thus** *?thesis* **using** *split* **by** *simp*  
**qed**

**lemma** *add-block-index-not-in:  $\neg (ps \subseteq b) \implies (\text{add-block } b) \text{ index } ps = \mathcal{B} \text{ index } ps$*   
**using** *point-index-distrib points-index-singleton-zero*  
**by** (*metis add.right-neutral add-block-def*)

Note the add block incidence system is defined slightly differently than textbook definitions due to the modification to the point set. This ensures

the operation is closed, where otherwise a condition that  $b \subseteq \mathcal{V}$  would be required.

**lemma** *add-block-wf*: *incidence-system*  $(\mathcal{V} \cup b)$  (*add-block*  $b$ )  
**using** *wellformed* *add-block-def* **by** (*unfold-locales*) *auto*

**lemma** *add-block-wf-cond*:  $b \subseteq \mathcal{V} \implies$  *incidence-system*  $\mathcal{V}$  (*add-block*  $b$ )  
**using** *add-block-wf* **by** (*metis sup.order-iff*)

Delete block removes a block from the block set. The point set is unchanged

**definition** *del-block* :: 'a set  $\Rightarrow$  'a set multiset **where**  
*del-block*  $b \equiv \mathcal{B} - \{\#b\}$

**lemma** *delete-block-subset*:  $(\text{del-block } b) \subseteq \# \mathcal{B}$   
**by** (*simp add: del-block-def*)

**lemma** *delete-invalid-block-eq*:  $b \notin \# \mathcal{B} \implies \text{del-block } b = \mathcal{B}$   
**by** (*simp add: del-block-def*)

**lemma** *delete-block-wf*: *incidence-system*  $\mathcal{V}$  (*del-block*  $b$ )  
**by** (*unfold-locales*) (*simp add: del-block-def in-diffD wellformed*)

The strong delete block operation effectively deletes the block, as well as all points in that block

**definition** *str-del-block* :: 'a set  $\Rightarrow$  'a set multiset **where**  
*str-del-block*  $b \equiv \{\# bl - b \mid bl \in \# \mathcal{B} . bl \neq b \#\}$

**lemma** *strong-del-block-alt-def*: *str-del-block*  $b = \{\# bl - b . bl \in \# \text{removeAll-mset } b \mathcal{B} \#\}$   
**by** (*simp add: filter-mset-neq str-del-block-def*)

**lemma** *strong-del-block-wf*: *incidence-system*  $(\mathcal{V} - b)$  (*str-del-block*  $b$ )  
**using** *wf-invalid-point str-del-block-def* **by** (*unfold-locales*) *auto*

**lemma** *str-del-block-del-point*:  
**assumes**  $\{x\} \notin \# \mathcal{B}$   
**shows** *str-del-block*  $\{x\} = (\text{del-point-blocks } x)$

**proof** –

**have** *neqx*:  $\bigwedge bl. bl \in \# \mathcal{B} \implies bl \neq \{x\}$  **using** *assms* **by** *auto*  
**have** *str-del-block*  $\{x\} = \{\# bl - \{x\} \mid bl \in \# \mathcal{B} . bl \neq \{x\} \#\}$  **by** (*simp add: str-del-block-def*)

**then have** *str-del-block*  $\{x\} = \{\# bl - \{x\} . bl \in \# \mathcal{B} \#\}$  **using** *assms neqx*  
**by** (*simp add: filter-mset-cong*)

**thus** *?thesis* **by** (*simp add: del-point-blocks-def*)

**qed**

### 3.2 Incidence System Interpretations

It is easy to interpret all operations as incidence systems in there own right. These can then be used to prove local properties on the new constructions, as well as reason on interactions between different operation sequences

**interpretation** *add-point-sys: incidence-system add-point p B*  
**using** *add-point-wf* **by** *simp*

**lemma** *add-point-sys-rep-numbers: add-point-sys.replication-numbers p = replication-numbers  $\cup$  {B rep p}*  
**using** *add-point-sys.replication-numbers-def replication-numbers-def add-point-def*  
**by** *auto*

**interpretation** *del-point-sys: incidence-system del-point p del-point-blocks p*  
**using** *delete-point-wf* **by** *auto*

**interpretation** *add-block-sys: incidence-system  $\mathcal{V} \cup bl$  add-block bl*  
**using** *add-block-wf* **by** *simp*

**interpretation** *del-block-sys: incidence-system  $\mathcal{V}$  del-block bl*  
**using** *delete-block-wf* **by** *simp*

**lemma** *add-del-block-inv:*  
**assumes** *bl  $\subseteq$   $\mathcal{V}$*   
**shows** *add-block-sys.del-block bl bl = B*  
**using** *add-block-sys.del-block-def add-block-def* **by** *simp*

**lemma** *del-add-block-inv: bl  $\in$  # B  $\implies$  del-block-sys.add-block bl bl = B*  
**using** *del-block-sys.add-block-def del-block-def wellformed* **by** *fastforce*

**lemma** *del-invalid-add-block-eq: bl  $\notin$  # B  $\implies$  del-block-sys.add-block bl bl = add-block bl*  
**using** *del-block-sys.add-block-def* **by** (*simp add: delete-invalid-block-eq*)

**lemma** *add-delete-point-inv:*  
**assumes** *p  $\notin$   $\mathcal{V}$*   
**shows** *add-point-sys.del-point p p =  $\mathcal{V}$*   
**proof** –  
**have** *(add-point-sys.del-point p p) = (insert p  $\mathcal{V}$ ) – {p}*  
**using** *add-point-sys.del-point-def del-block-sys.add-point-def* **by** *auto*  
**thus** *?thesis*  
**by** (*simp add: assms*)  
**qed**  
**end**

### 3.3 Operation Closure for Designs

**context** *finite-incidence-system*  
**begin**

**lemma** *add-point-finite: finite-incidence-system* (add-point p)  $\mathcal{B}$   
**using** add-point-wf finite-sets add-point-def  
**by** (unfold-locales) (simp-all add: incidence-system-def)

**lemma** *add-point-to-blocks-finite: finite-incidence-system* (add-point p) (add-point-to-blocks p bs)  
**using** add-point-blocks-wf add-point-finite finite-incidence-system.finite-sets  
incidence-system.finite-sysI **by** blast

**lemma** *delete-point-finite:*  
*finite-incidence-system* (del-point p) (del-point-blocks p)  
**using** finite-sets delete-point-wf  
**by** (unfold-locales) (simp-all add: incidence-system-def del-point-def)

**lemma** *del-point-order:*  
**assumes**  $p \in \mathcal{V}$   
**shows**  $\text{card } (\text{del-point } p) = v - 1$   
**proof** –  
**have**  $vg0: \text{card } \mathcal{V} > 0$  **using** assms finite-sets card-gt-0-iff **by** auto  
**then have**  $\text{card } (\text{del-point } p) = \text{card } \mathcal{V} - 1$  **using** assms del-point-def  
**by** (metis card-Diff-singleton)  
**thus** ?thesis  
**using** vg0 **by** linarith  
**qed**

**lemma** *strong-del-point-finite:finite-incidence-system* (del-point p) (str-del-point-blocks p)  
**using** strong-del-point-incidence-wf del-point-def  
**by** (intro-locales) (simp-all add: finite-incidence-system-axioms-def finite-sets)

**lemma** *add-block-fin: finite b  $\implies$  finite-incidence-system* ( $\mathcal{V} \cup b$ ) (add-block b)  
**using** wf-invalid-point finite-sets add-block-def **by** (unfold-locales) auto

**lemma** *add-block-fin-cond:  $b \subseteq \mathcal{V} \implies$  finite-incidence-system*  $\mathcal{V}$  (add-block b)  
**using** add-block-wf-cond finite-incidence-system-axioms.intro finite-sets  
**by** (intro-locales) (simp-all)

**lemma** *delete-block-fin-incidence-sys: finite-incidence-system*  $\mathcal{V}$  (del-block b)  
**using** delete-block-wf finite-sets **by** (unfold-locales) (simp-all add: incidence-system-def)

**lemma** *strong-del-block-fin: finite-incidence-system* ( $\mathcal{V} - b$ ) (str-del-block b)  
**using** strong-del-block-wf finite-sets finite-incidence-system-axioms-def **by** (intro-locales) auto

**end**

**context** design  
**begin**

```

lemma add-point-design: design (add-point p)  $\mathcal{B}$ 
  using add-point-wf finite-sets blocks-nempty add-point-def
  by (unfold-locales) (auto simp add: incidence-system-def)

lemma delete-point-design:
  assumes ( $\bigwedge bl . bl \in \# \mathcal{B} \implies p \in bl \implies \text{card } bl \geq 2$ )
  shows design (del-point p) (del-point-blocks p)
proof (cases p \in \mathcal{V})
  case True
  interpret fis: finite-incidence-system (del-point p) (del-point-blocks p)
  using delete-point-finite by simp
  show ?thesis
  proof (unfold-locales)
  show  $\bigwedge bl . bl \in \# (\text{del-point-blocks } p) \implies bl \neq \{\}$  using assms del-point-def
  proof –
  fix bl
  assume bl \in \# (del-point-blocks p)
  then obtain bl' where block: bl' \in \# \mathcal{B} and rem: bl = bl' - \{p\}
  by (auto simp add: del-point-blocks-def)
  then have eq: p \notin bl' \implies bl \neq \{\} using block blocks-nempty by (simp add: rem)
  have p \in bl' \implies \text{card } bl \geq 1 using rem finite-blocks block assms block by fastforce
  then show bl \neq \{\} using eq assms by fastforce
  qed
  qed
next
  case False
  then show ?thesis using del-invalid-point del-invalid-point-blocks
  by (simp add: wf-design)
qed

lemma strong-del-point-design: design (del-point p) (str-del-point-blocks p)
proof –
  interpret fin: finite-incidence-system (del-point p) (str-del-point-blocks p)
  using strong-del-point-finite by simp
  show ?thesis using wf-design wf-design-iff del-point-def str-del-point-blocks-def
  by (unfold-locales) (auto)
qed

lemma add-block-design:
  assumes finite bl
  assumes bl \neq \{\}
  shows design (\mathcal{V} \cup bl) (add-block bl)
proof –
  interpret fin: finite-incidence-system (\mathcal{V} \cup bl) (add-block bl)
  using add-block-fin assms by simp
  show ?thesis using blocks-nempty assms add-block-def by (unfold-locales) auto
qed

```

**lemma** *add-block-design-cond*:  
**assumes**  $bl \subseteq \mathcal{V}$  **and**  $bl \neq \{\}$   
**shows** *design*  $\mathcal{V}$  (*add-block*  $bl$ )  
**proof** –  
**interpret** *fin*: *finite-incidence-system*  $\mathcal{V}$  (*add-block*  $bl$ )  
**using** *add-block-fin-cond* *assms* **by** *simp*  
**show** *?thesis* **using** *blocks-nempty* *assms* *add-block-def* **by** (*unfold-locales*) *auto*  
**qed**

**lemma** *delete-block-design*: *design*  $\mathcal{V}$  (*del-block*  $bl$ )  
**proof** –  
**interpret** *fin*: *finite-incidence-system*  $\mathcal{V}$  (*del-block*  $bl$ )  
**using** *delete-block-fin-incidence-sys* **by** *simp*  
**have**  $\bigwedge b . b \in \# \mathcal{B} \implies b \neq \{\}$   
**using** *blocks-nempty* *delete-block-subset* **by** *blast*  
**then show** *?thesis* **by** (*unfold-locales*) (*simp-all*)  
**qed**

**lemma** *strong-del-block-des*:  
**assumes**  $\bigwedge bl . bl \in \# \mathcal{B} \implies \neg (bl \subset b)$   
**shows** *design*  $(\mathcal{V} - b)$  (*str-del-block*  $b$ )  
**proof** –  
**interpret** *fin*: *finite-incidence-system*  $\mathcal{V} - b$  (*str-del-block*  $b$ )  
**using** *strong-del-block-fin* **by** *simp*  
**show** *?thesis* **using** *assms* *str-del-block-def* **by** (*unfold-locales*) *auto*  
**qed**

**end**

**context** *proper-design*

**begin**

**lemma** *delete-point-proper*:  
**assumes**  $\bigwedge bl . bl \in \# \mathcal{B} \implies p \in bl \implies 2 \leq \text{card } bl$   
**shows** *proper-design* (*del-point*  $p$ ) (*del-point-blocks*  $p$ )  
**proof** –  
**interpret** *des*: *design* *del-point*  $p$  *del-point-blocks*  $p$   
**using** *delete-point-design* *assms* **by** *blast*  
**show** *?thesis* **using** *design-blocks-nempty* *del-point-def* *del-point-blocks-def*  
**by** (*unfold-locales*) *simp*  
**qed**

**lemma** *strong-delete-point-proper*:  
**assumes**  $\bigwedge bl . bl \in \# \mathcal{B} \implies p \in bl \implies 2 \leq \text{card } bl$   
**assumes**  $\mathcal{B} \text{ rep } p < b$   
**shows** *proper-design* (*del-point*  $p$ ) (*str-del-point-blocks*  $p$ )  
**proof** –  
**interpret** *des*: *design* *del-point*  $p$  *str-del-point-blocks*  $p$   
**using** *strong-del-point-design* *assms* **by** *blast*

```

show ?thesis using assms design-blocks-nempty point-rep-num-inv-non-empty
str-del-point-blocks-def del-point-def by (unfold-locales) simp
qed

end

```

### 3.4 Combining Set Systems

Similar to multiple, another way to construct a new set system is to combine two existing ones. We introduce a new locale enabling us to reason on two different incidence systems

```

locale two-set-systems = sys1: incidence-system  $\mathcal{V}$   $\mathcal{B}$  + sys2: incidence-system  $\mathcal{V}'$ 
 $\mathcal{B}'$ 

```

```

for  $\mathcal{V} :: ('a \text{ set})$  and  $\mathcal{B}$  and  $\mathcal{V}' :: ('a \text{ set})$  and  $\mathcal{B}'$ 
begin

```

```

abbreviation combine-points  $\equiv \mathcal{V} \cup \mathcal{V}'$ 

```

```

notation combine-points ( $\langle \mathcal{V}^+ \rangle$ )

```

```

abbreviation combine-blocks  $\equiv \mathcal{B} + \mathcal{B}'$ 

```

```

notation combine-blocks ( $\langle \mathcal{B}^+ \rangle$ )

```

```

sublocale combine-sys: incidence-system  $\mathcal{V}^+$   $\mathcal{B}^+$ 
using sys1.wellformed sys2.wellformed by (unfold-locales) auto

```

```

lemma combine-points-index:  $\mathcal{B}^+$  index ps =  $\mathcal{B}$  index ps +  $\mathcal{B}'$  index ps
by (simp add: point-index-distrib)

```

```

lemma combine-rep-number:  $\mathcal{B}^+$  rep x =  $\mathcal{B}$  rep x +  $\mathcal{B}'$  rep x
by (simp add: point-replication-number-def)

```

```

lemma combine-multiple1:  $\mathcal{V} = \mathcal{V}' \implies \mathcal{B} = \mathcal{B}' \implies \mathcal{B}^+ = \text{sys1.multiple-blocks } 2$ 
by auto

```

```

lemma combine-multiple2:  $\mathcal{V} = \mathcal{V}' \implies \mathcal{B} = \mathcal{B}' \implies \mathcal{B}^+ = \text{sys2.multiple-blocks } 2$ 
by auto

```

```

lemma combine-multiplicity: combine-sys.multiplicity b = sys1.multiplicity b +
sys2.multiplicity b
by simp

```

```

lemma combine-block-sizes: combine-sys.sys-block-sizes =
sys1.sys-block-sizes  $\cup$  sys2.sys-block-sizes
using sys1.sys-block-sizes-def sys2.sys-block-sizes-def combine-sys.sys-block-sizes-def
by (auto)

```

```

end

```

```

locale two-fin-set-systems = two-set-systems  $\mathcal{V}$   $\mathcal{B}$   $\mathcal{V}'$   $\mathcal{B}'$  + sys1: finite-incidence-system
 $\mathcal{V}$   $\mathcal{B}$  +
  sys2: finite-incidence-system  $\mathcal{V}'$   $\mathcal{B}'$  for  $\mathcal{V}$   $\mathcal{B}$   $\mathcal{V}'$   $\mathcal{B}'$ 
begin

sublocale combine-fin-sys: finite-incidence-system  $\mathcal{V}^+$   $\mathcal{B}^+$ 
  using sys1.finite-sets sys2.finite-sets by (unfold-locales) (simp)

lemma combine-order:  $\text{card } (\mathcal{V}^+) \geq \text{card } \mathcal{V}$ 
  by (meson Un-upper1 card-mono combine-fin-sys.finite-sets)

lemma combine-order-2:  $\text{card } (\mathcal{V}^+) \geq \text{card } \mathcal{V}'$ 
  by (meson Un-upper2 card-mono combine-fin-sys.finite-sets)

end

locale two-designs = two-fin-set-systems  $\mathcal{V}$   $\mathcal{B}$   $\mathcal{V}'$   $\mathcal{B}'$  + des1: design  $\mathcal{V}$   $\mathcal{B}$  +
  des2: design  $\mathcal{V}'$   $\mathcal{B}'$  for  $\mathcal{V}$   $\mathcal{B}$   $\mathcal{V}'$   $\mathcal{B}'$ 
begin

sublocale combine-des: design  $\mathcal{V}^+$   $\mathcal{B}^+$ 
  apply (unfold-locales)
  using des1.blocks-nempty-alt des2.blocks-nempty-alt by fastforce

end

locale two-designs-proper = two-designs +
  assumes blocks-nempty:  $\mathcal{B} \neq \{\#\} \vee \mathcal{B}' \neq \{\#\}$ 
begin

lemma des1-is-proper:  $\mathcal{B} \neq \{\#\} \implies \text{proper-design } \mathcal{V} \mathcal{B}$ 
  by (unfold-locales) (simp)

lemma des2-is-proper:  $\mathcal{B}' \neq \{\#\} \implies \text{proper-design } \mathcal{V}' \mathcal{B}'$ 
  by (unfold-locales) (simp)

lemma min-one-proper-design:  $\text{proper-design } \mathcal{V} \mathcal{B} \vee \text{proper-design } \mathcal{V}' \mathcal{B}'$ 
  using blocks-nempty des1-is-proper des2-is-proper by (unfold-locales, blast)

sublocale combine-proper-des: proper-design  $\mathcal{V}^+$   $\mathcal{B}^+$ 
  apply (unfold-locales)
  by (metis blocks-nempty size-eq-0-iff-empty subset-mset.zero-eq-add-iff-both-eq-0)
end

end

```

## 4 Block and Balanced Designs

We define a selection of the many different types of block and balanced designs, building up to properties required for defining a BIBD, in addition to several base generalisations

```
theory Block-Designs imports Design-Operations  
begin
```

### 4.1 Block Designs

A block design is a design where all blocks have the same size.

#### 4.1.1 K Block Designs

An important generalisation of a typical block design is the  $\mathcal{K}$  block design, where all blocks must have a size  $x$  where  $x \in \mathcal{K}$

```
locale K-block-design = proper-design +  
  fixes sizes :: nat set ( $\langle \mathcal{K} \rangle$ )  
  assumes block-sizes:  $bl \in \# \mathcal{B} \implies \text{card } bl \in \mathcal{K}$   
  assumes positive-ints:  $x \in \mathcal{K} \implies x > 0$   
begin
```

```
lemma sys-block-size-subset:  $\text{sys-block-sizes} \subseteq \mathcal{K}$   
  using block-sizes sys-block-sizes-obtain-bl by blast
```

```
end
```

#### 4.1.2 Uniform Block Design

The typical uniform block design is defined below

```
locale block-design = proper-design +  
  fixes u-block-size :: nat ( $\langle k \rangle$ )  
  assumes uniform [simp]:  $bl \in \# \mathcal{B} \implies \text{card } bl = k$   
begin
```

```
lemma k-non-zero:  $k \geq 1$ 
```

```
proof –
```

```
  obtain bl where bl-in:  $bl \in \# \mathcal{B}$   
    using design-blocks-nempty by auto  
  then have  $\text{card } bl \geq 1$  using block-size-gt-0  
    by (metis less-not-refl less-one not-le-imp-less)  
  thus ?thesis by (simp add: bl-in)  
qed
```

```
lemma uniform-alt-def-all:  $\forall bl \in \# \mathcal{B} . \text{card } bl = k$   
  using uniform by auto
```

**lemma** *uniform-unfold-point-set*:  $bl \in \# \mathcal{B} \implies \text{card } \{p \in \mathcal{V}. p \in bl\} = k$   
**using** *uniform wellformed* **by** (*simp add: Collect-conj-eq inf.absorb-iff2*)

**lemma** *uniform-unfold-point-set-mset*:  $bl \in \# \mathcal{B} \implies \text{size } \{\#p \in \# \text{mset-set } \mathcal{V}. p \in bl \#\} = k$   
**using** *uniform-unfold-point-set* **by** (*simp add: finite-sets*)

**lemma** *sys-block-sizes-uniform* [*simp*]:  $\text{sys-block-sizes} = \{k\}$   
**proof** –  
**have**  $\text{sys-block-sizes} = \{bs . \exists bl . bs = \text{card } bl \wedge bl \in \# \mathcal{B}\}$  **by** (*simp add: sys-block-sizes-def*)  
**then have**  $\text{sys-block-sizes} = \{bs . bs = k\}$  **using** *uniform uniform-unfold-point-set*  
  
*b-positive block-set-nempty-imp-block-ex*  
**by** (*smt (verit, best) Collect-cong design-blocks-nempty*)  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *sys-block-sizes-uniform-single*: *is-singleton* (*sys-block-sizes*)  
**by** *simp*

**lemma** *uniform-size-incomp*:  $k \leq v - 1 \implies bl \in \# \mathcal{B} \implies \text{incomplete-block } bl$   
**using** *uniform k-non-zero*  
**by** (*metis block-size-lt-v diff-diff-cancel diff-is-0-eq' less-numeral-extra(1) nat-less-le*)

**lemma** *uniform-complement-block-size*:  
**assumes**  $bl \in \# \mathcal{B}^C$   
**shows**  $\text{card } bl = v - k$   
**proof** –  
**obtain**  $bl'$  **where**  $bl\text{-assm}: bl = bl'^c \wedge bl' \in \# \mathcal{B}$   
**using** *wellformed assms* **by** (*auto simp add: complement-blocks-def*)  
**then have**  $\text{int } (\text{card } bl') = k$  **by** *simp*  
**thus** *?thesis* **using**  $bl\text{-assm}$  *block-complement-size wellformed*  
**by** (*simp add: block-size-lt-order of-nat-diff*)  
**qed**

**lemma** *uniform-complement[intro]*:  
**assumes**  $k \leq v - 1$   
**shows**  $\text{block-design } \mathcal{V} \mathcal{B}^C (v - k)$   
**proof** –  
**interpret**  $des: \text{proper-design } \mathcal{V} \mathcal{B}^C$   
**using** *uniform-size-incomp assms complement-proper-design* **by** *auto*  
**show** *?thesis* **using**  $\text{assms}$  *uniform-complement-block-size* **by** (*unfold-locales*)  
(*simp*)  
**qed**

**lemma** *block-size-lt-v*:  $k \leq v$   
**using** *v-non-zero block-size-lt-v design-blocks-nempty uniform* **by** *auto*

**end**

**lemma** (in *proper-design*) *block-designI*[intro]:  $(\bigwedge bl . bl \in \# \mathcal{B} \implies \text{card } bl = k)$   
 $\implies \text{block-design } \mathcal{V} \mathcal{B} k$   
**by** (*unfold-locales*) (*auto*)

**context** *block-design*  
**begin**

**lemma** *block-design-multiple*:  $n > 0 \implies \text{block-design } \mathcal{V} (\text{multiple-blocks } n) k$   
**using** *elem-in-repeat-in-original multiple-proper-design proper-design.block-designI*  
  
**by** (*metis uniform-alt-def-all*)

**end**

A uniform block design is clearly a type of *K\_block\_design* with a singleton *K* set

**sublocale** *block-design*  $\subseteq$  *K-block-design*  $\mathcal{V} \mathcal{B} \{k\}$   
**using** *k-non-zero uniform* **by** *unfold-locales simp-all*

### 4.1.3 Incomplete Designs

An incomplete design is a design where  $k < v$ , i.e. no block is equal to the point set

**locale** *incomplete-design* = *block-design* +  
**assumes** *incomplete*:  $k < v$

**begin**

**lemma** *incomplete-imp-incomp-block*:  $bl \in \# \mathcal{B} \implies \text{incomplete-block } bl$   
**using** *incomplete uniform uniform-size-incomp* **by** *fastforce*

**lemma** *incomplete-imp-proper-subset*:  $bl \in \# \mathcal{B} \implies bl \subset \mathcal{V}$   
**using** *incomplete-block-proper-subset incomplete-imp-incomp-block* **by** *auto*  
**end**

**lemma** (in *block-design*) *incomplete-designI*[intro]:  $k < v \implies \text{incomplete-design } \mathcal{V} \mathcal{B} k$   
**by** *unfold-locales auto*

**context** *incomplete-design*  
**begin**

**lemma** *multiple-incomplete*:  $n > 0 \implies \text{incomplete-design } \mathcal{V} (\text{multiple-blocks } n) k$   
**using** *block-design-multiple incomplete* **by** (*simp add: block-design.incomplete-designI*)

**lemma** *complement-incomplete*: *incomplete-design*  $\mathcal{V} (\mathcal{B}^C) (v - k)$

**proof** –  
**have**  $v - k < v$  **using** *v-non-zero k-non-zero* **by** *linarith*  
**thus** *?thesis* **using** *uniform-complement incomplete incomplete-designI*  
**by** (*simp add: block-design.incomplete-designI*)  
**qed**  
**end**

## 4.2 Balanced Designs

$t$ -wise balance is a design with the property that all point subsets of size  $t$  occur in  $\lambda_t$  blocks

**locale** *t-wise-balance = proper-design +*  
**fixes** *grouping :: nat (⟨t⟩)* **and** *index :: nat (⟨ $\Lambda_t$ ⟩)*  
**assumes** *t-non-zero:  $t \geq 1$*   
**assumes** *t-lt-order:  $t \leq v$*   
**assumes** *balanced [simp]:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps = \Lambda_t$*   
**begin**

**lemma** *t-non-zero-suc:  $t \geq \text{Suc } 0$*   
**using** *t-non-zero* **by** *auto*

**lemma** *balanced-alt-def-all:  $\forall ps \subseteq \mathcal{V} . \text{card } ps = t \implies \mathcal{B} \text{ index } ps = \Lambda_t$*   
**using** *balanced* **by** *auto*

**end**

**lemma** (*in proper-design*) *t-wise-balanceI[intro]:  $t \leq v \implies t \geq 1 \implies$*   
 $(\bigwedge ps . ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps = \Lambda_t) \implies t\text{-wise-balance } \mathcal{V} \mathcal{B} t$   
 $\Lambda_t$   
**by** (*unfold-locales*) *auto*

**context** *t-wise-balance*  
**begin**

**lemma** *obtain-t-subset-points:*  
**obtains**  $T$  **where**  $T \subseteq \mathcal{V}$  *card  $T = t$  finite  $T$*   
**using** *obtain-subset-with-card-n design-points-empty t-lt-order t-non-zero finite-sets*  
**by** *auto*

**lemma** *multiple-t-wise-balance-index [simp]:*  
**assumes**  $ps \subseteq \mathcal{V}$   
**assumes**  $\text{card } ps = t$   
**shows** (*multiple-blocks n*) *index  $ps = \Lambda_t * n$*   
**using** *multiple-point-index balanced assms* **by** *fastforce*

**lemma** *multiple-t-wise-balance:*  
**assumes**  $n > 0$   
**shows** *t-wise-balance  $\mathcal{V}$  (multiple-blocks n) t ( $\Lambda_t * n$ )*

**proof** –

**interpret** *des: proper-design*  $\mathcal{V}$  (*multiple-blocks*  $n$ ) **by** (*simp add: assms multiple-proper-design*)

**show** *?thesis* **using** *t-non-zero t-ll-order multiple-t-wise-balance-index*

**by** (*unfold-locales*) (*simp-all*)

**qed**

**lemma** *wise-set-pair-index*:  $ps \subseteq \mathcal{V} \implies ps2 \subseteq \mathcal{V} \implies ps \neq ps2 \implies \text{card } ps = t$   
 $\implies \text{card } ps2 = t$

$\implies \mathcal{B} \text{ index } ps = \mathcal{B} \text{ index } ps2$

**using** *balanced* **by** *simp*

**lemma** *t-wise-balance-alt*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps = l2$

$\implies (\bigwedge ps . ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps = l2)$

**using** *wise-set-pair-index* **by** *blast*

**lemma** *index-1-imp-mult-1* [*simp*]:

**assumes**  $\Lambda_t = 1$

**assumes**  $bl \in \# \mathcal{B}$

**assumes**  $\text{card } bl \geq t$

**shows** *multiplicity*  $bl = 1$

**proof** (*rule ccontr*)

**assume**  $\neg (\text{multiplicity } bl = 1)$

**then have** *not: multiplicity*  $bl \neq 1$  **by** *simp*

**have** *multiplicity*  $bl \neq 0$  **using** *assms* **by** *simp*

**then have**  $m$ : *multiplicity*  $bl \geq 2$  **using** *not* **by** *linarith*

**obtain**  $ps$  **where**  $ps \subseteq bl \wedge \text{card } ps = t$

**using** *assms* *obtain-t-subset-points*

**by** (*metis* *obtain-subset-with-card-n*)

**then have**  $\mathcal{B} \text{ index } ps \geq 2$

**using**  $m$  *points-index-count-min*  $ps$  **by** *blast*

**then show** *False* **using** *balanced*  $ps$  *antisym-conv2* *not-numeral-less-zero* *numeral-le-one-iff*

*points-index-ps-nin* *semiring-norm(69)* *zero-neq-numeral*

**by** (*metis* *assms(1)*)

**qed**

**end**

#### 4.2.1 Sub-types of t-wise balance

Pairwise balance is when  $t = 2$ . These are commonly of interest

**locale** *pairwise-balance* = *t-wise-balance*  $\mathcal{V} \mathcal{B} \geq 2 \Lambda$

**for** *point-set* ( $\langle \mathcal{V} \rangle$ ) **and** *block-collection* ( $\langle \mathcal{B} \rangle$ ) **and** *index* ( $\langle \Lambda \rangle$ )

We can combine the balance properties with  $K$ \_block design to define tBD's (t-wise balanced designs), and PBD's (pairwise balanced designs)

**locale** *tBD* = *t-wise-balance* + *K-block-design* +

**assumes** *block-size-gt-t*:  $k \in \mathcal{K} \implies k \geq t$

```

locale  $\Lambda$ -PBD = pairwise-balance + K-block-design +
  assumes block-size-gt-t:  $k \in \mathcal{K} \implies k \geq 2$ 

sublocale  $\Lambda$ -PBD  $\subseteq$  tBD  $\vee$   $\mathcal{B}$   $\geq 2$   $\Lambda$   $\mathcal{K}$ 
  using t-lt-order block-size-gt-t by (unfold-locales) (simp-all)

locale PBD =  $\Lambda$ -PBD  $\vee$   $\mathcal{B}$  1  $\mathcal{K}$  for point-set ( $\langle \mathcal{V} \rangle$ ) and block-collection ( $\langle \mathcal{B} \rangle$ ) and
  sizes ( $\langle \mathcal{K} \rangle$ )
begin
lemma multiplicity-is-1:
  assumes  $bl \in \# \mathcal{B}$ 
  shows multiplicity  $bl = 1$ 
  using block-size-gt-t index-1-imp-mult-1 by (simp add: assms block-sizes)

end

sublocale PBD  $\subseteq$  simple-design
  using multiplicity-is-1 by (unfold-locales)

  PBD's are often only used in the case where  $k$  is uniform, defined here.

locale  $k$ - $\Lambda$ -PBD = pairwise-balance + block-design +
  assumes block-size-t:  $2 \leq k$ 

sublocale  $k$ - $\Lambda$ -PBD  $\subseteq$   $\Lambda$ -PBD  $\vee$   $\mathcal{B}$   $\Lambda$  { $k$ }
  using  $k$ -non-zero uniform block-size-t by(unfold-locales) (simp-all)

locale  $k$ -PBD =  $k$ - $\Lambda$ -PBD  $\vee$   $\mathcal{B}$  1  $k$  for point-set ( $\langle \mathcal{V} \rangle$ ) and block-collection ( $\langle \mathcal{B} \rangle$ )
and u-block-size ( $\langle k \rangle$ )

sublocale  $k$ -PBD  $\subseteq$  PBD  $\vee$   $\mathcal{B}$  { $k$ }
  using block-size-t by (unfold-locales, simp-all)

```

#### 4.2.2 Covering and Packing Designs

Covering and packing designs involve a looser balance restriction. Upper/lower bounds are placed on the points index, instead of a strict equality

A  $t$ -covering design is a relaxed version of a tBD, where, for all point subsets of size  $t$ , a lower bound is put on the points index

```

locale t-covering-design = block-design +
  fixes grouping :: nat ( $\langle t \rangle$ )
  fixes min-index :: nat ( $\langle \Lambda_t \rangle$ )
  assumes covering:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps \geq \Lambda_t$ 
  assumes block-size-t:  $t \leq k$ 
  assumes t-non-zero:  $t \geq 1$ 
begin

lemma covering-alt-def-all:  $\forall ps \subseteq \mathcal{V} . \text{card } ps = t \longrightarrow \mathcal{B} \text{ index } ps \geq \Lambda_t$ 

```

```

using covering by auto

end

lemma (in block-design) t-covering-designI [intro]:  $t \leq k \implies t \geq 1 \implies$ 
 $(\bigwedge ps. ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps \geq \Lambda_t) \implies t\text{-covering-design } \mathcal{V} \mathcal{B}$ 
 $k \ t \ \Lambda_t$ 
by (unfold-locales) simp-all

A  $t$ -packing design is a relaxed version of a tBD, where, for all point
subsets of size  $t$ , an upper bound is put on the points index

locale t-packing-design = block-design +
fixes grouping :: nat ( $\langle t \rangle$ )
fixes min-index :: nat ( $\langle \Lambda_t \rangle$ )
assumes packing:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps \leq \Lambda_t$ 
assumes block-size-t:  $t \leq k$ 
assumes t-non-zero:  $t \geq 1$ 
begin

lemma packing-alt-def-all:  $\forall ps \subseteq \mathcal{V} . \text{card } ps = t \implies \mathcal{B} \text{ index } ps \leq \Lambda_t$ 
using packing by auto

end

lemma (in block-design) t-packing-designI [intro]:  $t \leq k \implies t \geq 1 \implies$ 
 $(\bigwedge ps . ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B} \text{ index } ps \leq \Lambda_t) \implies t\text{-packing-design } \mathcal{V} \mathcal{B}$ 
 $k \ t \ \Lambda_t$ 
by (unfold-locales) simp-all

lemma packing-covering-imp-balance:
assumes t-packing-design  $V \ B \ k \ t \ \Lambda_t$ 
assumes t-covering-design  $V \ B \ k \ t \ \Lambda_t$ 
shows t-wise-balance  $V \ B \ t \ \Lambda_t$ 
proof –
from assms interpret des: proper-design  $V \ B$ 
using block-design.axioms(1) t-covering-design.axioms(1) by blast
show ?thesis
proof (unfold-locales)
show  $1 \leq t$  using assms t-packing-design.t-non-zero by auto
show  $t \leq \text{des.v}$  using block-design.block-size-lt-v t-packing-design.axioms(1)
by (metis assms(1) dual-order.trans t-packing-design.block-size-t)
show  $\bigwedge ps. ps \subseteq V \implies \text{card } ps = t \implies B \text{ index } ps = \Lambda_t$ 
using t-packing-design.packing t-covering-design.covering by (metis assms
dual-order.antisym)
qed
qed

```

### 4.3 Constant Replication Design

When the replication number for all points in a design is constant, it is the design replication number.

```

locale constant-rep-design = proper-design +
  fixes design-rep-number :: nat (⟨r⟩)
  assumes rep-number [simp]:  $x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x = r$ 

begin

lemma rep-number-alt-def-all:  $\forall x \in \mathcal{V}. \mathcal{B} \text{ rep } x = r$ 
  by (simp)

lemma rep-number-unfold-set:  $x \in \mathcal{V} \implies \text{size } \{\#bl \in \# \mathcal{B} . x \in bl\# \} = r$ 
  using rep-number by (simp add: point-replication-number-def)

lemma rep-numbers-constant [simp]: replication-numbers = {r}
  unfolding replication-numbers-def using rep-number design-points-nempty Collect-cong finite.cases
  finite-sets insertCI singleton-conv
  by (smt (verit, ccfv-threshold) fst-conv snd-conv)

lemma replication-number-single: is-singleton (replication-numbers)
  using is-singleton-the-elem by simp

lemma constant-rep-point-pair:  $x1 \in \mathcal{V} \implies x2 \in \mathcal{V} \implies x1 \neq x2 \implies \mathcal{B} \text{ rep } x1 = \mathcal{B} \text{ rep } x2$ 
  using rep-number by auto

lemma constant-rep-alt:  $x1 \in \mathcal{V} \implies \mathcal{B} \text{ rep } x1 = r2 \implies (\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x = r2)$ 
  by (simp)

lemma constant-rep-point-not-0:
  assumes  $x \in \mathcal{V}$ 
  shows  $\mathcal{B} \text{ rep } x \neq 0$ 
proof (rule ccontr)
  assume  $\neg \mathcal{B} \text{ rep } x \neq 0$ 
  then have  $\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x = 0$  using rep-number assms by auto
  then have  $\bigwedge x . x \in \mathcal{V} \implies \text{size } \{\#bl \in \# \mathcal{B} . x \in bl\# \} = 0$ 
    by (simp add: point-replication-number-def)
  then show False using design-blocks-nempty wf-design wf-design-iff wf-invalid-point
    by (metis ex-in-conv filter-mset-empty-conv multiset-nonemptyE size-eq-0-iff-empty)
qed

lemma rep-not-zero:  $r \neq 0$ 
  using rep-number constant-rep-point-not-0 design-points-nempty by auto

lemma r-gzero:  $r > 0$ 

```

```

using rep-not-zero by auto

lemma r-lt-eq-b:  $r \leq b$ 
  using rep-number max-point-rep
  by (metis all-not-in-conv design-points-nempty)

lemma complement-rep-number:
  assumes  $\bigwedge bl . bl \in \# \mathcal{B} \implies \text{incomplete-block } bl$ 
  shows  $\text{constant-rep-design } \mathcal{V} \mathcal{B}^C (b - r)$ 
proof –
  interpret d: proper-design  $\mathcal{V} (\mathcal{B}^C)$  using complement-proper-design
    by (simp add: assms)
  show ?thesis using complement-rep-number rep-number by (unfold-locales) simp
qed

lemma multiple-rep-number:
  assumes  $n > 0$ 
  shows  $\text{constant-rep-design } \mathcal{V} (\text{multiple-blocks } n) (r * n)$ 
proof –
  interpret d: proper-design  $\mathcal{V} (\text{multiple-blocks } n)$  using multiple-proper-design
    by (simp add: assms)
  show ?thesis using multiple-point-rep-num by (unfold-locales) (simp-all)
qed
end

lemma (in proper-design) constant-rep-designI [intro]:  $(\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x = r)$ 
   $\implies \text{constant-rep-design } \mathcal{V} \mathcal{B} r$ 
  by unfold-locales auto

4.4 T-designs

All the before mentioned designs build up to the concept of a t-design, which has uniform block size and is t-wise balanced. We limit t to be less than k, so the balance condition has relevance

locale t-design = incomplete-design + t-wise-balance +
  assumes block-size-t:  $t \leq k$ 
begin

lemma point-indices-balanced: point-indices  $t = \{\Lambda_t\}$ 
proof –
  have point-indices  $t = \{i . \exists ps . i = \mathcal{B} \text{ index } ps \wedge \text{card } ps = t \wedge ps \subseteq \mathcal{V}\}$ 
    by (simp add: point-indices-def)
  then have point-indices  $t = \{i . i = \Lambda_t\}$  using balanced Collect-cong obtain-t-subset-points
    by (smt (verit, best))
  thus ?thesis by auto
qed

```

```

lemma point-indices-singleton: is-singleton (point-indices t)
  using point-indices-balanced is-singleton-the-elem by simp

end

lemma t-designI [intro]:
  assumes incomplete-design V B k
  assumes t-wise-balance V B t  $\Lambda_t$ 
  assumes  $t \leq k$ 
  shows t-design V B k t  $\Lambda_t$ 
  by (simp add: assms(1) assms(2) assms(3) t-design.intro t-design-axioms.intro)

sublocale t-design  $\subseteq$  t-covering-design  $\mathcal{V}$  B k t  $\Lambda_t$ 
  using t-non-zero by (unfold-locales) (auto simp add: block-size-t)

sublocale t-design  $\subseteq$  t-packing-design  $\mathcal{V}$  B k t  $\Lambda_t$ 
  using t-non-zero by (unfold-locales) (auto simp add: block-size-t)

lemma t-design-pack-cov [intro]:
  assumes  $k < \text{card } V$ 
  assumes t-covering-design V B k t  $\Lambda_t$ 
  assumes t-packing-design V B k t  $\Lambda_t$ 
  shows t-design V B k t  $\Lambda_t$ 
proof –
  from assms interpret id: incomplete-design V B k
    using block-design.incomplete-designI t-packing-design.axioms(1)
    by blast
  from assms interpret balance: t-wise-balance V B t  $\Lambda_t$ 
    using packing-covering-imp-balance by blast
  show ?thesis using assms(3)
    by (unfold-locales) (simp-all add: t-packing-design.block-size-t)
qed

sublocale t-design  $\subseteq$  tBD  $\mathcal{V}$  B t  $\Lambda_t$  {k}
  using uniform k-non-zero block-size-t by (unfold-locales) simp-all

context t-design
begin

lemma multiple-t-design: n > 0  $\implies$  t-design  $\mathcal{V}$  (multiple-blocks n) k t ( $\Lambda_t * n$ )
  using multiple-t-wise-balance multiple-incomplete block-size-t by (simp add: t-designI)

lemma t-design-min-v: v > 1
  using k-non-zero incomplete by simp

end

```

## 4.5 Steiner Systems

Steiner systems are a special type of  $t$ -design where  $\Lambda_t = 1$

**locale** *steiner-system* = *t-design*  $\mathcal{V} \mathcal{B} k t 1$   
**for** *point-set* ( $\langle \mathcal{V} \rangle$ ) **and** *block-collection* ( $\langle \mathcal{B} \rangle$ ) **and** *u-block-size* ( $\langle k \rangle$ ) **and** *grouping* ( $\langle t \rangle$ )

**begin**

**lemma** *block-multiplicity* [*simp*]:  
**assumes**  $bl \in \# \mathcal{B}$   
**shows** *multiplicity*  $bl = 1$   
**by** (*simp add: assms block-size-t*)

**end**

**sublocale** *steiner-system*  $\subseteq$  *simple-design*  
**by** *unfold-locales (simp)*

**lemma** (**in** *t-design*) *steiner-systemI*[*intro*]:  $\Lambda_t = 1 \implies$  *steiner-system*  $\mathcal{V} \mathcal{B} k t$   
**using** *t-non-zero t-lt-order block-size-t*  
**by** *unfold-locales auto*

## 4.6 Combining block designs

We define some closure properties for various block designs under the combine operator. This is done using locales to reason on multiple instances of the same type of design, building on what was presented in the design operations theory

**locale** *two-t-wise-eq-points* = *two-designs-proper*  $\mathcal{V} \mathcal{B} \mathcal{V} \mathcal{B}' +$  *des1: t-wise-balance*  $\mathcal{V} \mathcal{B} t \Lambda_t +$   
*des2: t-wise-balance*  $\mathcal{V} \mathcal{B}' t \Lambda_t' \text{ for } \mathcal{V} \mathcal{B} t \Lambda_t \mathcal{B}' \Lambda_t'$

**begin**

**lemma** *combine-t-wise-balance-index*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = t \implies \mathcal{B}^+ \text{ index } ps = (\Lambda_t + \Lambda_t')$   
**using** *des1.balanced des2.balanced* **by** (*simp add: combine-points-index*)

**lemma** *combine-t-wise-balance*: *t-wise-balance*  $\mathcal{V}^+ \mathcal{B}^+ t (\Lambda_t + \Lambda_t')$

**proof** (*unfold-locales, simp add: des1.t-non-zero-suc*)

**have**  $\text{card } \mathcal{V}^+ \geq \text{card } \mathcal{V}$  **by** *simp*

**then show**  $t \leq \text{card } (\mathcal{V}^+)$  **using** *des1.t-lt-order* **by** *linarith*

**show**  $\bigwedge ps. ps \subseteq \mathcal{V}^+ \implies \text{card } ps = t \implies (\mathcal{B}^+ \text{ index } ps) = \Lambda_t + \Lambda_t'$

**using** *combine-t-wise-balance-index* **by** *blast*

**qed**

**sublocale** *combine-t-wise-des*: *t-wise-balance*  $\mathcal{V}^+ \mathcal{B}^+ t (\Lambda_t + \Lambda_t')$   
**using** *combine-t-wise-balance* **by** *auto*

**end**

**locale** *two-k-block-designs* = *two-designs-proper*  $\mathcal{V} \mathcal{B} \mathcal{V}' \mathcal{B}' +$  *des1*: *block-design*  $\mathcal{V} \mathcal{B} k +$   
*des2*: *block-design*  $\mathcal{V}' \mathcal{B}' k$  **for**  $\mathcal{V} \mathcal{B} k \mathcal{V}' \mathcal{B}'$   
**begin**

**lemma** *block-design-combine*: *block-design*  $\mathcal{V}^+ \mathcal{B}^+ k$   
**using** *des1.uniform des2.uniform* **by** (*unfold-locales*) (*auto*)

**sublocale** *combine-block-des*: *block-design*  $\mathcal{V}^+ \mathcal{B}^+ k$   
**using** *block-design-combine* **by** *simp*

**end**

**locale** *two-rep-designs-eq-points* = *two-designs-proper*  $\mathcal{V} \mathcal{B} \mathcal{V} \mathcal{B}' +$  *des1*: *constant-rep-design*  $\mathcal{V} \mathcal{B} r +$   
*des2*: *constant-rep-design*  $\mathcal{V} \mathcal{B}' r'$  **for**  $\mathcal{V} \mathcal{B} r \mathcal{B}' r'$   
**begin**

**lemma** *combine-rep-number*: *constant-rep-design*  $\mathcal{V}^+ \mathcal{B}^+ (r + r')$   
**using** *combine-rep-number des1.rep-number des2.rep-number* **by** (*unfold-locales*) (*simp*)

**sublocale** *combine-const-rep*: *constant-rep-design*  $\mathcal{V}^+ \mathcal{B}^+ (r + r')$   
**using** *combine-rep-number* **by** *simp*

**end**

**locale** *two-incomplete-designs* = *two-k-block-designs*  $\mathcal{V} \mathcal{B} k \mathcal{V}' \mathcal{B}' +$  *des1*: *incomplete-design*  $\mathcal{V} \mathcal{B} k +$   
*des2*: *incomplete-design*  $\mathcal{V}' \mathcal{B}' k$  **for**  $\mathcal{V} \mathcal{B} k \mathcal{V}' \mathcal{B}'$   
**begin**

**lemma** *combine-is-incomplete*: *incomplete-design*  $\mathcal{V}^+ \mathcal{B}^+ k$   
**using** *combine-order des1.incomplete des2.incomplete* **by** (*unfold-locales*) (*simp*)

**sublocale** *combine-incomplete*: *incomplete-design*  $\mathcal{V}^+ \mathcal{B}^+ k$   
**using** *combine-is-incomplete* **by** *simp*

**end**

**locale** *two-t-designs-eq-points* = *two-incomplete-designs*  $\mathcal{V} \mathcal{B} k \mathcal{V} \mathcal{B}'$   
 $+ \textit{two-t-wise-eq-points} \mathcal{V} \mathcal{B} t \Lambda_t \mathcal{B}' \Lambda_t' +$  *des1*: *t-design*  $\mathcal{V} \mathcal{B} k t \Lambda_t +$   
*des2*: *t-design*  $\mathcal{V} \mathcal{B}' k t \Lambda_t'$  **for**  $\mathcal{V} \mathcal{B} k \mathcal{B}' t \Lambda_t \Lambda_t'$   
**begin**

**lemma** *combine-is-t-des*: *t-design*  $\mathcal{V}^+ \mathcal{B}^+ k t (\Lambda_t + \Lambda_t')$   
**using** *des1.block-size-t des2.block-size-t* **by** (*unfold-locales*)

```

sublocale combine-t-des: t-design  $\mathcal{V}^+ \mathcal{B}^+ k t (\Lambda_t + \Lambda_t')$ 
  using combine-is-t-des by blast

end
end

```

```

theory BIBD imports Block-Designs
begin

```

## 5 BIBD's

BIBD's are perhaps the most commonly studied type of design in combinatorial design theory, and usually the first type of design explored in a design theory course. These designs are a type of  $t$ -design, where  $t = 2$

### 5.1 BIBD Basics

```

locale bibd = t-design  $\mathcal{V} \mathcal{B} k 2 \Lambda$ 
  for point-set  $\langle \mathcal{V} \rangle$  and block-collection  $\langle \mathcal{B} \rangle$ 
  and u-block-size  $\langle k \rangle$  and index  $\langle \Lambda \rangle$ 

```

```

begin

```

```

lemma min-block-size-2:  $k \geq 2$ 
  using block-size-t by simp

```

```

lemma points-index-pair:  $y \in \mathcal{V} \implies x \in \mathcal{V} \implies x \neq y \implies \text{size} (\{\# bl \in \# \mathcal{B} . \{x, y\} \subseteq bl\# \}) = \Lambda$ 
  using balanced card-2-iff empty-subsetI insert-subset points-index-def by metis

```

```

lemma index-one-empty-rm-bl [simp]:
  assumes  $\Lambda = 1$  and  $blv \in \# \mathcal{B}$  and  $p \subseteq blv$  and  $\text{card } p = 2$ 
  shows  $\{\# bl \in \# \text{remove1-mset } blv \mathcal{B} . p \subseteq bl\#\} = \{\#\}$ 
proof –
  have blv-in:  $blv \in \# \text{filter-mset } ((\subseteq) p) \mathcal{B}$ 
    using assms by simp
  have  $p \subseteq \mathcal{V}$  using assms wellformed by auto
  then have  $\text{size} (\text{filter-mset } ((\subseteq) p) \mathcal{B}) = 1$ 
    using balanced assms by (simp add: points-index-def)
  then show ?thesis using blv-in filter-diff-mset filter-single-mset
    by (metis (no-types, lifting) add-mset-eq-single assms(3) insert-DiffM size-1-singleton-mset)

```

```

qed

```

```

lemma index-one-alt-bl-not-exist:
  assumes  $\Lambda = 1$  and  $blv \in \# \mathcal{B}$  and  $p \subseteq blv$  and  $\text{card } p = 2$ 

```

**shows**  $\bigwedge bl. bl \in \# \text{ remove1-mset } bl \ \mathcal{B} \implies \neg (p \subseteq bl)$   
**using** *index-one-empty-rm-blv*  
**by** (*metis* *assms(1)* *assms(2)* *assms(3)* *assms(4)* *filter-mset-empty-conv*)

## 5.2 Necessary Conditions for Existence

The necessary conditions on the existence of a  $(v, k, \lambda)$ -bibd are one of the fundamental first theorems on designs. Proofs based off MATH3301 lecture notes [4] and Stinson [6]

**lemma** *necess-cond-1-rhs*:

**assumes**  $x \in \mathcal{V}$   
**shows**  $\text{size} (\{\# p \in \# (\text{mset-set } (\mathcal{V} - \{x\}) \times \# \{\# bl \in \# \mathcal{B} . x \in bl \#\}) . \text{fst } p \in \text{snd } p\#\}) = \Lambda * (v - 1)$   
**proof** –  
**let**  $?M = \text{mset-set } (\mathcal{V} - \{x\})$   
**let**  $?B = \{\# bl \in \# \mathcal{B} . x \in bl \#\}$   
**have** *m-distinct*: *distinct-mset*  $?M$  **using** *assms* *mset-points-distinct-diff-one* **by** *simp*  
**have** *y-point*:  $\bigwedge y . y \in \# ?M \implies y \in \mathcal{V}$  **using** *assms*  
**by** (*simp* *add*: *finite-sets*)  
**have** *b-contents*:  $\bigwedge bl. bl \in \# ?B \implies x \in bl$  **using** *assms* **by** *auto*  
**have**  $\bigwedge y. y \in \# ?M \implies y \neq x$  **using** *assms*  
**by** (*simp* *add*: *finite-sets*)  
**then have**  $\bigwedge y . y \in \# ?M \implies \text{size} (\{\# bl \in \# ?B . \{x, y\} \subseteq bl\#\}) = \Lambda$   
**using** *points-index-pair* *filter-filter-mset-ss-member* *y-point* *assms* *finite-sets*  
**by** *metis*  
**then have**  $\bigwedge y . y \in \# ?M \implies \text{size} (\{\# bl \in \# ?B . x \in bl \wedge y \in bl\#\}) = \Lambda$   
**by** *auto*  
**then have** *bl-set-size*:  $\bigwedge y . y \in \# ?M \implies \text{size} (\{\# bl \in \# ?B . y \in bl\#\}) = \Lambda$   
**using** *b-contents* **by** (*metis* (*no-types*, *lifting*) *filter-mset-cong*)  
**then have** *final-size*:  $\text{size} (\sum p \in \# ?M . (\{\# p\#\} \times \# \{\# bl \in \# ?B . p \in bl\#\}))$   
 $= \text{size } (?M) * \Lambda$   
**using** *m-distinct* *size-Union-distinct-cart-prod-filter* *bl-set-size* **by** *blast*  
**have**  $\text{size } ?M = v - 1$  **using** *v-non-zero*  
**by** (*simp* *add*: *assms(1)* *finite-sets*)  
**thus** *thesis* **using** *final-size*  
**by** (*simp* *add*: *set-break-down-left*)  
**qed**

**lemma** *necess-cond-1-lhs*:

**assumes**  $x \in \mathcal{V}$   
**shows**  $\text{size} (\{\# p \in \# (\text{mset-set } (\mathcal{V} - \{x\}) \times \# \{\# bl \in \# \mathcal{B} . x \in bl \#\}) . \text{fst } p \in \text{snd } p\#\})$   
 $= (\mathcal{B} \text{ rep } x) * (k - 1)$   
 $(\text{is } \text{size} (\{\# p \in \# (?M \times \# ?B) . \text{fst } p \in \text{snd } p\#\}) = (\mathcal{B} \text{ rep } x) * (k - 1) )$   
**proof** –  
**have**  $\bigwedge y. y \in \# ?M \implies y \neq x$  **using** *assms*  
**by** (*simp* *add*: *finite-sets*)

**have** *distinct-m*: *distinct-mset* ?*M* **using** *assms* *mset-points-distinct-diff-one* **by**  
*simp*  
**have** *finite-M*: *finite* ( $\mathcal{V} - \{x\}$ ) **using** *finite-sets* **by** *auto*  
**have** *block-choices*: *size* ?*B* =  $\mathcal{B}$  *rep* *x*  
**by** (*simp* *add*: *assms*(1) *point-replication-number-def*)  
**have** *bl-size*:  $\forall bl \in \# ?B. \text{card } \{p \in \mathcal{V} . p \in bl\} = k$  **using** *uniform-unfold-point-set*  
**by** *simp*  
**have** *x-in-set*:  $\forall bl \in \# ?B . \{x\} \subseteq \{p \in \mathcal{V} . p \in bl\}$  **using** *assms* **by** *auto*  
**then have**  $\forall bl \in \# ?B. \text{card } \{p \in (\mathcal{V} - \{x\}) . p \in bl\} = \text{card } (\{p \in \mathcal{V} . p \in bl\} - \{x\})$   
**by** (*simp* *add*: *set-filter-diff-card*)  
**then have**  $\forall bl \in \# ?B. \text{card } \{p \in (\mathcal{V} - \{x\}) . p \in bl\} = k - 1$   
**using** *bl-size* *x-in-set* *card-Diff-subset* *finite-sets* *k-non-zero* **by** *auto*  
**then have**  $\bigwedge bl . bl \in \# ?B \implies \text{size } \{\#p \in \# ?M . p \in bl\# \} = k - 1$   
**using** *assms* *finite-M* *card-size-filter-eq* **by** *auto*  
**then have** *size* ( $\sum bl \in \# ?B. (\{\#p \in \# ?M . p \in bl\# \} \times \# \{\#bl\#\})$ ) = *size*  
(*B*) \* (k - 1)  
**using** *distinct-m* *size-Union-distinct-cart-prod-filter2* **by** *blast*  
**thus** ?*thesis* **using** *block-choices* *k-non-zero* **by** (*simp* *add*: *set-break-down-right*)  
**qed**

**lemma** *r-constant*:  $x \in \mathcal{V} \implies (\mathcal{B} \text{ rep } x) * (k - 1) = \Lambda * (v - 1)$   
**using** *necess-cond-1-rhs* *necess-cond-1-lhs* *design-points-nempty* **by** *force*

**lemma** *replication-number-value*:  
**assumes**  $x \in \mathcal{V}$   
**shows**  $(\mathcal{B} \text{ rep } x) = \Lambda * (v - 1) \text{ div } (k - 1)$   
**using** *min-block-size-2* *r-constant* *assms*  
**by** (*metis* *diff-is-0-eq* *diffs0-imp-equal* *div-by-1* *k-non-zero* *nonzero-mult-div-cancel-right*  
*one-div-two-eq-zero* *one-le-numeral* *zero-neq-one*)

**lemma** *r-constant-alt*:  $\forall x \in \mathcal{V}. \mathcal{B} \text{ rep } x = \Lambda * (v - 1) \text{ div } (k - 1)$   
**using** *r-constant* *replication-number-value* **by** *blast*

**end**

Using the first necessary condition, it is possible to show that a bibd has a constant replication number

**sublocale** *bibd*  $\subseteq$  *constant-rep-design*  $\mathcal{V} \mathcal{B} (\Lambda * (v - 1) \text{ div } (k - 1))$   
**using** *r-constant-alt* **by** (*unfold-locales*) *simp-all*

**lemma** (*in t-design*) *bibdI* [*intro*]:  $t = 2 \implies \text{bibd } \mathcal{V} \mathcal{B} k \Lambda_t$   
**using** *t-lt-order* *block-size-t* **by** (*unfold-locales*) (*simp-all*)

**context** *bibd*  
**begin**

**abbreviation**  $r \equiv (\Lambda * (v - 1) \text{ div } (k - 1))$

**lemma necessary-condition-one:**  
**shows**  $r * (k - 1) = \Lambda * (v - 1)$   
**using** *necess-cond-1-rhs* *necess-cond-1-lhs* *design-points-nempty* *rep-number* **by**  
*auto*

**lemma bibd-point-occ-rep:**  
**assumes**  $x \in bl$   
**assumes**  $bl \in \# \mathcal{B}$   
**shows**  $(\mathcal{B} - \{\#bl\}) \text{ rep } x = r - 1$   
**proof** –  
**have**  $xin: x \in \mathcal{V}$  **using** *assms wf-invalid-point* **by** *blast*  
**then have**  $rep: \text{size } \{\# \text{ blk } \in \# \mathcal{B}. x \in \text{blk } \#\} = r$  **using** *rep-number-unfold-set*  
**by** *simp*  
**have**  $(\mathcal{B} - \{\#bl\}) \text{ rep } x = \text{size } \{\# \text{ blk } \in \# (\mathcal{B} - \{\#bl\}). x \in \text{blk } \#\}$   
**by** (*simp add: point-replication-number-def*)  
**then have**  $(\mathcal{B} - \{\#bl\}) \text{ rep } x = \text{size } \{\# \text{ blk } \in \# \mathcal{B}. x \in \text{blk } \#\} - 1$   
**by** (*simp add: assms size-Diff-singleton*)  
**then show** *?thesis* **using** *assms rep r-gzero* **by** *simp*  
**qed**

**lemma necess-cond-2-lhs:**  $\text{size } \{\# x \in \# (\text{mset-set } \mathcal{V} \times \# \mathcal{B}) . (\text{fst } x) \in (\text{snd } x) \#\} = v * r$   
**proof** –  
**let**  $?M = \text{mset-set } \mathcal{V}$   
**have**  $\bigwedge p . p \in \# ?M \implies \text{size } (\{\# bl \in \# \mathcal{B} . p \in bl \#\}) = r$   
**using** *finite-sets rep-number-unfold-set r-gzero nat-eq-iff2* **by** *auto*  
**then have**  $\text{size } (\sum p \in \# ?M. (\{\# p \#\} \times \# \{\# bl \in \# \mathcal{B}. p \in bl \#\})) = \text{size } ?M * (r)$   
**using** *mset-points-distinct size-Union-distinct-cart-prod-filter* **by** *blast*  
**thus** *?thesis* **using** *r-gzero*  
**by** (*simp add: set-break-down-left*)  
**qed**

**lemma necess-cond-2-rhs:**  $\text{size } \{\# x \in \# (\text{mset-set } \mathcal{V} \times \# \mathcal{B}) . (\text{fst } x) \in (\text{snd } x) \#\} = b * k$   
**(is**  $\text{size } \{\# x \in \# (?M \times \# ?B) . (\text{fst } x) \in (\text{snd } x) \#\} = b * k$ **)**  
**proof** –  
**have**  $\bigwedge bl . bl \in \# ?B \implies \text{size } (\{\# p \in \# ?M . p \in bl \#\}) = k$   
**using** *uniform k-non-zero uniform-unfold-point-set-mset* **by** *fastforce*  
**then have**  $\text{size } (\sum bl \in \# ?B. (\{\# p \in \# ?M . p \in bl \#\} \times \# \{\# bl \#\})) = \text{size } (?B) * k$   
**using** *mset-points-distinct size-Union-distinct-cart-prod-filter2* **by** *blast*  
**thus** *?thesis* **using** *k-non-zero* **by** (*simp add: set-break-down-right*)  
**qed**

**lemma necessary-condition-two:**  
**shows**  $v * r = b * k$   
**using** *necess-cond-2-lhs* *necess-cond-2-rhs* **by** *simp*

**theorem** *admissability-conditions*:  
 $r * (k - 1) = \Lambda * (v - 1)$   
 $v * r = b * k$   
**using** *necessary-condition-one necessary-condition-two* **by** *auto*

### 5.2.1 BIBD Param Relationships

**lemma** *bibd-block-number*:  $b = \Lambda * v * (v - 1) \text{ div } (k * (k - 1))$

**proof** –

**have**  $b * k = (v * r)$  **using** *necessary-condition-two* **by** *simp*

**then have** *k-dvd*:  $k \text{ dvd } (v * r)$  **by** (*metis dvd-triv-right*)

**then have**  $b = (v * r) \text{ div } k$  **using** *necessary-condition-two min-block-size-2* **by** *auto*

**then have**  $b = (v * ((\Lambda * (v - 1) \text{ div } (k - 1)))) \text{ div } k$  **by** *simp*

**then have**  $b = (v * \Lambda * (v - 1)) \text{ div } ((k - 1) * k)$  **using** *necessary-condition-one*

*necessary-condition-two dvd-div-div-eq-mult dvd-div-eq-0-iff dvd-triv-right mult.assoc*

*mult.commute mult.left-commute mult-eq-0-iff*

**by** (*smt (verit) b-non-zero*)

**then show** *?thesis* **by** (*simp add: mult.commute*)

**qed**

**lemma** *symmetric-condition-1*:  $\Lambda * (v - 1) = k * (k - 1) \implies b = v \wedge r = k$

**using** *b-non-zero bibd-block-number mult-eq-0-iff necessary-condition-two necessary-condition-one*

**by** *auto*

**lemma** *index-lt-replication*:  $\Lambda < r$

**proof** –

**have** *1*:  $r * (k - 1) = \Lambda * (v - 1)$  **using** *admissability-conditions* **by** *simp*

**have** *lhsnot0*:  $r * (k - 1) \neq 0$

**using** *no-zero-divisors rep-not-zero* **by** (*metis div-by-0*)

**then have** *rhsnot0*:  $\Lambda * (v - 1) \neq 0$  **using** *1* **by** *presburger*

**have**  $k - 1 < v - 1$  **using** *incomplete b-non-zero bibd-block-number not-less-eq* **by** *fastforce*

**thus** *?thesis* **using** *1 lhsnot0 rhsnot0 k-non-zero mult-le-less-imp-less r-gzero*

**by** (*metis div-greater-zero-iff less-or-eq-imp-le nat-less-le nat-neq-iff*)

**qed**

**lemma** *index-not-zero*:  $\Lambda \geq 1$

**by** (*metis div-0 leI less-one mult-not-zero rep-not-zero*)

**lemma** *r-ge-two*:  $r \geq 2$

**using** *index-lt-replication index-not-zero* **by** *linarith*

**lemma** *block-num-gt-rep*:  $b > r$

**proof** –

**have** *fact*:  $b * k = v * r$  **using** *admissability-conditions* **by** *auto*  
**have** *lhsnot0*:  $b * k \neq 0$  **using** *k-non-zero b-non-zero* **by** *auto*  
**then have** *rhsnot0*:  $v * r \neq 0$  **using** *fact* **by** *simp*  
**then show** *?thesis* **using** *incomplete lhsnot0*  
**using** *complement-rep-number constant-rep-design.r-gzero incomplete-imp-incomp-block*  
**by** *fastforce*  
**qed**

**lemma** *bibd-subset-occ*:

**assumes**  $x \subseteq bl$  **and**  $bl \in \# \mathcal{B}$  **and**  $card\ x = 2$   
**shows**  $size\ \{\#\ blk \in \# (\mathcal{B} - \{\#bl\#\}) . x \subseteq blk\ \# \} = \Lambda - 1$   
**proof** –  
**have** *index*:  $size\ \{\#\ blk \in \# \mathcal{B} . x \subseteq blk\ \# \} = \Lambda$  **using** *points-index-def balanced*  
*assms*  
**by** (*metis (full-types) subset-eq wf-invalid-point*)  
**then have**  $size\ \{\#\ blk \in \# (\mathcal{B} - \{\#bl\#\}) . x \subseteq blk\ \# \} = size\ \{\#\ blk \in \# \mathcal{B} . x \subseteq blk\ \# \} - 1$   
**by** (*simp add: assms size-Diff-singleton*)  
**then show** *?thesis* **using** *assms index-not-zero index* **by** *simp*  
**qed**

**lemma** *necess-cond-one-param-balance*:  $b > v \implies r > k$

**using** *necessary-condition-two b-positive*  
**by** (*metis div-le-mono2 div-mult-self1-is-m div-mult-self-is-m nat-less-le r-gzero v-non-zero*)

### 5.3 Constructing New bibd's

There are many constructions on bibd's to establish new bibds (or other types of designs). This section demonstrates this using both existing constructions, and by defining new constructions.

#### 5.3.1 BIBD Complement, Multiple, Combine

**lemma** *comp-params-index-pair*:

**assumes**  $\{x, y\} \subseteq \mathcal{V}$   
**assumes**  $x \neq y$   
**shows**  $\mathcal{B}^C\ index\ \{x, y\} = b + \Lambda - 2*r$   
**proof** –  
**have** *xin*:  $x \in \mathcal{V}$  **and** *yin*:  $y \in \mathcal{V}$  **using** *assms* **by** *auto*  
**have** *ge*:  $2*r \geq \Lambda$  **using** *index-1t-replication*  
**using** *r-gzero* **by** *linarith*  
**have** *lambda*:  $size\ \{\#\ b \in \# \mathcal{B} . x \in b \wedge y \in b\ \# \} = \Lambda$  **using** *points-index-pair*  
*assms* **by** *simp*  
**have** *s1*:  $\mathcal{B}^C\ index\ \{x, y\} = size\ \{\#\ b \in \# \mathcal{B} . x \notin b \wedge y \notin b\ \# \}$   
**using** *complement-index-2 assms* **by** *simp*  
**also have** *s2*:  $\dots = size\ \mathcal{B} - (size\ \{\#\ b \in \# \mathcal{B} . \neg (x \notin b \wedge y \notin b)\ \# \})$   
**using** *size-filter-neg* **by** *blast*  
**also have**  $\dots = size\ \mathcal{B} - (size\ \{\#\ b \in \# \mathcal{B} . x \in b \vee y \in b\ \# \})$  **by** *auto*

**also have**  $\dots = b - (\text{size } \{\# b \in \# \mathcal{B} . x \in b \vee y \in b\# \})$  **by** (*simp add: of-nat-diff*)  
**finally have**  $\mathcal{B}^C \text{ index } \{x, y\} = b - (\text{size } \{\# b \in \# \mathcal{B} . x \in b\# \} + \text{size } \{\# b \in \# \mathcal{B} . y \in b\# \} - \text{size } \{\# b \in \# \mathcal{B} . x \in b \wedge y \in b\# \})$   
**by** (*simp add: mset-size-partition-dep s2 s1*)  
**then have**  $\mathcal{B}^C \text{ index } \{x, y\} = b - (r + r - \Lambda)$  **using** *rep-number-unfold-set lambda xin yin*  
**by** *presburger*  
**then have**  $\mathcal{B}^C \text{ index } \{x, y\} = b - (2*r - \Lambda)$   
**using** *index-lt-replication* **by** (*metis mult-2*)  
**thus** *?thesis* **using** *ge diff-diff-right* **by** *simp*  
**qed**

**lemma** *complement-bibd-index:*

**assumes**  $ps \subseteq \mathcal{V}$   
**assumes**  $\text{card } ps = 2$   
**shows**  $\mathcal{B}^C \text{ index } ps = b + \Lambda - 2*r$   
**proof** –  
**obtain**  $x y$  **where**  $set: ps = \{x, y\}$  **using** *b-non-zero bibd-block-number diff-is-0-eq incomplete*  
*mult-0-right nat-less-le design-points-nempty assms* **by** (*metis card-2-iff*)  
**then have**  $x \neq y$  **using** *assms* **by** *auto*  
**thus** *?thesis* **using** *comp-params-index-pair assms*  
**by** (*simp add: set*)  
**qed**

**lemma** *complement-bibd:*

**assumes**  $k \leq v - 2$   
**shows**  $\text{bibd } \mathcal{V} \mathcal{B}^C (v - k) (b + \Lambda - 2*r)$   
**proof** –  
**interpret** *des: incomplete-design*  $\mathcal{V} \mathcal{B}^C (v - k)$   
**using** *assms complement-incomplete* **by** *blast*  
**show** *?thesis* **proof** (*unfold-locales, simp-all*)  
**show**  $2 \leq \text{des.v}$  **using** *assms block-size-t* **by** *linarith*  
**show**  $\bigwedge ps. ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B}^C \text{ index } ps = b + \Lambda - 2 * (\Lambda * (\text{des.v} - \text{Suc } 0) \text{ div } (k - \text{Suc } 0))$   
**using** *complement-bibd-index* **by** *simp*  
**show**  $2 \leq \text{des.v} - k$  **using** *assms block-size-t* **by** *linarith*  
**qed**  
**qed**

**lemma** *multiple-bibd:  $n > 0 \implies \text{bibd } \mathcal{V} (\text{multiple-blocks } n) k (\Lambda * n)$*   
**using** *multiple-t-design* **by** (*simp add: bibd-def*)

**end**

**locale** *two-bibd-eq-points = two-t-designs-eq-points*  $\mathcal{V} \mathcal{B} k \mathcal{B}' 2 \Lambda \Lambda'$   
 $+ \text{des1: bibd } \mathcal{V} \mathcal{B} k \Lambda + \text{des2: bibd } \mathcal{V} \mathcal{B}' k \Lambda'$  **for**  $\mathcal{V} \mathcal{B} k \mathcal{B}' \Lambda \Lambda'$   
**begin**

**lemma** *combine-is-bibd*:  $\text{bibd } \mathcal{V}^+ \mathcal{B}^+ k (\Lambda + \Lambda')$   
**by** (*unfold-locales*)

**sublocale** *combine-bibd*:  $\text{bibd } \mathcal{V}^+ \mathcal{B}^+ k (\Lambda + \Lambda')$   
**by** (*unfold-locales*)

**end**

### 5.3.2 Derived Designs

A derived bibd takes a block from a valid bibd as the new point sets, and the intersection of that block with other blocks as it's block set

**locale** *bibd-block-transformations* = *bibd* +  
**fixes** *block* :: 'a set ( $\langle \text{bl} \rangle$ )  
**assumes** *valid-block*:  $\text{bl} \in \# \mathcal{B}$   
**begin**

**definition** *derived-blocks* :: 'a set multiset ( $\langle \langle \mathcal{B}^D \rangle \rangle$ ) **where**  
 $\mathcal{B}^D \equiv \{ \# \text{bl} \cap b . b \in \# (\mathcal{B} - \{ \# \text{bl} \# \}) \# \}$

**lemma** *derive-define-flip*:  $\{ \# b \cap \text{bl} . b \in \# (\mathcal{B} - \{ \# \text{bl} \# \}) \# \} = \mathcal{B}^D$   
**by** (*simp add: derived-blocks-def inf-sup-aci(1)*)

**lemma** *derived-points-order*:  $\text{card } \text{bl} = k$   
**using** *uniform valid-block* **by** *simp*

**lemma** *derived-block-num*:  $\text{bl} \in \# \mathcal{B} \implies \text{size } \mathcal{B}^D = k - 1$   
**by** (*simp add: derived-blocks-def size-remove1-mset-If valid-block*)

**lemma** *derived-is-wellformed*:  $b \in \# \mathcal{B}^D \implies b \subseteq \text{bl}$   
**by** (*simp add: derived-blocks-def valid-block*) (*auto*)

**lemma** *derived-point-subset-orig*:  $ps \subseteq \text{bl} \implies ps \subset \mathcal{V}$   
**by** (*simp add: valid-block incomplete-imp-proper-subset subset-psubset-trans*)

**lemma** *derived-obtain-orig-block*:  
**assumes**  $b \in \# \mathcal{B}^D$   
**obtains** *b2* **where**  $b = b2 \cap \text{bl}$  **and**  $b2 \in \# \text{remove1-mset } \text{bl } \mathcal{B}$   
**using** *assms derived-blocks-def* **by** *auto*

**sublocale** *derived-incidence-sys*: *incidence-system*  $\text{bl } \mathcal{B}^D$   
**using** *derived-is-wellformed valid-block* **by** (*unfold-locales*) (*auto*)

**sublocale** *derived-fin-incidence-system*: *finite-incidence-system*  $\text{bl } \mathcal{B}^D$   
**using** *valid-block finite-blocks* **by** (*unfold-locales*) *simp-all*

**lemma** *derived-blocks-nempty*:  
**assumes**  $\bigwedge b . b \in \# \text{remove1-mset } \text{bl } \mathcal{B} \implies \text{bl } |\cap| b > 0$

**assumes**  $bl \in \# \mathcal{B}^D$   
**shows**  $bl \neq \{\}$   
**proof** –  
**obtain**  $bl2$  **where**  $inter: bl = bl2 \cap bl$  **and**  $member: bl2 \in \# \text{remove1-mset } bl$   
 $\mathcal{B}$   
**using** *assms derived-obtain-orig-block* **by** *blast*  
**then have**  $bl \cap bl2 > 0$  **using** *assms(1)* **by** *blast*  
**thus** *?thesis using intersection-number-empty-iff finite-blocks valid-block*  
**by** (*metis Int-commute dual-order.irrefl inter*)  
**qed**

**lemma** *derived-is-design*:  
**assumes**  $\bigwedge b. b \in \# \text{remove1-mset } bl \mathcal{B} \implies bl \cap b > 0$   
**shows** *design*  $bl \mathcal{B}^D$   
**proof** –  
**interpret** *fin: finite-incidence-system*  $bl \mathcal{B}^D$   
**by** (*unfold-locales*)  
**show** *?thesis using assms derived-blocks-nempty* **by** (*unfold-locales*) *simp*  
**qed**

**lemma** *derived-is-proper*:  
**assumes**  $\bigwedge b. b \in \# \text{remove1-mset } bl \mathcal{B} \implies bl \cap b > 0$   
**shows** *proper-design*  $bl \mathcal{B}^D$   
**proof** –  
**interpret** *des: design*  $bl \mathcal{B}^D$   
**using** *derived-is-design assms* **by** *fastforce*  
**have**  $b - 1 > 1$  **using** *block-num-gt-rep r-ge-two* **by** *linarith*  
**then show** *?thesis* **by** (*unfold-locales*) (*simp add: derived-block-num valid-block*)  
**qed**

### 5.3.3 Residual Designs

Similar to derived designs, a residual design takes the complement of a block  $bl$  as it's new point set, and the complement of all other blocks with respect to  $bl$ .

**definition** *residual-blocks* :: 'a set multiset  $\langle (\mathcal{B}^R) \rangle$  **where**  
 $\mathcal{B}^R \equiv \{\# b - bl . b \in \# (\mathcal{B} - \{\#bl\}) \# \}$

**lemma** *residual-order*:  $\text{card } (bl^c) = v - k$   
**by** (*simp add: valid-block wellformed block-complement-size*)

**lemma** *residual-block-num*:  $\text{size } (\mathcal{B}^R) = b - 1$   
**using** *b-positive* **by** (*simp add: residual-blocks-def size-remove1-mset-If valid-block int-ops(6)*)

**lemma** *residual-obtain-orig-block*:  
**assumes**  $b \in \# \mathcal{B}^R$   
**obtains**  $bl2$  **where**  $b = bl2 - bl$  **and**  $bl2 \in \# \text{remove1-mset } bl \mathcal{B}$   
**using** *assms residual-blocks-def* **by** *auto*

**lemma** *residual-blocks-ss*: **assumes**  $b \in \# \mathcal{B}^R$  **shows**  $b \subseteq \mathcal{V}$   
**proof** –  
    **have**  $b \subseteq (bl^c)$  **using** *residual-obtain-orig-block*  
    **by** (*metis Diff-mono assms block-complement-def in-diffD order-refl wellformed*)  
    **thus** *?thesis*  
    **using** *block-complement-subset-points* **by** *auto*  
**qed**

**lemma** *residual-blocks-exclude*:  $b \in \# \mathcal{B}^R \implies x \in b \implies x \notin bl$   
**using** *residual-obtain-orig-block* **by** *auto*

**lemma** *residual-is-wellformed*:  $b \in \# \mathcal{B}^R \implies b \subseteq (bl^c)$   
**apply** (*auto simp add: residual-blocks-def*)  
**by** (*metis DiffI block-complement-def in-diffD wf-invalid-point*)

**sublocale** *residual-incidence-sys*: *incidence-system*  $bl^c \mathcal{B}^R$   
**using** *residual-is-wellformed* **by** (*unfold-locales*)

**lemma** *residual-is-finite*: *finite*  $(bl^c)$   
**by** (*simp add: block-complement-def finite-sets*)

**sublocale** *residual-fin-incidence-sys*: *finite-incidence-system*  $bl^c \mathcal{B}^R$   
**using** *residual-is-finite* **by** (*unfold-locales*)

**lemma** *residual-blocks-nempty*:  
**assumes**  $bld \in \# \mathcal{B}^R$   
**assumes** *multiplicity*  $bl = 1$   
**shows**  $bld \neq \{\}$   
**proof** –  
    **obtain**  $bl2$  **where** *inter*:  $bld = bl2 - bl$  **and** *member*:  $bl2 \in \# \text{remove1-mset } bl$   
     $\mathcal{B}$   
    **using** *assms residual-blocks-def* **by** *auto*  
    **then have** *ne*:  $bl2 \neq bl$  **using** *assms*  
    **by** (*metis count-eq-zero-iff in-diff-count less-one union-single-eq-member*)  
    **have**  $\text{card } bl2 = \text{card } bl$  **using** *uniform valid-block member*  
    **using** *in-diffD* **by** *fastforce*  
    **then have**  $\text{card } (bl2 - bl) > 0$   
    **using** *finite-blocks member uniform set-card-diff-ge-zero valid-block* **by** (*metis in-diffD ne*)  
    **thus** *?thesis* **using** *inter* **by** *fastforce*  
**qed**

**lemma** *residual-is-design*: *multiplicity*  $bl = 1 \implies \text{design } (bl^c) \mathcal{B}^R$   
**using** *residual-blocks-nempty* **by** (*unfold-locales*)

**lemma** *residual-is-proper*:  
**assumes** *multiplicity*  $bl = 1$   
**shows** *proper-design*  $(bl^c) \mathcal{B}^R$

```

proof –
  interpret des: design blc  $\mathcal{B}^R$  using residual-is-design assms by blast
  have  $b - 1 > 1$  using r-ge-two block-num-gt-rep by linarith
  then show ?thesis using residual-block-num by (unfold-locales) auto
qed

end

```

## 5.4 Symmetric BIBD's

Symmetric bibd's are those where the order of the design equals the number of blocks

```

locale symmetric-bibd = bibd +
  assumes symmetric: b = v
begin

```

```

lemma rep-value-sym: r = k
  using b-non-zero local.symmetric necessary-condition-two by auto

```

```

lemma symmetric-condition-2:  $\Lambda * (v - 1) = k * (k - 1)$ 
  using necessary-condition-one rep-value-sym by auto

```

```

lemma sym-design-vk-gt-kl:
  assumes  $k \geq \Lambda + 2$ 
  shows  $v - k > k - \Lambda$ 

```

```

proof (rule ccontr)

```

```

  define k l v where kdef: k  $\equiv$  int k and ldef: l  $\equiv$  int  $\Lambda$  and vdef: v  $\equiv$  int v

```

```

  assume  $\neg (v - k > k - \Lambda)$ 

```

```

  then have a:  $\neg (v - k > k - l)$  using kdef ldef vdef

```

```

  by (metis block-size-lt-v index-lt-replication less-imp-le-nat of-nat-diff of-nat-less-imp-less)

```

```

    rep-value-sym)

```

```

  have lge:  $l \geq 0$  using ldef by simp

```

```

  have sym:  $l * (v - 1) = k * (k - 1)$ 

```

```

    using symmetric-condition-2 ldef vdef kdef

```

```

    by (metis (mono-tags, lifting) block-size-lt-v int-ops(2) k-non-zero le-trans of-nat-diff of-nat-mult)

```

```

  then have  $v \leq 2 * k - l$  using a by linarith

```

```

  then have  $v - 1 \leq 2 * k - l - 1$  by linarith

```

```

  then have  $l * (v - 1) \leq l * (2 * k - l - 1)$ 

```

```

    using lge mult-le-cancel-left by fastforce

```

```

  then have  $k * (k - 1) \leq l * (2 * k - l - 1)$ 

```

```

    by (simp add: sym)

```

```

  then have  $k * (k - 1) - l * (2 * k - l - 1) \leq 0$  by linarith

```

```

  then have  $k^2 - k - l * 2 * k + l^2 + l \leq 0$ 

```

```

    by (simp add: mult-ac right-diff-distrib' power2-eq-square)

```

```

  then have  $(k - l) * (k - l - 1) \leq 0$ 

```

```

    by (simp add: mult-ac right-diff-distrib' power2-eq-square)

```

```

  then have  $k = l \vee k = l + 1$ 

```

```

    using mult-le-0-iff by force
    thus False using assms kdef ldef by auto
qed

end

```

```

context bibd
begin

```

```

lemma symmetric-bibdI: b = v  $\implies$  symmetric-bibd  $\mathcal{V}$   $\mathcal{B}$  k  $\Lambda$ 
  by unfold-locales simp

```

```

lemma symmetric-bibdII:  $\Lambda * (v - 1) = k * (k - 1) \implies$  symmetric-bibd  $\mathcal{V}$   $\mathcal{B}$  k  $\Lambda$ 
  using symmetric-condition-1 by unfold-locales blast

```

```

lemma symmetric-not-admissable:  $\Lambda * (v - 1) \neq k * (k - 1) \implies \neg$  symmetric-bibd  $\mathcal{V}$   $\mathcal{B}$  k  $\Lambda$ 
  using symmetric-bibd.symmetric-condition-2 by blast
end

```

```

context symmetric-bibd
begin

```

#### 5.4.1 Intersection Property on Symmetric BIBDs

Below is a proof of an important property on symmetric BIBD's regarding the equivalence of intersection numbers and the design index. This is an intuitive counting proof, and involved significantly more work in a formal environment. Based of Lecture Note [4]

```

lemma intersect-mult-set-eq-block:
  assumes blv  $\in \# \mathcal{B}$ 
  shows  $p \in \# \sum \# \{ \# \text{mset-set } (bl \cap blv) . bl \in \# (\mathcal{B} - \{ \#blv\# \}) \# \} \longleftrightarrow p \in blv$ 
proof (auto, simp add: assms finite-blocks)
  assume assm:  $p \in blv$ 
  then have  $(\mathcal{B} - \{ \#blv\# \}) \text{ rep } p > 0$  using bibd-point-occ-rep r-ge-two assms
  by auto
  then obtain bl where  $bl \in \# (\mathcal{B} - \{ \#blv\# \}) \wedge p \in bl$  using assms rep-number-g0-exists
  by metis
  then show  $\exists x \in \# \text{remove1-mset } blv \mathcal{B} . p \in \# \text{mset-set } (x \cap blv)$ 
    using assms assm finite-blocks by auto
qed

```

```

lemma intersect-mult-set-block-subset-iff:
  assumes blv  $\in \# \mathcal{B}$ 
  assumes  $p \in \# \sum \# \{ \# \text{mset-set } \{ y . y \subseteq blv \cap b2 \wedge \text{card } y = 2 \} . b2 \in \# (\mathcal{B} - \{ \#blv\# \}) \# \}$ 
  shows  $p \subseteq blv$ 
proof (rule subsetI)

```

**fix**  $x$   
**assume**  $asm: x \in p$   
**obtain**  $b2$  **where**  $p \in \# \text{ mset-set } \{y . y \subseteq blv \cap b2 \wedge \text{card } y = 2\} \wedge b2 \in \#(\mathcal{B} - \{\#blv\#})$   
**using**  $assms$  **by**  $blast$   
**then have**  $p \subseteq blv \cap b2$   
**by** ( $metis$  ( $no-types$ ,  $lifting$ )  $elem\text{-mset-set equals0D infinite-set-mset-mset-set mem-Collect-eq$ )  
**thus**  $x \in blv$  **using**  $asm$  **by**  $auto$   
**qed**

**lemma**  $intersect\text{-mult-set-block-subset-card}$ :  
**assumes**  $blv \in \# \mathcal{B}$   
**assumes**  $p \in \# \sum \# \{\# \text{ mset-set } \{y . y \subseteq blv \cap b2 \wedge \text{card } y = 2\} . b2 \in \# (\mathcal{B} - \{\#blv\#})\# \}$   
**shows**  $\text{card } p = 2$   
**proof** –  
**obtain**  $b2$  **where**  $p \in \# \text{ mset-set } \{y . y \subseteq blv \cap b2 \wedge \text{card } y = 2\} \wedge b2 \in \#(\mathcal{B} - \{\#blv\#})$   
**using**  $assms$  **by**  $blast$   
**thus**  $?thesis$   
**by** ( $metis$  ( $mono-tags$ ,  $lifting$ )  $elem\text{-mset-set equals0D infinite-set-mset-mset-set mem-Collect-eq$ )  
**qed**

**lemma**  $intersect\text{-mult-set-block-with-point-exists}$ :  
**assumes**  $blv \in \# \mathcal{B}$  **and**  $p \subseteq blv$  **and**  $\Lambda \geq 2$  **and**  $\text{card } p = 2$   
**shows**  $\exists x \in \# \text{ remove1-mset } blv \mathcal{B} . p \in \# \text{ mset-set } \{y . y \subseteq blv \wedge y \subseteq x \wedge \text{card } y = 2\}$   
**proof** –  
**have**  $\text{size } \{\#b \in \# \mathcal{B} . p \subseteq b\# \} = \Lambda$  **using**  $points\text{-index-def assms}$   
**by** ( $metis$   $balanced\text{-alt-def-all dual-order.trans wellformed}$ )  
**then have**  $\text{size } \{\#bl \in \# (\mathcal{B} - \{\#blv\#}) . p \subseteq bl\# \} \geq 1$   
**using**  $assms$  **by** ( $simp$   $add: size\text{-Diff-singleton}$ )  
**then obtain**  $bl$  **where**  $bl \in \# (\mathcal{B} - \{\#blv\#}) \wedge p \subseteq bl$  **using**  $assms$   $filter\text{-mset-empty-conv}$   
**by** ( $metis$   $diff\text{-diff-cancel diff-is-0-eq' le-numeral-extra(4) size-empty zero-neq-one$ )  
  
**thus**  $?thesis$   
**using**  $assms$   $finite\text{-blocks}$  **by**  $auto$   
**qed**

**lemma**  $intersect\text{-mult-set-block-subset-iff-2}$ :  
**assumes**  $blv \in \# \mathcal{B}$  **and**  $p \subseteq blv$  **and**  $\Lambda \geq 2$  **and**  $\text{card } p = 2$   
**shows**  $p \in \# \sum \# \{\# \text{ mset-set } \{y . y \subseteq blv \cap b2 \wedge \text{card } y = 2\} . b2 \in \# (\mathcal{B} - \{\#blv\#})\# \}$   
**by** ( $auto$   $simp$   $add: intersect\text{-mult-set-block-with-point-exists assms}$ )

**lemma**  $sym\text{-sum-mset-inter-sets-count}$ :

**assumes**  $bl \in \# \mathcal{B}$   
**assumes**  $p \in bl$   
**shows**  $count (\sum \# \{ \# mset-set (bl \cap blv) . bl \in \# (\mathcal{B} - \{ \# blv \# \}) \# \}) p = r - 1$   
(is  $count (\sum \# ?M) p = r - 1$ )  
**proof** –  
**have**  $size-inter: size \{ \# mset-set (bl \cap blv) \mid bl \in \# (\mathcal{B} - \{ \# blv \# \}) . p \in bl \# \}$   
 $= r - 1$   
**using** *bibd-point-occ-rep point-replication-number-def*  
**by** (*metis assms(1) assms(2) size-image-mset*)  
**have**  $inter-finite: \forall bl \in \# (\mathcal{B} - \{ \# blv \# \}) . finite (bl \cap blv)$   
**by** (*simp add: assms(1) finite-blocks*)  
**have**  $\bigwedge bl . bl \in \# (\mathcal{B} - \{ \# blv \# \}) \implies p \in bl \longrightarrow count (mset-set (bl \cap blv)) p$   
 $= 1$   
**using** *assms count-mset-set(1) inter-finite by simp*  
**then have**  $\bigwedge bl . bl \in \# \{ \# b1 \in \# (\mathcal{B} - \{ \# blv \# \}) . p \in b1 \# \} \implies count (mset-set$   
 $(bl \cap blv)) p = 1$   
**by** (*metis (full-types) count-eq-zero-iff count-filter-mset*)  
**then have**  $pin: \bigwedge P . P \in \# \{ \# mset-set (bl \cap blv) \mid bl \in \# (\mathcal{B} - \{ \# blv \# \}) . p$   
 $\in bl \# \}$   
 $\implies count P p = 1$  **by** *blast*  
**have**  $?M = \{ \# mset-set (bl \cap blv) \mid bl \in \# (\mathcal{B} - \{ \# blv \# \}) . p \in bl \# \}$   
 $+ \{ \# mset-set (bl \cap blv) \mid bl \in \# (\mathcal{B} - \{ \# blv \# \}) . p \notin bl \# \}$   
**by** (*metis image-mset-union multiset-partition*)  
**then have**  $count (\sum \# ?M) p = size \{ \# mset-set (bl \cap blv) \mid bl \in \# (\mathcal{B} -$   
 $\{ \# blv \# \}) . p \in bl \# \}$   
**using** *pin by (auto simp add: count-sum-mset)*  
**then show** *?thesis using size-inter by linarith*  
**qed**

**lemma** *sym-sum-mset-inter-sets-size:*

**assumes**  $blv \in \# \mathcal{B}$   
**shows**  $size (\sum \# \{ \# mset-set (bl \cap blv) . bl \in \# (\mathcal{B} - \{ \# blv \# \}) \# \}) = k * (r - 1)$   
(is  $size (\sum \# ?M) = k * (r - 1)$ )

**proof** –

**have**  $eq: set-mset (\sum \# \{ \# mset-set (bl \cap blv) . bl \in \# (\mathcal{B} - \{ \# blv \# \}) \# \}) = blv$   
**using** *intersect-mult-set-eq-block assms by auto*  
**then have**  $k: card (set-mset (\sum \# ?M)) = k$   
**by** (*simp add: assms*)  
**have**  $\bigwedge p . p \in \# (\sum \# ?M) \implies count (\sum \# ?M) p = r - 1$   
**using** *sym-sum-mset-inter-sets-count assms eq by blast*  
**thus** *?thesis using k size-multiset-set-mset-const-count by metis*  
**qed**

**lemma** *sym-sum-inter-num:*

**assumes**  $b1 \in \# \mathcal{B}$   
**shows**  $(\sum b2 \in \# (\mathcal{B} - \{ \# b1 \# \}) . b1 \mid \cap \mid b2) = k * (r - 1)$

**proof** –

**have**  $(\sum b2 \in \# (\mathcal{B} - \{ \# b1 \# \}) . b1 \mid \cap \mid b2) = (\sum b2 \in \# (\mathcal{B} - \{ \# b1 \# \}) . size$

```

(mset-set (b1 ∩ b2)))
  by (simp add: intersection-number-def)
  also have ... = size (∑ # {#mset-set (b1 ∩ bl). bl ∈ # (B - {#b1#})#})
  by (auto simp add: size-big-union-sum)
  also have ... = size (∑ # {#mset-set (bl ∩ b1). bl ∈ # (B - {#b1#})#})
  by (metis Int-commute)
  finally have (∑ b2 ∈ # (B - {#b1#}). b1 |∩| b2) = k * (r - 1)
  using sym-sum-mset-inter-sets-size assms by auto
  then show ?thesis by simp
qed

```

**lemma** *sym-sum-mset-inter2-sets-count*:

```

  assumes blv ∈ # B
  assumes p ⊆ blv
  assumes card p = 2
  shows count (∑ # {#mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2}. b2 ∈ # (B -
  {#blv#})#}) p = Λ - 1
  (is count (∑ # ?M) p = Λ - 1)
proof -
  have size-inter: size {# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} | b2 ∈ # (B -
  {#blv#}) . p ⊆ b2#}
    = Λ - 1
  using bibd-subset-occ assms by simp
  have ∀ b2 ∈ # (B - {#blv#}) . p ⊆ b2 → count (mset-set {y . y ⊆ blv ∩ b2 ∧
  card y = 2}) p = 1
  using assms(2) count-mset-set(1) assms(3) by (auto simp add: assms(1)
  finite-blocks)
  then have ∀ bl ∈ # {#b1 ∈ # (B - {#blv#}) . p ⊆ b1#}.
    count (mset-set {y . y ⊆ blv ∩ bl ∧ card y = 2}) p = 1
  using count-eq-zero-iff count-filter-mset by (metis (no-types, lifting))
  then have pin: ∀ P ∈ # {# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} | b2 ∈ #
  (B - {#blv#}) . p ⊆ b2#}.
    count P p = 1
  using count-eq-zero-iff count-filter-mset by blast
  have ?M = {# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} | b2 ∈ # (B - {#blv#})
  . p ⊆ b2#} +
    {# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} | b2 ∈ # (B - {#blv#})
  . ¬ (p ⊆ b2)#}
  by (metis image-mset-union multiset-partition)
  then have count (∑ # ?M) p =
    size {# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} | b2 ∈ # (B - {#blv#}) .
  p ⊆ b2#}
  using pin by (auto simp add: count-sum-mset)
  then show ?thesis using size-inter by linarith
qed

```

**lemma** *sym-sum-mset-inter2-sets-size*:

```

  assumes blv ∈ # B
  shows size (∑ # {#mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2}. b2 ∈ # (B -

```

```

{#blv#})#}) =
  (k choose 2) * (Λ - 1)
  (is size (∑ # ?M) = (k choose 2) * (Λ - 1))
proof (cases Λ = 1)
  case True
  have empty: ∧ b2 . b2 ∈# remove1-mset blv B ⇒ {y . y ⊆ blv ∧ y ⊆ b2 ∧ card
y = 2} = {}
  using index-one-alt-bl-not-exist assms True by blast
  then show ?thesis using sum-mset.neutral True by (simp add: empty)
next
  case False
  then have index-min: Λ ≥ 2 using index-not-zero by linarith
  have subset-card: ∧ x . x ∈# (∑ # ?M) ⇒ card x = 2
  proof -
  fix x
  assume a: x ∈# (∑ # ?M)
  then obtain b2 where x ∈# mset-set {y . y ⊆ blv ∩ b2 ∧ card y = 2} ∧ b2
∈#(B - {#blv#})
  by blast
  thus card x = 2 using mem-Collect-eq
  by (metis (mono-tags, lifting) elem-mset-set equals0D infinite-set-mset-mset-set)
qed
  have eq: set-mset (∑ # ?M) = {bl . bl ⊆ blv ∧ card bl = 2}
  proof
  show set-mset (∑ # ?M) ⊆ {bl . bl ⊆ blv ∧ card bl = 2}
  using subset-card intersect-mult-set-block-subset-iff assms by blast
  show {bl . bl ⊆ blv ∧ card bl = 2} ⊆ set-mset (∑ # ?M)
  using intersect-mult-set-block-subset-iff-2 assms index-min by blast
qed
  have card blv = k using uniform assms by simp
  then have k: card (set-mset (∑ # ?M)) = (k choose 2) using eq n-subsets
  by (simp add: n-subsets assms finite-blocks)
  thus ?thesis using k size-multiset-set-mset-const-count sym-sum-mset-inter2-sets-count
assms eq
  by (metis (no-types, lifting) intersect-mult-set-block-subset-iff subset-card)
qed

```

**lemma** *sum-choose-two-inter-num:*

```

assumes b1 ∈# B
shows (∑ b2 ∈# (B - {#b1#}). ((b1 |∩| b2) choose 2)) = ((Λ * (Λ - 1) div
2)) * (v - 1)
proof -
  have div-fact: 2 dvd (Λ * (Λ - 1))
  by fastforce
  have div-fact-2: 2 dvd (Λ * (v - 1)) using symmetric-condition-2 by fastforce
  have (∑ b2 ∈# (B - {#b1#}). ((b1 |∩| b2) choose 2)) = (∑ b2 ∈# (B -
{#b1#}). (b1 |∩|2 b2))
  using n-inter-num-choose-design-inter assms by (simp add: in-diffD)
  then have sum-fact: (∑ b2 ∈# (B - {#b1#}). ((b1 |∩| b2) choose 2))

```

$= (k \text{ choose } 2) * (\Lambda - 1)$   
**using** *assms sym-sum-mset-inter2-sets-size*  
**by** (*auto simp add: size-big-union-sum n-intersect-num-subset-def*)  
**have**  $(k \text{ choose } 2) * (\Lambda - 1) = ((\Lambda * (v - 1) \text{ div } 2)) * (\Lambda - 1)$   
**using** *choose-two symmetric-condition-2 k-non-zero* **by** *auto*  
**also have**  $\dots = ((\Lambda * (\Lambda - 1) \text{ div } 2)) * (v - 1)$   
**using** *div-fact div-fact-2*  
**by** (*metis (no-types, lifting) ab-semigroup-mult-class.mult-ac(1) dvd-div-mult mult.commute*)  
**finally have**  $(k \text{ choose } 2) * (\Lambda - 1) = ((\Lambda * (\Lambda - 1) \text{ div } 2)) * (v - 1)$ .  
**then show** *?thesis* **using** *sum-fact* **by** *simp*  
**qed**

**lemma** *sym-sum-inter-num-sq*:

**assumes**  $b1 \in \# \mathcal{B}$   
**shows**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl) \wedge 2) = \Lambda \wedge 2 * (v - 1)$   
**proof** –  
**have** *dvd: 2 dvd ((v - 1) \* (\Lambda \* (\Lambda - 1)))* **by** *fastforce*  
**have** *inner-dvd:  $\forall bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). 2 \text{ dvd } ((b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1))$*   
– *)*  
**by** *force*  
**have** *diff-le:  $\bigwedge bl. bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}) \implies (b1 \mid \cap \mid bl) \leq (b1 \mid \cap \mid bl) \wedge 2$*   
**by** (*simp add: power2-nat-le-imp-le*)  
**have** *a:  $(\sum b2 \in \# (\mathcal{B} - \{b1\}). ((b1 \mid \cap \mid b2) \text{ choose } 2)) = (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). ((b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1)) \text{ div } 2)$*   
**using** *choose-two* **by** (*simp add: intersection-num-non-neg*)  
**have** *b:  $(\sum b2 \in \# (\mathcal{B} - \{b1\}). ((b1 \mid \cap \mid b2) \text{ choose } 2)) = (\sum b2 \in \# (\mathcal{B} - \{b1\}). ((b1 \mid \cap \mid b2) \text{ choose } 2))$*  **by** *simp*  
**have** *gtsq:  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl) \wedge 2) \geq (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl))$*   
**by** (*simp add: diff-le sum-mset-mono*)  
**have**  $(\sum b2 \in \# (\mathcal{B} - \{b1\}). ((b1 \mid \cap \mid b2) \text{ choose } 2)) = ((\Lambda * (\Lambda - 1)) \text{ div } 2) * (v - 1)$   
**using** *sum-choose-two-inter-num assms* **by** *blast*  
**then have** *start:  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). ((b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1)) \text{ div } 2)$*   
 $= ((\Lambda * (\Lambda - 1)) \text{ div } 2) * (v - 1)$   
**using** *a b* **by** *linarith*  
**have** *sum-dvd:  $2 \text{ dvd } (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1))$*   
– *)*  
**using** *sum-mset-dvd*  
**by** (*metis (no-types, lifting) dvd-mult dvd-mult2 dvd-refl odd-two-times-div-two-nat*)  
**then have**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1)) \text{ div } 2$   
 $= ((\Lambda * (\Lambda - 1)) \text{ div } 2) * (v - 1)$   
**using** *start sum-mset-distrib-div-if-dvd inner-dvd*  
**by** (*metis (mono-tags, lifting) image-mset-cong2*)  
**then have**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (b1 \mid \cap \mid bl) * ((b1 \mid \cap \mid bl) - 1)) \text{ div } 2$

2

= (v - 1) \* ((Λ \* (Λ - 1)) div 2)

by *simp*

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl) \* ((b1 |∩| bl) - 1))

= (v - 1) \* (Λ \* (Λ - 1))

by (metis (no-types, lifting) div-mult-swap dvdI dvd-div-eq-iff dvd-mult dvd-mult2 odd-two-times-div-two-nat sum-dvd)

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup> - (b1 |∩| bl))

= (v - 1) \* (Λ \* (Λ - 1))

using *diff-mult-distrib2*

by (metis (no-types, lifting) multiset.map-cong0 nat-mult-1-right power2-eq-square)

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>)

- (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)) = (v - 1) \* (Λ \* (Λ - 1))

using *sum-mset-add-diff-nat*[of (remove1-mset b1 B) λ bl . (b1 |∩| bl) λ bl . (b1 |∩| bl)<sup>2</sup>]

*diff-le* by *presburger*

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>)

= (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)) + (v - 1) \* (Λ \* (Λ - 1))

using *gtsq*

by (metis *le-add-diff-inverse*)

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>) = (Λ \* (v - 1)) + ((v - 1) \* (Λ \* (Λ - 1)))

using *sym-sum-inter-num assms rep-value-sym symmetric-condition-2* by *auto*

then have *prev*: (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>) = (Λ \* (v - 1)) \* (Λ - 1) + (Λ \* (v - 1))

by *fastforce*

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>) = (Λ \* (v - 1)) \* (Λ)

by (metis *Nat.le-imp-diff-is-add add-mult-distrib2 index-not-zero nat-mult-1-right*)

then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)<sup>2</sup>) = Λ \* Λ \* (v - 1)

using *mult commute* by *simp*

thus *?thesis* by (*simp add: power2-eq-square*)

qed

**lemma** *sym-sum-inter-num-to-zero*:

assumes *b1* ∈# B

shows (∑ bl ∈# (remove1-mset b1 B). (int (b1 |∩| bl) - (int Λ))<sup>2</sup>) = 0

**proof** -

have *rm1-size*: *size* (remove1-mset b1 B) = v - 1 using *assms b-non-zero int-ops(6)*

by (*auto simp add: symmetric size-remove1-mset-If*)

have (∑ bl ∈# (remove1-mset b1 B). ((int (b1 |∩| bl))<sup>2</sup>)) = (∑ bl ∈# (remove1-mset b1 B). (((b1 |∩| bl))<sup>2</sup>)) by *simp*

then have *ssi*: (∑ bl ∈# (remove1-mset b1 B). ((int (b1 |∩| bl))<sup>2</sup>)) = Λ<sup>2</sup> \* (v - 1)

using *sym-sum-inter-num-sq assms* by *simp*

**have**  $\bigwedge bl . bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}) \implies (\text{int } (b1 \ | \cap | \ bl) - (\text{int } \Lambda))^{\wedge 2} =$   
 $((\text{int } (b1 \ | \cap | \ bl))^{\wedge 2}) - (2 * (\text{int } \Lambda) * (\text{int } (b1 \ | \cap | \ bl))) + ((\text{int } \Lambda)^{\wedge 2})$   
**by** (*simp add: power2-diff*)  
**then have**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (\text{int } (b1 \ | \cap | \ bl) - (\text{int } \Lambda))^{\wedge 2}) =$   
 $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (((\text{int } (b1 \ | \cap | \ bl))^{\wedge 2}) - (2 * (\text{int } \Lambda)$   
 $* (\text{int } (b1 \ | \cap | \ bl)))) + ((\text{int } \Lambda)^{\wedge 2}))$   
**using** *sum-over-fun-eq by auto*  
**also have**  $\dots = (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (((\text{int } (b1 \ | \cap | \ bl))^{\wedge 2})$   
 $- (2 * (\text{int } \Lambda) * (\text{int } (b1 \ | \cap | \ bl)))) + (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}).$   
 $((\text{int } \Lambda)^{\wedge 2}))$   
**by** (*simp add: sum-mset.distrib*)  
**also have**  $\dots = (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). ((\text{int } (b1 \ | \cap | \ bl))^{\wedge 2})$   
 $- (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (2 * (\text{int } \Lambda) * (\text{int } (b1 \ | \cap | \ bl)))) + (\sum bl$   
 $\in \# (\text{remove1-mset } b1 \ \mathcal{B}). ((\text{int } \Lambda)^{\wedge 2}))$   
**using** *sum-mset-add-diff-int[of ( $\lambda bl . ((\text{int } (b1 \ | \cap | \ bl))^{\wedge 2})$ ) ( $\lambda bl . (2 * (\text{int } \Lambda) * (\text{int } (b1 \ | \cap | \ bl)))$ ) ( $\text{remove1-mset } b1 \ \mathcal{B}$ )]*  
**by** *simp*  
**also have**  $\dots = \Lambda^{\wedge 2} * (v - 1) - 2 * (\text{int } \Lambda) * (\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}).$   
 $((b1 \ | \cap | \ bl)))$   
 $+ (v - 1) * ((\text{int } \Lambda)^{\wedge 2})$  **using** *ssi rm1-size assms by (simp add: sum-mset-distrib-left)*  
**also have**  $\dots = 2 * \Lambda^{\wedge 2} * (v - 1) - 2 * (\text{int } \Lambda) * (k * (r - 1))$   
**using** *sym-sum-inter-num assms by simp*  
**also have**  $\dots = 2 * \Lambda^{\wedge 2} * (v - 1) - 2 * (\text{int } \Lambda) * (\Lambda * (v - 1))$   
**using** *rep-value-sym symmetric-condition-2 by simp*  
**finally have**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (\text{int } (b1 \ | \cap | \ bl) - \text{int } \Lambda)^{\wedge 2}) = 0$   
**by** (*auto simp add: power2-eq-square*)  
**thus** *?thesis by simp*  
**qed**

**theorem** *sym-block-intersections-index [simp]:*

**assumes**  $b1 \in \# \mathcal{B}$   
**assumes**  $b2 \in \# (\mathcal{B} - \{b1\})$   
**shows**  $b1 \ | \cap | \ b2 = \Lambda$

**proof** –

**define**  $l$  **where** *l-def:  $l = \text{int } \Lambda$*   
**then have** *pos:  $\bigwedge bl . (\text{int } (b1 \ | \cap | \ bl) - l)^{\wedge 2} \geq 0$*  **by** *simp*  
**have**  $(\sum bl \in \# (\text{remove1-mset } b1 \ \mathcal{B}). (\text{int } (b1 \ | \cap | \ bl) - l)^{\wedge 2}) = 0$   
**using** *sym-sum-inter-num-to-zero assms l-def by simp*  
**then have**  $\bigwedge bl . bl \in \text{set-mset } (\text{remove1-mset } b1 \ \mathcal{B}) \implies (\text{int } (b1 \ | \cap | \ bl) - l)^{\wedge 2}$   
 $= 0$   
**using** *sum-mset-0-iff-ge-0 pos by (metis (no-types, lifting))*  
**then have**  $\bigwedge bl . bl \in \text{set-mset } (\text{remove1-mset } b1 \ \mathcal{B}) \implies \text{int } (b1 \ | \cap | \ bl) = l$   
**by** *auto*  
**thus** *?thesis using assms(2) l-def of-nat-eq-iff by blast*  
**qed**

## 5.4.2 Symmetric BIBD is Simple

**lemma** *sym-block-mult-one [simp]:*

```

assumes  $bl \in \# \mathcal{B}$ 
shows multiplicity  $bl = 1$ 
proof (rule ccontr)
  assume  $\neg$  (multiplicity  $bl = 1$ )
  then have not: multiplicity  $bl \neq 1$  by simp
  have multiplicity  $bl \neq 0$  using assms
    by simp
  then have  $m$ : multiplicity  $bl \geq 2$  using not by linarith
  then have  $bl_{left}$ :  $bl \in \# (\mathcal{B} - \{\#bl\# \})$ 
    using in-diff-count by fastforce
  have  $bl \cap bl = k$  using k-non-zero assms
    by (simp add: intersection-number-def)
  then have  $k_{eq}$ :  $k = \Lambda$  using sym-block-intersections-index  $bl_{left}$  assms by simp
  then have  $v = k$ 
    using keq index-1t-replication rep-value-sym block-size-1t-v diffs0-imp-equal k-non-zero
zero-diff by linarith
  then show False using incomplete
    by simp
qed

end

sublocale symmetric-bibd  $\subseteq$  simple-design
  by unfold-locales simp

```

### 5.4.3 Residual/Derived Sym BIBD Constructions

Using the intersect result, we can reason further on residual and derived designs. Proofs based off lecture notes [4]

**locale** *symmetric-bibd-block-transformations* = *symmetric-bibd* + *bibd-block-transformations*  
**begin**

```

lemma derived-block-size [simp]:
  assumes  $b \in \# \mathcal{B}^D$ 
  shows card  $b = \Lambda$ 
proof –
  obtain  $bl_2$  where set:  $bl_2 \in \#$  remove1-mset  $bl \mathcal{B}$  and inter:  $b = bl_2 \cap bl$ 
    using derived-blocks-def assms by (meson derived-obtain-orig-block)
  then have card  $b = bl_2 \cap bl$ 
    by (simp add: intersection-number-def)
  thus ?thesis using sym-block-intersections-index
    using set intersect-num-commute valid-block by fastforce
qed

```

```

lemma derived-points-index [simp]:
  assumes  $ps \subseteq bl$ 
  assumes card  $ps = 2$ 
  shows  $\mathcal{B}^D$  index  $ps = \Lambda - 1$ 
proof –

```

**have**  $b\text{-in}$ :  $\bigwedge b . b \in \# (\text{remove1-mset } bl \ \mathcal{B}) \implies ps \subseteq b \implies ps \subseteq b \cap bl$   
**using** *assms by blast*  
**then have** *orig*:  $ps \subseteq \mathcal{V}$   
**using** *valid-block assms wellformed by blast*  
**then have** *lam*:  $\text{size } \{\# b \in \# \mathcal{B} . ps \subseteq b \#\} = \Lambda$  **using** *balanced*  
**by** (*simp add: assms(2) points-index-def*)  
**then have**  $\text{size } \{\# b \in \# \text{remove1-mset } bl \ \mathcal{B} . ps \subseteq b \#\} = \text{size } \{\# b \in \# \mathcal{B} . ps \subseteq b \#\} - 1$   
**using** *assms valid-block by (simp add: size-Diff-submset)*  
**then have**  $\text{size } \{\# b \in \# \text{remove1-mset } bl \ \mathcal{B} . ps \subseteq b \#\} = \Lambda - 1$   
**using** *lam index-not-zero by linarith*  
**then have**  $\text{size } \{\# bl \cap b \mid b \in \# (\text{remove1-mset } bl \ \mathcal{B}) . ps \subseteq bl \cap b \#\} = \Lambda - 1$   
**using** *b-in by (metis (no-types, lifting) Int-subset-iff-filter-mset-cong size-image-mset)*  
**then have**  $\text{size } \{\# x \in \# \{\# bl \cap b . b \in \# (\text{remove1-mset } bl \ \mathcal{B}) \#\} . ps \subseteq x \#\} = \Lambda - 1$   
**by** (*metis image-mset-filter-swap*)  
**then have**  $\text{size } \{\# x \in \# \mathcal{B}^D . ps \subseteq x \#\} = \Lambda - 1$  **by** (*simp add: derived-blocks-def*)  
**thus** *?thesis* **by** (*simp add: points-index-def*)  
**qed**

**lemma** *sym-derive-design-bibd*:

**assumes**  $\Lambda > 1$   
**shows** *bid*  $bl \ \mathcal{B}^D \ \Lambda (\Lambda - 1)$

**proof** –

**interpret** *des*: *proper-design*  $bl \ \mathcal{B}^D$  **using** *derived-is-proper assms valid-block by auto*

**have**  $\Lambda < k$  **using** *index-lt-replication rep-value-sym by linarith*

**then show** *?thesis* **using** *derived-block-size assms derived-points-index derived-points-order by (unfold-locales) (simp-all)*

**qed**

**lemma** *residual-block-size [simp]*:

**assumes**  $b \in \# \mathcal{B}^R$   
**shows**  $\text{card } b = k - \Lambda$

**proof** –

**obtain** *bl2* **where**  $sub: b = bl2 - bl$  **and**  $mem: bl2 \in \# \text{remove1-mset } bl \ \mathcal{B}$

**using** *assms residual-blocks-def by auto*

**then have**  $\text{card } b = \text{card } bl2 - \text{card } (bl2 \cap bl)$

**using** *card-Diff-subset-Int valid-block finite-blocks*

**by** (*simp add: card-Diff-subset-Int*)

**then have**  $\text{card } b = \text{card } bl2 - bl2 \cap bl$

**using** *finite-blocks card-inter-lt-single by (simp add: intersection-number-def)*

**thus** *?thesis* **using** *sym-block-intersections-index uniform*

**by** (*metis valid-block in-diffD intersect-num-commute mem*)

**qed**

**lemma** *residual-index [simp]*:

**assumes**  $ps \subseteq bl^c$   
**assumes**  $card\ ps = 2$   
**shows**  $(\mathcal{B}^R)\ index\ ps = \Lambda$   
**proof** –  
**have**  $a: \bigwedge b . (b \in\# \text{remove1-mset } bl\ \mathcal{B} \implies ps \subseteq b \implies ps \subseteq (b - bl))$  **using**  
*assms*  
**by** (*meson DiffI block-complement-elem-iff block-complement-subset-points subsetD subsetI*)  
**have**  $b: \bigwedge b . (b \in\# \text{remove1-mset } bl\ \mathcal{B} \implies ps \subseteq (b - bl) \implies ps \subseteq b)$   
**by** *auto*  
**have** *not-ss*:  $\neg (ps \subseteq bl)$  **using** *set-diff-non-empty-not-subset blocks-nempty t-non-zero*  
*assms*  
*block-complement-def* **by** *fastforce*  
**have**  $\mathcal{B}^R\ index\ ps = size\ \{\# x \in\# \{\# b - bl . b \in\# (\text{remove1-mset } bl\ \mathcal{B})\ \#\} .$   
 $ps \subseteq x\ \#\}$   
**using** *assms valid-block* **by** (*simp add: points-index-def residual-blocks-def*)  
**also have**  $\dots = size\ \{\# b - bl \mid b \in\# (\text{remove1-mset } bl\ \mathcal{B}) . ps \subseteq b - bl\ \#\}$   
**by** (*metis image-mset-filter-swap*)  
**finally have**  $\mathcal{B}^R\ index\ ps = size\ \{\# b \in\# (\text{remove1-mset } bl\ \mathcal{B}) . ps \subseteq b\ \#\}$   
**using**  $a\ b$   
**by** (*metis (no-types, lifting) filter-mset-cong size-image-mset*)  
**thus** *?thesis*  
**using** *balanced not-ss assms points-index-alt-def block-complement-subset-points*  
**by** *auto*  
**qed**

**lemma** *sym-residual-design-bibd*:

**assumes**  $k \geq \Lambda + 2$   
**shows** *bid*  $(bl^c)\ \mathcal{B}^R\ (k - \Lambda)\ \Lambda$

**proof** –

**interpret** *des: proper-design*  $bl^c\ \mathcal{B}^R$

**using** *residual-is-proper assms(1) valid-block sym-block-mult-one* **by** *fastforce*

**show** *?thesis* **using** *residual-block-size assms sym-design-vk-gt-kl residual-order*  
*residual-index*

**by** (*unfold-locales*) *simp-all*

**qed**

**end**

## 5.5 BIBD's and Other Block Designs

BIBD's are closely related to other block designs by indirect inheritance

**sublocale** *bid*  $\subseteq k\text{-}\Lambda\text{-PBD } \mathcal{V}\ \mathcal{B}\ \Lambda\ k$

**using** *block-size-gt-t* **by** (*unfold-locales*) *simp-all*

**lemma** *incomplete-PBD-is-bibd*:

**assumes**  $k < card\ V$  **and**  $k\text{-}\Lambda\text{-PBD } V\ B\ \Lambda\ k$

**shows** *bid*  $V\ B\ k\ \Lambda$

**proof** –

```

interpret inc: incomplete-design  $V B k$  using assms
  by (auto simp add: block-design.incomplete-designI k- $\Lambda$ -PBD.axioms(2))
interpret pairwise-balance: pairwise-balance  $V B \Lambda$  using assms
  by (auto simp add: k- $\Lambda$ -PBD.axioms(1))
show ?thesis using assms k- $\Lambda$ -PBD.block-size-t by (unfold-locales) (simp-all)
qed

lemma (in bibd) bibd-to-pbdI[intro]:
  assumes  $\Lambda = 1$ 
  shows  $k\text{-PBD} \vee \mathcal{B} k$ 
proof –
  interpret pbd:  $k\text{-PBD} \vee \mathcal{B} \Lambda k$ 
  by (simp add: k- $\Lambda$ -PBD.axioms)
  show ?thesis using assms by (unfold-locales) (simp-all add: t-lt-order min-block-size-2)
qed

locale incomplete-PBD = incomplete-design + k- $\Lambda$ -PBD

sublocale incomplete-PBD  $\subseteq$  bibd
  using block-size-t by (unfold-locales) simp

end

```

## 6 Resolvable Designs

Resolvable designs have further structure, and can be "resolved" into a set of resolution classes. A resolution class is a subset of blocks which exactly partitions the point set. Definitions based off the handbook [3] and Stinson [6]. This theory includes a proof of an alternate statement of Bose's theorem

```

theory Resolvable-Designs imports BIBD
begin

```

### 6.1 Resolutions and Resolution Classes

A resolution class is a partition of the point set using a set of blocks from the design A resolution is a group of resolution classes partitioning the block collection

```

context incidence-system
begin

```

```

definition resolution-class :: 'a set  $set \Rightarrow bool$  where
resolution-class  $S \iff partition\text{-on } \mathcal{V} S \wedge (\forall bl \in S . bl \in\# \mathcal{B})$ 

```

```

lemma resolution-classI [intro]:  $partition\text{-on } \mathcal{V} S \implies (\bigwedge bl . bl \in S \implies bl \in\# \mathcal{B})$ 
   $\implies resolution\text{-class } S$ 
  by (simp add: resolution-class-def)

```

**lemma** *resolution-classD1*: *resolution-class*  $S \implies$  *partition-on*  $\mathcal{V} S$   
**by** (*simp add: resolution-class-def*)

**lemma** *resolution-classD2*: *resolution-class*  $S \implies$   $bl \in S \implies bl \in\# \mathcal{B}$   
**by** (*simp add: resolution-class-def*)

**lemma** *resolution-class-empty-iff*: *resolution-class*  $\{\}$   $\longleftrightarrow \mathcal{V} = \{\}$   
**by** (*auto simp add: resolution-class-def partition-on-def*)

**lemma** *resolution-class-complete*:  $\mathcal{V} \neq \{\} \implies \mathcal{V} \in\# \mathcal{B} \implies$  *resolution-class*  $\{\mathcal{V}\}$   
**by** (*auto simp add: resolution-class-def partition-on-space*)

**lemma** *resolution-class-union*: *resolution-class*  $S \implies \bigcup S = \mathcal{V}$   
**by** (*simp add: resolution-class-def partition-on-def*)

**lemma** (**in** *finite-incidence-system*) *resolution-class-finite*: *resolution-class*  $S \implies$   
*finite*  $S$   
**using** *finite-elements finite-sets* **by** (*auto simp add: resolution-class-def*)

**lemma** (**in** *design*) *resolution-class-sum-card*: *resolution-class*  $S \implies (\sum bl \in S .$   
*card*  $bl) = v$   
**using** *resolution-class-union finite-blocks*  
**by** (*auto simp add: resolution-class-def partition-on-def card-Union-disjoint*)

**definition** *resolution*:: 'a set multiset multiset  $\implies$  bool **where**  
*resolution*  $P \longleftrightarrow$  *partition-on-mset*  $\mathcal{B} P \wedge (\forall S \in\# P .$  *distinct-mset*  $S \wedge$  *resolu-*  
*tion-class* (*set-mset*  $S$ ))

**lemma** *resolutionI* : *partition-on-mset*  $\mathcal{B} P \implies (\bigwedge S . S \in\# P \implies$  *distinct-mset*  
 $S) \implies$   
 $(\bigwedge S . S \in\# P \implies$  *resolution-class* (*set-mset*  $S)) \implies$  *resolution*  $P$   
**by** (*simp add: resolution-def*)

**lemma** (**in** *proper-design*) *resolution-blocks*: *distinct-mset*  $\mathcal{B} \implies$  *disjoint* (*set-mset*  
 $\mathcal{B}) \implies$   
 $\bigcup (\text{set-mset } \mathcal{B}) = \mathcal{V} \implies$  *resolution*  $\{\#\mathcal{B}\#$   
**unfolding** *resolution-def resolution-class-def partition-on-mset-def partition-on-def*  
**using** *design-blocks-nempty blocks-nempty* **by** *auto*

**end**

## 6.2 Resolvable Design Locale

A resolvable design is one with a resolution  $P$

**locale** *resolvable-design* = *design* +  
**fixes** *partition* :: 'a set multiset multiset ( $\langle \mathcal{P} \rangle$ )  
**assumes** *resolvable*: *resolution*  $\mathcal{P}$   
**begin**

```

lemma resolutionD1: partition-on-mset  $\mathcal{B} \mathcal{P}$ 
  using resolvable by (simp add: resolution-def)

lemma resolutionD2:  $S \in \# \mathcal{P} \implies$  distinct-mset  $S$ 
  using resolvable by (simp add: resolution-def)

lemma resolutionD3:  $S \in \# \mathcal{P} \implies$  resolution-class (set-mset  $S$ )
  using resolvable by (simp add: resolution-def)

lemma resolution-class-blocks-disjoint:  $S \in \# \mathcal{P} \implies$  disjoint (set-mset  $S$ )
  using resolutionD3 by (simp add: partition-on-def resolution-class-def)

lemma resolution-not-empty:  $\mathcal{B} \neq \{\#\} \implies \mathcal{P} \neq \{\#\}$ 
  using partition-on-mset-not-empty resolutionD1 by auto

lemma resolution-blocks-subset:  $S \in \# \mathcal{P} \implies S \subseteq \# \mathcal{B}$ 
  using partition-on-mset-subsets resolutionD1 by auto

end

lemma (in incidence-system) resolvable-designI [intro]: resolution  $\mathcal{P} \implies$  design
 $\mathcal{V} \mathcal{B} \implies$ 
  resolvable-design  $\mathcal{V} \mathcal{B} \mathcal{P}$ 
  by (simp add: resolvable-design.intro resolvable-design-axioms.intro)

```

### 6.3 Resolvable Block Designs

An RBIBD is a resolvable BIBD - a common subclass of interest for block designs

```

locale r-block-design = resolvable-design + block-design
begin
lemma resolution-class-blocks-constant-size:  $S \in \# \mathcal{P} \implies$   $bl \in \# S \implies$  card  $bl =$ 
 $k$ 
  by (metis resolutionD3 resolution-classD2 uniform-alt-def-all)

lemma resolution-class-size1:
  assumes  $S \in \# \mathcal{P}$ 
  shows  $v = k * \text{size } S$ 
proof -
  have  $(\sum bl \in \# S . \text{card } bl) = (\sum bl \in (\text{set-mset } S) . \text{card } bl)$  using resolutionD2
  assms
  by (simp add: sum-unfold-sum-mset)
  then have eqv:  $(\sum bl \in \# S . \text{card } bl) = v$  using resolutionD3 assms resolu-
tion-class-sum-card
  by presburger
  have  $(\sum bl \in \# S . \text{card } bl) = (\sum bl \in \# S . k)$  using resolution-class-blocks-constant-size
  assms
  by auto

```

**thus** *?thesis* **using** *eqv* **by** *auto*  
**qed**

**lemma** *resolution-class-size2*:  
**assumes**  $S \in \# \mathcal{P}$   
**shows**  $\text{size } S = v \text{ div } k$   
**using** *resolution-class-size1* *assms*  
**by** (*metis nonzero-mult-div-cancel-left not-one-le-zero k-non-zero*)

**lemma** *resolvable-necessary-cond-v*:  $k \text{ dvd } v$   
**proof** –  
**obtain**  $S$  **where** *s-in*:  $S \in \# \mathcal{P}$  **using** *resolution-not-empty design-blocks-nempty*  
**by** *blast*  
**then have**  $k * \text{size } S = v$  **using** *resolution-class-size1* **by** *simp*  
**thus** *?thesis* **by** (*metis dvd-triv-left*)  
**qed**

**end**

**locale** *rbibd* = *r-block-design* + *bibd*

**begin**

**lemma** *resolvable-design-num-res-classes*:  $\text{size } \mathcal{P} = r$   
**proof** –  
**have** *k-ne0*:  $k \neq 0$  **using** *k-non-zero* **by** *auto*  
**have** *f1*:  $b = (\sum S \in \# \mathcal{P} . \text{size } S)$   
**by** (*metis partition-on-msetD1 resolutionD1 size-big-union-sum*)  
**then have**  $b = (\sum S \in \# \mathcal{P} . v \text{ div } k)$  **using** *resolution-class-size2* *f1* **by** *auto*  
**then have** *f2*:  $b = (\text{size } \mathcal{P}) * (v \text{ div } k)$  **by** *simp*  
**then have**  $\text{size } \mathcal{P} = b \text{ div } (v \text{ div } k)$   
**using** *b-non-zero* **by** *auto*  
**then have**  $\text{size } \mathcal{P} = (b * k) \text{ div } v$  **using** *f2* *resolvable-necessary-cond-v*  
**by** (*metis div-div-div-same div-dvd-div dvd-triv-right k-ne0 nonzero-mult-div-cancel-right*)  
**thus** *?thesis* **using** *necessary-condition-two*  
**by** (*metis nonzero-mult-div-cancel-left not-one-less-zero t-design-min-v*)  
**qed**

**lemma** *resolvable-necessary-cond-b*:  $r \text{ dvd } b$   
**proof** –  
**have** *f1*:  $b = (\sum S \in \# \mathcal{P} . \text{size } S)$   
**by** (*metis partition-on-msetD1 resolutionD1 size-big-union-sum*)  
**then have**  $b = (\sum S \in \# \mathcal{P} . v \text{ div } k)$  **using** *resolution-class-size2* *f1* **by** *auto*  
**thus** *?thesis* **using** *resolvable-design-num-res-classes* **by** *simp*  
**qed**

### 6.3.1 Bose's Inequality

Bose's inequality is an important theorem on RBIBD's. This is a proof of an alternate statement of the thm, which does not require a linear algebraic approach, taken directly from Stinson [6]

**theorem** *bose-inequality-alternate*:  $b \geq v + r - 1 \iff r \geq k + \Lambda$

**proof** –

**from** *necessary-condition-two v-non-zero* **have**  $r: \langle r = b * k \text{ div } v \rangle$   
**by** (*metis div-mult-self1-is-m*)  
**define**  $k \ b \ v \ l \ r$  **where** *intdefs*:  $k \equiv (int \ k) \ b \equiv int \ b \ v = int \ v \ l \equiv int \ \Lambda \ r \equiv int \ r$   
**have** *kdvd*:  $k \ dvd \ (v * (r - k))$   
**using** *intdefs*  
**by** (*simp add: resolvable-necessary-cond-v*)  
**have** *necess1-alt*:  $l * v - l = r * (k - 1)$  **using** *necessary-condition-one intdefs*  
**by** (*smt (verit) diff-diff-cancel int-ops(2) int-ops(6) k-non-zero nat-mult-1-right of-nat-0-less-iff of-nat-mult right-diff-distrib' v-non-zero*)  
**then have** *v-eq*:  $v = (r * (k - 1) + l) \text{ div } l$   
**using** *necessary-condition-one index-not-zero intdefs*  
**by** (*metis diff-add-cancel nonzero-mult-div-cancel-left not-one-le-zero of-nat-mult linordered-euclidean-semiring-class.of-nat-div*)  
**have** *ldvd*:  $\bigwedge x. l \ dvd \ (x * (r * (k - 1) + l))$   
**by** (*metis necess1-alt diff-add-cancel dvd-mult dvd-triv-left*)  
**have**  $(b \geq v + r - 1) \iff ((v * r) \text{ div } k \geq v + r - 1)$   
**using** *necessary-condition-two k-non-zero intdefs*  
**by** (*metis (no-types, lifting) nonzero-mult-div-cancel-right not-one-le-zero of-nat-eq-0-iff of-nat-mult*)  
**also have**  $\dots \iff (((v * r) - (v * k)) \text{ div } k \geq r - 1)$   
**using** *k-non-zero k-non-zero r intdefs*  
**by** (*simp add: of-nat-div algebra-simps*)  
*(smt (verit, ccfv-threshold) One-nat-def div-mult-self4 of-nat-1 of-nat-mono)*  
**also have** *f2*:  $\dots \iff ((v * (r - k)) \text{ div } k \geq (r - 1))$   
**using** *int-distrib(3)* **by** (*simp add: mult.commute*)  
**also have** *f2*:  $\dots \iff ((v * (r - k)) \geq k * (r - 1))$   
**using** *k-non-zero kdvd intdefs* **by** *auto*  
**also have**  $\dots \iff (((r * (k - 1) + l) \text{ div } l) * (r - k) \geq k * (r - 1))$   
**using** *v-eq* **by** *presburger*  
**also have**  $\dots \iff ((r - k) * ((r * (k - 1) + l) \text{ div } l) \geq k * (r - 1))$   
**by** (*simp add: mult.commute*)  
**also have**  $\dots \iff ((r - k) * (r * (k - 1) + l) \text{ div } l \geq k * (r - 1))$   
**using** *div-mult-swap necessary-condition-one intdefs*  
**by** (*metis diff-add-cancel dvd-triv-left necess1-alt*)  
**also have**  $\dots \iff (((r - k) * (r * (k - 1) + l)) \geq l * (k * (r - 1)))$   
**using** *ldvd[of (r - k)] dvd-mult-div-cancel index-not-zero mult-strict-left-mono intdefs*  
**by** (*smt (verit) b-non-zero bibd-block-number bot-nat-0.extremum-strict div-0 less-eq-nat.simps(1)*)

```

    mult-eq-0-iff mult-left-le-imp-le mult-left-mono of-nat-0 of-nat-le-0-iff of-nat-le-iff
of-nat-less-iff)
  also have 1: ...  $\longleftrightarrow$   $((r - k) * (r * (k - 1))) + ((r - k) * l) \geq l * (k * (r - 1))$ 
  by (simp add: distrib-left)
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq l * k * (r - 1) - ((r - k) * l)$ 
  using mult.assoc by linarith
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq (l * k * r) - (l * k) - ((r * l) - (k * l))$ 
  using distrib-right by (simp add: distrib-left right-diff-distrib' left-diff-distrib')
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq (l * k * r) - (l * r)$ 
  by (simp add: mult.commute)
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq (l * (k * r)) - (l * r)$ 
  by linarith
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq (l * (r * k)) - (l * r)$ 
  by (simp add: mult.commute)
  also have ...  $\longleftrightarrow$   $((r - k) * r * (k - 1)) \geq l * r * (k - 1)$ 
  by (simp add: mult.assoc int-distrib(4))
  finally have  $(b \geq v + r - 1) \longleftrightarrow (r \geq k + l)$ 
  using index-lt-replication mult-right-le-imp-le r-gzero mult-cancel-right k-non-zero
intdefs
  by (smt (verit) of-nat-0-less-iff of-nat-1 of-nat-le-iff of-nat-less-iff)
  then have  $b \geq v + r - 1 \longleftrightarrow r \geq k + l$ 
  using k-non-zero le-add-diff-inverse of-nat-1 of-nat-le-iff intdefs by linarith
  thus ?thesis by simp
qed
end
end

```

## 7 Group Divisible Designs

Definitions in this section taken from the handbook [3] and Stinson [6]

```

theory Group-Divisible-Designs imports Resolvable-Designs
begin

```

### 7.1 Group design

We define a group design to have an additional parameter  $G$  which is a partition on the point set  $V$ . This is not defined in the handbook, but is a precursor to GDD's without index constraints

```

locale group-design = proper-design +
  fixes groups :: 'a set set ( $\langle \mathcal{G} \rangle$ )
  assumes group-partitions: partition-on  $\mathcal{V} \mathcal{G}$ 
  assumes groups-size: card  $\mathcal{G} > 1$ 
begin

```

```

lemma groups-not-empty:  $\mathcal{G} \neq \{\}$ 
  using groups-size by auto

```

**lemma** *num-groups-lt-points*:  $\text{card } \mathcal{G} \leq v$   
**by** (*simp add: partition-on-le-set-elements finite-sets group-partitions*)

**lemma** *groups-disjoint*: *disjoint*  $\mathcal{G}$   
**using** *group-partitions partition-onD2* **by** *auto*

**lemma** *groups-disjoint-pairwise*:  $G1 \in \mathcal{G} \implies G2 \in \mathcal{G} \implies G1 \neq G2 \implies \text{disjnt } G1\ G2$   
**using** *group-partitions partition-onD2 pairwiseD* **by** *fastforce*

**lemma** *point-in-one-group*:  $x \in G1 \implies G1 \in \mathcal{G} \implies G2 \in \mathcal{G} \implies G1 \neq G2 \implies x \notin G2$   
**using** *groups-disjoint-pairwise* **by** (*simp add: disjnt-iff*)

**lemma** *point-has-unique-group*:  $x \in \mathcal{V} \implies \exists! G. x \in G \wedge G \in \mathcal{G}$   
**using** *partition-on-partition-on-unique group-partitions*  
**by** *fastforce*

**lemma** *rep-number-point-group-one*:  
**assumes**  $x \in \mathcal{V}$   
**shows**  $\text{card } \{g \in \mathcal{G} . x \in g\} = 1$   
**proof** –  
**obtain**  $g'$  **where**  $g' \in \mathcal{G}$  **and**  $x \in g'$   
**using** *assms point-has-unique-group* **by** *blast*  
**then have**  $\{g \in \mathcal{G} . x \in g\} = \{g'\}$   
**using** *group-partitions partition-onD4* **by** *force*  
**thus** *?thesis*  
**by** *simp*  
**qed**

**lemma** *point-in-group*:  $G \in \mathcal{G} \implies x \in G \implies x \in \mathcal{V}$   
**using** *group-partitions partition-onD1* **by** *auto*

**lemma** *point-subset-in-group*:  $G \in \mathcal{G} \implies ps \subseteq G \implies ps \subseteq \mathcal{V}$   
**using** *point-in-group* **by** *auto*

**lemma** *group-subset-point-subset*:  $G \in \mathcal{G} \implies G' \subseteq G \implies ps \subseteq G' \implies ps \subseteq \mathcal{V}$   
**using** *point-subset-in-group* **by** *auto*

**lemma** *groups-finite*: *finite*  $\mathcal{G}$   
**using** *finite-elements finite-sets group-partitions* **by** *auto*

**lemma** *group-elements-finite*:  $G \in \mathcal{G} \implies \text{finite } G$   
**using** *groups-finite finite-sets group-partitions*  
**by** (*meson finite-subset point-in-group subset-iff*)

**lemma** *v-equals-sum-group-sizes*:  $v = (\sum G \in \mathcal{G}. \text{card } G)$   
**using** *group-partitions groups-disjoint partition-onD1 card-Union-disjoint group-elements-finite*

by *fastforce*

**lemma** *gdd-min-v*:  $v \geq 2$

**proof** –

have *assm*:  $\text{card } \mathcal{G} \geq 2$  **using** *groups-size* **by** *simp*

then have  $\bigwedge G . G \in \mathcal{G} \implies G \neq \{\}$  **using** *partition-onD3* *group-partitions* **by** *auto*

then have  $\bigwedge G . G \in \mathcal{G} \implies \text{card } G \geq 1$

using *group-elements-finite* *card-0-eq* **by** *fastforce*

then have  $(\sum G \in \mathcal{G} . \text{card } G) \geq 2$  **using** *assm*

using *sum-mono* **by** *force*

thus *?thesis* **using** *v-equals-sum-group-sizes*

by *linarith*

**qed**

**lemma** *min-group-size*:  $G \in \mathcal{G} \implies \text{card } G \geq 1$

**using** *partition-onD3* *group-partitions*

**using** *group-elements-finite* *not-le-imp-less* **by** *fastforce*

**lemma** *group-size-lt-v*:

**assumes**  $G \in \mathcal{G}$

**shows**  $\text{card } G < v$

**proof** –

have  $(\sum G' \in \mathcal{G} . \text{card } G') = v$  **using** *gdd-min-v* *v-equals-sum-group-sizes*

by *linarith*

then have *split-sum*:  $\text{card } G + (\sum G' \in (\mathcal{G} - \{G\}) . \text{card } G') = v$  **using** *assms* *sum.remove*

by (*metis* *groups-finite* *v-equals-sum-group-sizes*)

have  $\text{card } (\mathcal{G} - \{G\}) \geq 1$  **using** *groups-size*

by (*simp* *add*: *assms* *groups-finite*)

then obtain  $G'$  **where**  $g_{in}$ :  $G' \in (\mathcal{G} - \{G\})$

by (*meson* *elem-exists-non-empty-set* *less-le-trans* *less-numeral-extra*(1))

then have  $\text{card } G' \geq 1$  **using** *min-group-size* **by** *auto*

then have  $(\sum G' \in (\mathcal{G} - \{G\}) . \text{card } G') \geq 1$

by (*metis* *gin* *finite-Diff* *groups-finite* *leI* *less-one* *sum-eq-0-iff*)

thus *?thesis* **using** *split-sum*

by *linarith*

**qed**

### 7.1.1 Group Type

GDD's have a "type", which is defined by a sequence of group sizes  $g_i$ , and the number of groups of that size  $a_i$ :  $g_1^{a_1} g_2^{a_2} \dots g_n^{a_n}$

**definition** *group-sizes* :: *nat set* **where**

*group-sizes*  $\equiv \{\text{card } G \mid G . G \in \mathcal{G}\}$

**definition** *groups-of-size* :: *nat*  $\Rightarrow$  *nat* **where**

*groups-of-size*  $g \equiv \text{card } \{ G \in \mathcal{G} . \text{card } G = g \}$

**definition** *group-type* :: (nat × nat) set **where**  
*group-type* ≡ { (g, groups-of-size g) | g . g ∈ group-sizes }

**lemma** *group-sizes-min*: x ∈ group-sizes ⇒ x ≥ 1  
**unfolding** *group-sizes-def* **using** *min-group-size* *group-size-lt-v* **by** *auto*

**lemma** *group-sizes-max*: x ∈ group-sizes ⇒ x < v  
**unfolding** *group-sizes-def* **using** *min-group-size* *group-size-lt-v* **by** *auto*

**lemma** *group-size-implies-group-existence*: x ∈ group-sizes ⇒ ∃ G. G ∈  $\mathcal{G}$  ∧ card G = x  
**unfolding** *group-sizes-def* **by** *auto*

**lemma** *groups-of-size-zero*: groups-of-size 0 = 0  
**proof** –  
**have** *empty*: { G ∈  $\mathcal{G}$  . card G = 0 } = {} **using** *min-group-size*  
**by** *fastforce*  
**thus** ?thesis **unfolding** *groups-of-size-def*  
**by** (*simp add: empty*)  
**qed**

**lemma** *groups-of-size-max*:  
**assumes** g ≥ v  
**shows** groups-of-size g = 0  
**proof** –  
**have** { G ∈  $\mathcal{G}$  . card G = g } = {} **using** *group-size-lt-v* *assms* **by** *fastforce*  
**thus** ?thesis **unfolding** *groups-of-size-def*  
**by** (*simp add:* ⟨{ G ∈  $\mathcal{G}$  . card G = g } = {}⟩)  
**qed**

**lemma** *group-type-contained-sizes*: (g, a) ∈ group-type ⇒ g ∈ group-sizes  
**unfolding** *group-type-def* **by** *simp*

**lemma** *group-type-contained-count*: (g, a) ∈ group-type ⇒ card { G ∈  $\mathcal{G}$  . card G = g } = a  
**unfolding** *group-type-def* *groups-of-size-def* **by** *simp*

**lemma** *group-card-in-sizes*: g ∈  $\mathcal{G}$  ⇒ card g ∈ group-sizes  
**unfolding** *group-sizes-def* **by** *auto*

**lemma** *group-card-non-zero-groups-of-size-min*:  
**assumes** g ∈  $\mathcal{G}$   
**assumes** card g = a  
**shows** groups-of-size a ≥ 1  
**proof** –  
**have** g ∈ { G ∈  $\mathcal{G}$  . card G = a } **using** *assms* **by** *simp*  
**then have** { G ∈  $\mathcal{G}$  . card G = a } ≠ {} **by** *auto*  
**then have** card { G ∈  $\mathcal{G}$  . card G = a } ≠ 0

by (*simp add: groups-finite*)  
thus ?thesis **unfolding** *groups-of-size-def* **by** *simp*  
**qed**

**lemma** *elem-in-group-sizes-min-of-size*:  
**assumes**  $a \in \text{group-sizes}$   
**shows**  $\text{groups-of-size } a \geq 1$   
**using** *assms group-card-non-zero-groups-of-size-min group-size-implies-group-existance*  
**by** *blast*

**lemma** *group-card-non-zero-groups-of-size-max*:  
**shows**  $\text{groups-of-size } a \leq v$   
**proof** –  
**have**  $\{G \in \mathcal{G} . \text{card } G = a\} \subseteq \mathcal{G}$  **by** *simp*  
**then have**  $\text{card } \{G \in \mathcal{G} . \text{card } G = a\} \leq \text{card } \mathcal{G}$   
**by** (*simp add: card-mono groups-finite*)  
**thus** ?thesis  
**using** *groups-of-size-def num-groups-lt-points* **by** *auto*  
**qed**

**lemma** *group-card-in-type*:  $g \in \mathcal{G} \implies \exists x . (\text{card } g, x) \in \text{group-type} \wedge x \geq 1$   
**unfolding** *group-type-def* **using** *group-card-non-zero-groups-of-size-min*  
**by** (*simp add: group-card-in-sizes*)

**lemma** *partition-groups-on-size*:  $\text{partition-on } \mathcal{G} \{ \{ G \in \mathcal{G} . \text{card } G = g \} \mid g . g \in \text{group-sizes} \}$   
**proof** (*intro partition-onI, auto*)  
**fix**  $g$   
**assume**  $a1: g \in \text{group-sizes}$   
**assume**  $\forall x. x \in \mathcal{G} \longrightarrow \text{card } x \neq g$   
**then show** *False* **using**  $a1$  *group-size-implies-group-existance* **by** *auto*  
**next**  
**fix**  $x$   
**assume**  $x \in \mathcal{G}$   
**then show**  $\exists xa. (\exists g. xa = \{G \in \mathcal{G} . \text{card } G = g\} \wedge g \in \text{group-sizes}) \wedge x \in xa$   
**using** *group-card-in-sizes* **by** *auto*  
**qed**

**lemma** *group-size-partition-covers-points*:  $\bigcup (\bigcup \{ \{ G \in \mathcal{G} . \text{card } G = g \} \mid g . g \in \text{group-sizes} \}) = \mathcal{V}$   
**by** (*metis (no-types, lifting) group-partitions partition-groups-on-size partition-onD1*)

**lemma** *groups-of-size-alt-def-count*:  $\text{groups-of-size } g = \text{count } \{ \# \text{card } G . G \in \# \text{mset-set } \mathcal{G} \# \} g$   
**proof** –  
**have**  $a: \text{groups-of-size } g = \text{card } \{ G \in \mathcal{G} . \text{card } G = g \}$  **unfolding** *groups-of-size-def*  
**by** *simp*  
**then have**  $\text{groups-of-size } g = \text{size } \{ \# G \in \# (\text{mset-set } \mathcal{G}) . \text{card } G = g \# \}$   
**using** *groups-finite* **by** *auto*

```

then have size-repr: groups-of-size g = size {# x ∈# {# card G . G ∈#
mset-set G #} . x = g #}
using groups-finite by (simp add: filter-mset-image-mset)
have group-sizes = set-mset ({# card G . G ∈# mset-set G #})
using group-sizes-def groups-finite by auto
thus ?thesis using size-repr by (simp add: count-size-set-repr)
qed

```

```

lemma v-sum-type-rep: v = (∑ g ∈ group-sizes . g * (groups-of-size g))
proof –
have gs: set-mset {# card G . G ∈# mset-set G #} = group-sizes
unfolding group-sizes-def using groups-finite by auto
have v = card (∪(∪{{ G ∈ G . card G = g } | g . g ∈ group-sizes}))
using group-size-partition-covers-points by simp
have v1: v = (∑ x ∈# {# card G . G ∈# mset-set G #}. x)
by (simp add: sum-unfold-sum-mset v-equals-sum-group-sizes)
then have v = (∑ x ∈ set-mset {# card G . G ∈# mset-set G #} . x * (count
{# card G . G ∈# mset-set G #} x))
using mset-set-size-card-count by (simp add: v1)
thus ?thesis using gs groups-of-size-alt-def-count by auto
qed

```

**end**

### 7.1.2 Uniform Group designs

A group design requiring all groups are the same size

```

locale uniform-group-design = group-design +
fixes u-group-size :: nat (⟨m⟩)
assumes uniform-groups: G ∈ G ⇒ card G = m

```

**begin**

```

lemma m-positive: m ≥ 1
proof –
obtain G where G ∈ G using groups-size elem-exists-non-empty-set gr-implies-not-zero
by blast
thus ?thesis using uniform-groups min-group-size by fastforce
qed

```

```

lemma uniform-groups-alt: ∀ G ∈ G . card G = m
using uniform-groups by blast

```

```

lemma uniform-groups-group-sizes: group-sizes = {m}
using design-points-nempty group-card-in-sizes group-size-implies-group-existance
point-has-unique-group uniform-groups-alt by force

```

```

lemma uniform-groups-group-size-singleton: is-singleton (group-sizes)

```

**using** *uniform-groups-group-sizes* **by** *auto*

**lemma** *set-filter-eq-P-forall*:  $\forall x \in X . P x \implies \text{Set.filter } P X = X$   
**by** (*simp add: Collect-conj-eq Int-absorb2 subsetI*)

**lemma** *uniform-groups-groups-of-size-m*: *groups-of-size*  $m = \text{card } \mathcal{G}$   
**proof**(*simp add: groups-of-size-def*)  
**have**  $\{G \in \mathcal{G} . \text{card } G = m\} = \mathcal{G}$  **using** *uniform-groups-alt set-filter-eq-P-forall*  
**by** *auto*  
**thus**  $\text{card } \{G \in \mathcal{G} . \text{card } G = m\} = \text{card } \mathcal{G}$  **by** *simp*  
**qed**

**lemma** *uniform-groups-of-size-not-m*:  $x \neq m \implies \text{groups-of-size } x = 0$   
**by** (*simp add: groups-of-size-def card-eq-0-iff uniform-groups*)

**end**

## 7.2 GDD

A GDD extends a group design with an additional index parameter. Each pair of elements must occur either  $\Lambda$  times if in diff groups, or 0 times if in the same group

**locale** *GDD* = *group-design* +  
**fixes** *index* :: *int* ( $\langle \Lambda \rangle$ )  
**assumes** *index-ge-1*:  $\Lambda \geq 1$   
**assumes** *index-together*:  $G \in \mathcal{G} \implies x \in G \implies y \in G \implies x \neq y \implies \mathcal{B} \text{ index } \{x, y\} = 0$   
**assumes** *index-distinct*:  $G1 \in \mathcal{G} \implies G2 \in \mathcal{G} \implies G1 \neq G2 \implies x \in G1 \implies y \in G2 \implies \mathcal{B} \text{ index } \{x, y\} = \Lambda$   
**begin**

**lemma** *points-sep-groups-ne*:  $G1 \in \mathcal{G} \implies G2 \in \mathcal{G} \implies G1 \neq G2 \implies x \in G1 \implies y \in G2 \implies x \neq y$   
**by** (*meson point-in-one-group*)

**lemma** *index-together-alt-ss*:  $ps \subseteq G \implies G \in \mathcal{G} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = 0$   
**using** *index-together* **by** (*metis card-2-iff insert-subset*)

**lemma** *index-distinct-alt-ss*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies (\bigwedge G . G \in \mathcal{G} \implies \neg ps \subseteq G) \implies \mathcal{B} \text{ index } ps = \Lambda$   
**using** *index-distinct* **by** (*metis card-2-iff empty-subsetI insert-subset point-has-unique-group*)

**lemma** *gdd-index-options*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = 0 \vee \mathcal{B} \text{ index } ps = \Lambda$   
**using** *index-distinct-alt-ss index-together-alt-ss* **by** *blast*

**lemma** *index-zero-implies-same-group*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = 0 \implies$   
 $\exists G \in \mathcal{G} . ps \subseteq G$  **using** *index-distinct-alt-ss gr-implies-not-zero*  
**by** (*metis index-ge-1 less-one of-nat-0 of-nat-1 of-nat-le-0-iff*)

**lemma** *index-zero-implies-same-group-unique*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = 0 \implies$   
 $\exists! G \in \mathcal{G} . ps \subseteq G$   
**by** (*meson GDD.index-zero-implies-same-group GDD-axioms card-2-iff' group-design.point-in-one-group*  
*group-design-axioms in-mono*)

**lemma** *index-not-zero-impl-diff-group*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = \Lambda \implies$   
 $(\bigwedge G . G \in \mathcal{G} \implies \neg ps \subseteq G)$   
**using** *index-ge-1 index-together-alt-ss* **by** *auto*

**lemma** *index-zero-implies-one-group*:  
**assumes**  $ps \subseteq \mathcal{V}$   
**and**  $\text{card } ps = 2$   
**and**  $\mathcal{B} \text{ index } ps = 0$   
**shows**  $\text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\# \} = 1$   
**proof** –  
**obtain**  $G$  **where**  $ging: G \in \mathcal{G}$  **and**  $psin: ps \subseteq G$   
**using** *index-zero-implies-same-group groups-size assms* **by** *blast*  
**then have** *unique*:  $\bigwedge G2 . G2 \in \mathcal{G} \implies G \neq G2 \implies \neg ps \subseteq G2$   
**using** *index-zero-implies-same-group-unique* **by** (*metis assms*)  
**have**  $\bigwedge G' . G' \in \mathcal{G} \longleftrightarrow G' \in \# \text{ mset-set } \mathcal{G}$   
**by** (*simp add: groups-finite*)  
**then have** *eq-mset*:  $\{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\# \} = \text{mset-set } \{b \in \mathcal{G} . ps \subseteq b\}$   
**using** *filter-mset-mset-set groups-finite* **by** *blast*  
**then have**  $\{b \in \mathcal{G} . ps \subseteq b\} = \{G\}$  **using** *ging unique psin* **by** *blast*  
**thus** *?thesis* **by** (*simp add: eq-mset*)  
**qed**

**lemma** *index-distinct-group-num-alt-def*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies$   
 $\text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\# \} = 0 \implies \mathcal{B} \text{ index } ps = \Lambda$   
**by** (*metis gdd-index-options index-zero-implies-one-group numeral-One zero-neq-numeral*)

**lemma** *index-non-zero-implies-no-group*:  
**assumes**  $ps \subseteq \mathcal{V}$   
**and**  $\text{card } ps = 2$   
**and**  $\mathcal{B} \text{ index } ps = \Lambda$   
**shows**  $\text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\# \} = 0$   
**proof** –  
**have**  $\bigwedge G . G \in \mathcal{G} \implies \neg ps \subseteq G$  **using** *index-not-zero-impl-diff-group assms*  
**by** *simp*

**then have**  $\{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\#\} = \{\#\}$   
**using** *filter-mset-empty-if-finite-and-filter-set-empty* **by** *force*  
**thus** *?thesis* **by** *simp*  
**qed**

**lemma** *gdd-index-non-zero-iff*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies$   
 $\mathcal{B} \text{ index } ps = \Lambda \iff \text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\#\} = 0$   
**using** *index-non-zero-implies-no-group index-distinct-group-num-alt-def* **by** *auto*

**lemma** *gdd-index-zero-iff*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies$   
 $\mathcal{B} \text{ index } ps = 0 \iff \text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\#\} = 1$   
**apply** (*auto simp add: index-zero-implies-one-group*)  
**by** (*metis GDD.gdd-index-options GDD-axioms index-non-zero-implies-no-group old.nat.distinct(2)*)

**lemma** *points-index-upper-bound*:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps \leq \Lambda$   
**using** *gdd-index-options index-ge-1*  
**by** (*metis int-one-le-iff-zero-less le\_refl of-nat-0 of-nat-0-le-iff of-nat-le-iff zero-less-imp-eq-int*)

**lemma** *index-1-imp-mult-1*:  
**assumes**  $\Lambda = 1$   
**assumes**  $bl \in \# \mathcal{B}$   
**assumes**  $\text{card } bl \geq 2$   
**shows** *multiplicity*  $bl = 1$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\text{multiplicity } bl = 1)$   
**then have** *multiplicity*  $bl \neq 1$  **and** *multiplicity*  $bl \neq 0$  **using** *assms* **by** *simp-all*

**then have**  $m$ : *multiplicity*  $bl \geq 2$  **by** *linarith*  
**obtain**  $ps$  **where**  $ps \subseteq bl \wedge \text{card } ps = 2$   
**using** *nat-int-comparison(3) obtain-subset-with-card-n* **by** (*metis assms(3)*)  
**then have**  $\mathcal{B} \text{ index } ps \geq 2$   
**using**  $m$  *points-index-count-min ps* **by** *blast*  
**then show** *False* **using** *assms index-distinct ps antisym-conv2 not-numeral-less-zero*

$\text{numeral-le-one-iff points-index-ps-nin semiring-norm(69) zero-neq-numeral}$   
**by** (*metis gdd-index-options int-int-eq int-ops(2)*)  
**qed**

**lemma** *simple-if-block-size-gt-2*:  
**assumes**  $\bigwedge bl . \text{card } bl \geq 2$   
**assumes**  $\Lambda = 1$   
**shows** *simple-design*  $\mathcal{V} \mathcal{B}$   
**using** *index-1-imp-mult-1 assms* **apply** (*unfold-locales*)  
**by** (*metis card.empty not-numeral-le-zero*)

**end**

### 7.2.1 Sub types of GDD's

In literature, a GDD is usually defined in a number of different ways, including factors such as block size limitations

**locale**  $K\text{-}\Lambda\text{-GDD} = K\text{-block-design} + \text{GDD}$

**locale**  $k\text{-}\Lambda\text{-GDD} = \text{block-design} + \text{GDD}$

**sublocale**  $k\text{-}\Lambda\text{-GDD} \subseteq K\text{-}\Lambda\text{-GDD} \vee \mathcal{B} \{k\} \mathcal{G} \Lambda$   
**by** (*unfold-locales*)

**locale**  $K\text{-GDD} = K\text{-}\Lambda\text{-GDD} \vee \mathcal{B} \mathcal{K} \mathcal{G} 1$   
**for** *point-set* ( $\langle \mathcal{V} \rangle$ ) **and** *block-collection* ( $\langle \mathcal{B} \rangle$ ) **and** *sizes* ( $\langle \mathcal{K} \rangle$ ) **and** *groups* ( $\langle \mathcal{G} \rangle$ )

**locale**  $k\text{-GDD} = k\text{-}\Lambda\text{-GDD} \vee \mathcal{B} k \mathcal{G} 1$   
**for** *point-set* ( $\langle \mathcal{V} \rangle$ ) **and** *block-collection* ( $\langle \mathcal{B} \rangle$ ) **and** *u-block-size* ( $\langle k \rangle$ ) **and** *groups* ( $\langle \mathcal{G} \rangle$ )

**sublocale**  $k\text{-GDD} \subseteq K\text{-GDD} \vee \mathcal{B} \{k\} \mathcal{G}$   
**by** (*unfold-locales*)

**lemma** (**in**  $K\text{-GDD}$ ) *multiplicity-1*:  $bl \in \# \mathcal{B} \implies \text{card } bl \geq 2 \implies \text{multiplicity } bl = 1$

**using** *index-1-imp-mult-1* **by** *simp*

**locale**  $R\text{GDD} = \text{GDD} + \text{resolvable-design}$

## 7.3 GDD and PBD Constructions

GDD's are commonly studied alongside PBD's (pairwise balanced designs). Many constructions have been developed for designs to create a GDD from a PBD and vice versa. In particular, Wilsons Construction is a well known construction, which is formalised in this section. It should be noted that many of the more basic constructions in this section are often stated without proof/all the necessary assumptions in textbooks/course notes.

**context**  $\text{GDD}$

**begin**

### 7.3.1 GDD Delete Point construction

**lemma** *delete-point-index-zero*:

**assumes**  $G \in \{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\}$

**and**  $y \in G$  **and**  $z \in G$  **and**  $z \neq y$

**shows** (*del-point-blocks*  $x$ ) *index*  $\{y, z\} = 0$

**proof** –

**have**  $y \neq x$  **using** *assms(1)* *assms(2)* **by** *blast*

**have**  $z \neq x$  **using** *assms(1)* *assms(3)* **by** *blast*

**obtain**  $G'$  **where** *ing*:  $G' \in \mathcal{G}$  **and** *ss*:  $G \subseteq G'$

**using** *assms(1)* **by** *auto*  
**have**  $\{y, z\} \subseteq G$  **by** (*simp add: assms(2) assms(3)*)  
**then have**  $\{y, z\} \subseteq \mathcal{V}$   
**by** (*meson ss ing group-subset-point-subset*)  
**then have**  $\{y, z\} \subseteq (\text{del-point } x)$   
**using**  $\langle y \neq x \rangle \langle z \neq x \rangle$  *del-point-def* **by** *fastforce*  
**thus** *?thesis* **using** *delete-point-index-eq index-together*  
**by** (*metis assms(2) assms(3) assms(4) in-mono ing ss*)  
**qed**

**lemma** *delete-point-index:*

**assumes**  $G1 \in \{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\}$   
**assumes**  $G2 \in \{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\}$   
**assumes**  $G1 \neq G2$  **and**  $y \in G1$  **and**  $z \in G2$   
**shows** *del-point-blocks x index*  $\{y, z\} = \Lambda$   
**proof** –  
**have**  $y \neq x$  **using** *assms* **by** *blast*  
**have**  $z \neq x$  **using** *assms* **by** *blast*  
**obtain**  $G1'$  **where** *ing1*:  $G1' \in \mathcal{G}$  **and** *t1*:  $G1 = G1' - \{x\}$   
**using** *assms(1)* **by** *auto*  
**obtain**  $G2'$  **where** *ing2*:  $G2' \in \mathcal{G}$  **and** *t2*:  $G2 = G2' - \{x\}$   
**using** *assms(2)* **by** *auto*  
**then have** *ss1*:  $G1 \subseteq G1'$  **and** *ss2*:  $G2 \subseteq G2'$  **using** *t1* **by** *auto*  
**then have**  $\{y, z\} \subseteq \mathcal{V}$  **using** *ing1 ing2 ss1 ss2 assms(4) assms(5)*  
**by** (*metis empty-subsetI insert-absorb insert-subset point-in-group*)  
**then have**  $\{y, z\} \subseteq \text{del-point } x$   
**using**  $\langle y \neq x \rangle \langle z \neq x \rangle$  *del-point-def* **by** *auto*  
**then have** *indx*: *del-point-blocks x index*  $\{y, z\} = \mathcal{B}$  *index*  $\{y, z\}$   
**using** *delete-point-index-eq* **by** *auto*  
**have**  $G1' \neq G2'$  **using** *assms t1 t2* **by** *fastforce*  
**thus** *?thesis* **using** *index-distinct*  
**using** *indx assms(4) assms(5) ing1 ing2 t1 t2* **by** *auto*  
**qed**

**lemma** *delete-point-group-size:*

**assumes**  $\{x\} \in \mathcal{G} \implies \text{card } \mathcal{G} > 2$   
**shows**  $1 < \text{card } \{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\}$   
**proof** (*cases*  $\{x\} \in \mathcal{G}$ )  
**case** *True*  
**then have**  $\bigwedge g. g \in (\mathcal{G} - \{\{x\}\}) \implies x \notin g$   
**by** (*meson disjnt-insert1 groups-disjoint pairwise-alt*)  
**then have** *simp*:  $\bigwedge g. g \in (\mathcal{G} - \{\{x\}\}) \implies g - \{x\} = g$   
**by** *simp*  
**have**  $\{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} = \{g - \{x\} \mid g. (g \in \mathcal{G} - \{\{x\}\})\}$  **using**  
*True*  
**by** *force*  
**then have**  $\{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} = \{g \mid g. (g \in \mathcal{G} - \{\{x\}\})\}$  **using**  
*simp*  
**by** (*smt (verit) Collect-cong*)

```

then have eq:  $\{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} = \mathcal{G} - \{\{x\}\}$  using set-self-img-compr
by blast
have  $\text{card } (\mathcal{G} - \{\{x\}\}) = \text{card } \mathcal{G} - 1$  using True
by (simp add: groups-finite)
then show ?thesis using True assms eq diff-is-0-eq' by force
next
case False
then have  $\bigwedge g' y. \{x\} \notin \mathcal{G} \implies g' \in \mathcal{G} \implies y \in \mathcal{G} \implies g' - \{x\} = y - \{x\} \implies$ 
 $g' = y$ 
by (metis all-not-in-conv insert-Diff-single insert-absorb insert-iff points-sep-groups-ne)
then have inj: inj-on  $(\lambda g. g - \{x\}) \mathcal{G}$  by (simp add: inj-onI False)
have  $\{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} = \{g - \{x\} \mid g. g \in \mathcal{G}\}$  using False by auto
then have  $\text{card } \{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} = \text{card } \mathcal{G}$  using inj groups-finite
card-image
by (auto simp add: card-image setcompr-eq-image)
then show ?thesis using groups-size by presburger
qed

```

```

lemma GDD-by-deleting-point:
assumes  $\bigwedge bl. bl \in \# \mathcal{B} \implies x \in bl \implies 2 \leq \text{card } bl$ 
assumes  $\{x\} \in \mathcal{G} \implies \text{card } \mathcal{G} > 2$ 
shows GDD (del-point  $x$ ) (del-point-blocks  $x$ )  $\{g - \{x\} \mid g. g \in \mathcal{G} \wedge g \neq \{x\}\} \wedge$ 
proof -
interpret pd: proper-design del-point  $x$  del-point-blocks  $x$ 
using delete-point-proper assms by blast
show ?thesis using delete-point-index-zero delete-point-index assms delete-point-group-size
by (unfold-locales) (simp-all add: partition-on-remove-pt group-partitions index-ge-1 del-point-def)
qed

```

**end**

**context** *K-GDD* **begin**

### 7.3.2 PBD construction from GDD

Two well known PBD constructions involve taking a GDD and either combining the groups and blocks to form a new block collection, or by adjoining a point

First prove that combining the groups and block set results in a constant index

```

lemma kgdd1-points-index-group-block:
assumes  $ps \subseteq \mathcal{V}$ 
and  $\text{card } ps = 2$ 
shows  $(\mathcal{B} + \text{mset-set } \mathcal{G}) \text{ index } ps = 1$ 
proof -
have index1:  $(\bigwedge G. G \in \mathcal{G} \implies \neg ps \subseteq G) \implies \mathcal{B} \text{ index } ps = 1$ 
using index-distinct-alt-ss assms by fastforce

```

**have** *groups1*:  $\mathcal{B}$  *index*  $ps = 0 \implies \text{size } \{\#b \in \# \text{ mset-set } \mathcal{G} . ps \subseteq b\# \} = 1$   
**using** *index-zero-implies-one-group* *assms* **by** *simp*  
**then have**  $(\mathcal{B} + \text{mset-set } \mathcal{G})$  *index*  $ps = \text{size } (\text{filter-mset } ((\subseteq) ps) (\mathcal{B} + \text{mset-set } \mathcal{G}))$   
**by** (*simp add: points-index-def*)  
**thus** *?thesis* **using** *index1* *groups1* *gdd-index-non-zero-iff* *gdd-index-zero-iff* *assms*  
  
*gdd-index-options* *points-index-def* *filter-union-mset* *union-commute*  
**by** (*metis add-cancel-right-left index-together-alt-ss point-index-distrib*)  
**qed**

Combining blocks and the group set forms a PBD

**lemma** *combine-block-groups-pairwise*: *pairwise-balance*  $\mathcal{V} (\mathcal{B} + \text{mset-set } \mathcal{G})$  1

**proof** –

**let**  $?B = \mathcal{B} + \text{mset-set } \mathcal{G}$   
**have**  $ss: \bigwedge G. G \in \mathcal{G} \implies G \subseteq \mathcal{V}$   
**by** (*simp add: point-in-group subsetI*)  
**have**  $\bigwedge G. G \in \mathcal{G} \implies G \neq \{\}$  **using** *group-partitions*  
**using** *partition-onD3* **by** *auto*  
**then interpret** *inc*: *design*  $\mathcal{V}$   $?B$   
**proof** (*unfold-locales*)  
**show**  $\bigwedge b. (\bigwedge G. G \in \mathcal{G} \implies G \neq \{\}) \implies b \in \# \mathcal{B} + \text{mset-set } \mathcal{G} \implies b \subseteq \mathcal{V}$   
**by** (*metis finite-set-mset-mset-set groups-finite ss union-iff wellformed*)  
**show**  $(\bigwedge G. G \in \mathcal{G} \implies G \neq \{\}) \implies \text{finite } \mathcal{V}$  **by** (*simp add: finite-sets*)  
**show**  $\bigwedge bl. (\bigwedge G. G \in \mathcal{G} \implies G \neq \{\}) \implies bl \in \# \mathcal{B} + \text{mset-set } \mathcal{G} \implies bl \neq \{\}$   
**using** *blocks-nempty* *groups-finite* **by** *auto*  
**qed**  
**show** *?thesis* **proof** (*unfold-locales*)  
**show**  $\text{inc.b} \neq 0$  **using** *b-positive* **by** *auto*  
**show**  $(1 :: \text{nat}) \leq 2$  **by** *simp*  
**show**  $2 \leq \text{inc.v}$  **by** (*simp add: gdd-min-v*)  
**then show**  $\bigwedge ps. ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies (\mathcal{B} + \text{mset-set } \mathcal{G})$  *index*  $ps = 1$   
**using** *kgdd1-points-index-group-block* **by** *simp*

**qed**

**qed**

**lemma** *combine-block-groups-PBD*:

**assumes**  $\bigwedge G. G \in \mathcal{G} \implies \text{card } G \in \mathcal{K}$

**assumes**  $\bigwedge k. k \in \mathcal{K} \implies k \geq 2$

**shows** *PBD*  $\mathcal{V} (\mathcal{B} + \text{mset-set } \mathcal{G})$   $\mathcal{K}$

**proof** –

**let**  $?B = \mathcal{B} + \text{mset-set } \mathcal{G}$

**interpret** *inc*: *pairwise-balance*  $\mathcal{V}$   $?B$  1 **using** *combine-block-groups-pairwise* **by** *simp*

**show** *?thesis* **using** *assms* *block-sizes* *groups-finite* *positive-ints*

**by** (*unfold-locales*) *auto*

**qed**

Prove adjoining a point to each group set results in a constant points index

**lemma** *kgdd1-index-adjoin-group-block*:

**assumes**  $x \notin \mathcal{V}$

**assumes**  $ps \subseteq \text{insert } x \ \mathcal{V}$

**assumes**  $\text{card } ps = 2$

**shows**  $(\mathcal{B} + \text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ index } ps = 1$

**proof** –

**have** *inj-on*  $((\text{insert}) \ x) \ \mathcal{G}$

**by** *(meson assms(1) inj-onI insert-ident point-in-group)*

**then have** *eq*:  $\text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\} = \{\# \text{ insert } x \ g . g \in \# \text{ mset-set } \mathcal{G}\# \}$

**by** *(simp add: image-mset-mset-set setcompr-eq-image)*

**thus** *?thesis*

**proof** *(cases x ∈ ps)*

**case** *True*

**then obtain** *y* **where** *y-ps*:  $ps = \{x, y\}$  **using** *assms(3)*

**by** *(metis card-2-iff doubleton-eq-iff insertE singletonD)*

**then have** *ynex*:  $y \neq x$  **using** *assms* **by** *fastforce*

**have** *yinv*:  $y \in \mathcal{V}$

**using** *assms(2) y-ps ynex* **by** *auto*

**have** *all-g*:  $\bigwedge g. g \in \# (\text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \implies x \in g$

**using** *eq* **by** *force*

**have** *iff*:  $\bigwedge g. g \in \mathcal{G} \implies y \in (\text{insert } x \ g) \longleftrightarrow y \in g$  **using** *ynex* **by** *simp*

**have** *b*:  $\mathcal{B} \text{ index } ps = 0$

**using** *True assms(1) points-index-ps-nin* **by** *fastforce*

**then have**  $(\mathcal{B} + \text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ index } ps =$   
 $(\text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ index } ps$

**using** *eq* **by** *(simp add: point-index-distrib)*

**also have**  $\dots = (\text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ rep } y$  **using** *points-index-pair-rep-num*

**by** *(metis (no-types, lifting) all-g y-ps)*

**also have** *0*:  $\dots = \text{card } \{b \in \{\text{insert } x \ g \mid g. g \in \mathcal{G}\} . y \in b\}$

**by** *(simp add: groups-finite rep-number-on-set-def)*

**also have** *1*:  $\dots = \text{card } \{\text{insert } x \ g \mid g. g \in \mathcal{G} \wedge y \in \text{insert } x \ g\}$

**by** *(smt (verit) Collect-cong mem-Collect-eq)*

**also have** *2*:  $\dots = \text{card } \{\text{insert } x \ g \mid g. g \in \mathcal{G} \wedge y \in g\}$

**using** *iff* **by** *metis*

**also have**  $\dots = \text{card } \{g \in \mathcal{G} . y \in g\}$  **using** *1 2 0 empty-iff eq groups-finite ynex insert-iff*

**by** *(metis points-index-block-image-add-eq points-index-single-rep-num rep-number-on-set-def)*

**finally have**  $(\mathcal{B} + \text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ index } ps = 1$

**using** *rep-number-point-group-one yinv* **by** *simp*

**then show** *?thesis*

**by** *simp*

**next**

**case** *False*

**then have** *v*:  $ps \subseteq \mathcal{V}$  **using** *assms(2)* **by** *auto*

**then have**  $(\mathcal{B} + \text{mset-set } \{\text{insert } x \ g \mid g. g \in \mathcal{G}\}) \text{ index } ps = (\mathcal{B} + \text{mset-set } \mathcal{G})$   
*index ps*

**using** *eq* **by** *(simp add: points-index-block-image-add-eq False point-index-distrib)*

**then show** *?thesis* **using** *v* *assms* *kgdd1-points-index-group-block* **by** *simp*  
**qed**  
**qed**

**lemma** *pairwise-by-adjoining-point*:  
**assumes**  $x \notin \mathcal{V}$   
**shows** *pairwise-balance* (*add-point*  $x$ ) ( $\mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}$ ) 1  
**proof** –  
**let**  $?B = \mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}$   
**let**  $?V = \text{add-point } x$   
**have** *vdef*:  $?V = \mathcal{V} \cup \{x\}$  **using** *add-point-def* **by** *simp*  
**show** *?thesis* **unfolding** *add-point-def* **using** *finite-sets* *design-blocks-nempty*  
**proof** (*unfold-locales*, *simp-all*)  
**have**  $\bigwedge G. G \in \mathcal{G} \implies \text{insert } x \ G \subseteq ?V$   
**by** (*simp* *add: point-in-group* *subsetI* *vdef*)  
**then have**  $\bigwedge G. G \in \# (\text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}) \implies G \subseteq ?V$   
**by** (*smt* (*verit*, *del-insts*) *elem-mset-set* *empty-iff* *infinite-set-mset-mset-set* *mem-Collect-eq*)  
**then show**  $\bigwedge b. b \in \# \mathcal{B} \vee b \in \# \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \} \implies b \subseteq \text{insert } x \ \mathcal{V}$   
**using** *wellformed* *add-point-def* **by** *fastforce*  
**next**  
**have**  $\bigwedge G. G \in \mathcal{G} \implies \text{insert } x \ G \neq \{\}$  **using** *group-partitions*  
**using** *partition-onD3* **by** *auto*  
**then have** *gnempty*:  $\bigwedge G. G \in \# (\text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}) \implies G \neq \{\}$   
**by** (*smt* (*verit*, *del-insts*) *elem-mset-set* *empty-iff* *infinite-set-mset-mset-set* *mem-Collect-eq*)  
**then show**  $\bigwedge bl. bl \in \# \mathcal{B} \vee bl \in \# \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \} \implies bl \neq \{\}$   
**using** *blocks-nempty* **by** *auto*  
**next**  
**have**  $\text{card } \mathcal{V} \geq 2$  **using** *gdd-min-v* **by** *simp*  
**then have**  $\text{card } (\text{insert } x \ \mathcal{V}) \geq 2$   
**by** (*meson* *card-insert-le* *dual-order.trans* *finite-sets*)  
**then show**  $2 \leq \text{card } (\text{insert } x \ \mathcal{V})$  **by** *auto*  
**next**  
**show**  $\bigwedge ps. ps \subseteq \text{insert } x \ \mathcal{V} \implies$   
 $\text{card } ps = 2 \implies (\mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}) \text{ index } ps = \text{Suc } 0$   
**using** *kgdd1-index-adjoin-group-block* **by** (*simp* *add: assms*)  
**qed**  
**qed**

**lemma** *PBD-by-adjoining-point*:  
**assumes**  $x \notin \mathcal{V}$   
**assumes**  $\bigwedge k. k \in \mathcal{K} \implies k \geq 2$   
**shows** *PBD* (*add-point*  $x$ ) ( $\mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}$ ) ( $\mathcal{K} \cup \{(\text{card } g) + 1 \mid g. g \in \mathcal{G}\}$ )  
**proof** –

```

let ?B = B + mset-set { insert x g | g. g ∈ G }
let ?V = (add-point x)
interpret inc: pairwise-balance ?V ?B 1 using pairwise-by-adjoining-point assms
by auto
show ?thesis using block-sizes positive-ints proof (unfold-locales)
  have xg:  $\bigwedge g. g \in \mathcal{G} \implies x \notin g$ 
    using assms point-in-group by auto
  have  $\bigwedge bl. bl \in \# \mathcal{B} \implies \text{card } bl \in \mathcal{K}$  by (simp add: block-sizes)
  have  $\bigwedge bl. bl \in \# \text{mset-set } \{ \text{insert } x g \mid g. g \in \mathcal{G} \} \implies bl \in \{ \text{insert } x g \mid g. g \in \mathcal{G} \}$ 
    by (simp add: groups-finite)
  then have  $\bigwedge bl. bl \in \# \text{mset-set } \{ \text{insert } x g \mid g. g \in \mathcal{G} \} \implies \text{card } bl \in \{ \text{card } g + 1 \mid g. g \in \mathcal{G} \}$ 
    proof -
      fix bl
      assume bl ∈ # mset-set { insert x g | g. g ∈ G }
      then have bl ∈ { insert x g | g. g ∈ G } by (simp add: groups-finite)
      then obtain g where gin: g ∈ G and i: bl = insert x g by auto
      thus card bl ∈ { (card g + 1) | g. g ∈ G }
        using gin group-elements-finite i xg by auto
    qed
  then show  $\bigwedge bl. bl \in \# \mathcal{B} + \text{mset-set } \{ \text{insert } x g \mid g. g \in \mathcal{G} \} \implies (\text{card } bl) \in \mathcal{K} \cup \{ (\text{card } g + 1) \mid g. g \in \mathcal{G} \}$ 
    using UnI1 UnI2 block-sizes union-iff by auto
  show  $\bigwedge x. x \in \mathcal{K} \cup \{ \text{card } g + 1 \mid g. g \in \mathcal{G} \} \implies 0 < x$ 
    using min-group-size positive-ints by auto
  show  $\bigwedge k. k \in \mathcal{K} \cup \{ \text{card } g + 1 \mid g. g \in \mathcal{G} \} \implies 2 \leq k$ 
    using min-group-size positive-ints assms by fastforce
  qed
qed

```

### 7.3.3 Wilson's Construction

Wilson's construction involves the combination of multiple k-GDD's. This proof was based of Stinson [6]

**lemma** *wilsons-construction-proper*:

```

assumes card I = w
assumes w > 0
assumes  $\bigwedge n. n \in \mathcal{K}' \implies n \geq 2$ 
assumes  $\bigwedge B. B \in \# \mathcal{B} \implies K\text{-GDD } (B \times I) (f B) \mathcal{K}' \{ \{x\} \times I \mid x. x \in B \}$ 
shows proper-design (V × I) (∑ B ∈ # B. (f B)) (is proper-design ?Y ?B)
proof (unfold-locales, simp-all)
  show  $\bigwedge b. \exists x \in \# \mathcal{B}. b \in \# f x \implies b \subseteq \mathcal{V} \times I$ 
    proof -
      fix b
      assume  $\exists x \in \# \mathcal{B}. b \in \# f x$ 
      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  $b \subseteq \mathcal{V} \times I$  using kgdd.wellformed
      using  $\langle B \in \# \mathcal{B} \rangle \langle b \in \# f B \rangle$  wellformed by fastforce
  qed
  show finite  $(\mathcal{V} \times I)$  using finite-sets assms bot-nat-0.not-eq-extremum card.infinite
  by blast
  show  $\bigwedge bl. \exists x \in \# \mathcal{B}. bl \in \# f x \implies bl \neq \{\}$ 
  proof -
    fix bl
    assume  $\exists x \in \# \mathcal{B}. bl \in \# f x$ 
    then obtain B where  $B \in \# \mathcal{B}$  and  $bl \in \# (f B)$  by auto
    then interpret kgdd: K-GDD  $(B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$  using
  assms by auto
    show  $bl \neq \{\}$  using kgdd.blocks-nempty by (simp add:  $\langle bl \in \# f B \rangle$ )
  qed
  show  $\exists i \in \# \mathcal{B}. f i \neq \{\# \}$ 
  proof -
    obtain B where  $B \in \# \mathcal{B}$ 
      using design-blocks-nempty by auto
    then interpret kgdd: K-GDD  $(B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$  using
  assms by auto
    have  $f B \neq \{\# \}$  using kgdd.design-blocks-nempty by simp
    then show  $\exists i \in \# \mathcal{B}. f i \neq \{\# \}$  using  $\langle B \in \# \mathcal{B} \rangle$  by auto
  qed
  qed

```

**lemma** *pair-construction-block-sizes*:

```

  assumes K-GDD  $(B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$ 
  assumes  $B \in \# \mathcal{B}$ 
  assumes  $b \in \# (f B)$ 
  shows card  $b \in \mathcal{K}'$ 
  proof -
    interpret bkgdd: K-GDD  $(B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$ 
      using assms by simp
    show card  $b \in \mathcal{K}'$  using bkgdd.block-sizes by (simp add:assms)
  qed

```

**lemma** *wilsons-construction-index-0*:

```

  assumes  $\bigwedge B . B \in \# \mathcal{B} \implies$  K-GDD  $(B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$ 
  assumes  $G \in \{GG \times I \mid GG. GG \in \mathcal{G}\}$ 
  assumes  $X \in G$ 
  assumes  $Y \in G$ 
  assumes  $X \neq Y$ 
  shows  $(\sum \# (image-mset f \mathcal{B})) index \{X, Y\} = 0$ 
  proof -
    obtain G' where  $g_i: G = G' \times I$  and  $g_{in}: G' \in \mathcal{G}$  using assms by auto
    obtain x y ix iy where xpair:  $X = (x, ix)$  and ypair:  $Y = (y, iy)$  using assms
  by auto
    then have ixin:  $ix \in I$  and xing:  $x \in G'$  using assms gi by auto
    have iyin:  $iy \in I$  and ying:  $y \in G'$  using assms ypair gi by auto

```

**have** *ne-index-0*:  $x \neq y \implies \mathcal{B} \text{ index } \{x, y\} = 0$   
**using** *ying xing index-together ging by simp*  
**have**  $\bigwedge B. B \in \# \mathcal{B} \implies (f B) \text{ index } \{(x, ix), (y, iy)\} = 0$   
**proof** –  
**fix**  $B$   
**assume** *assm*:  $B \in \# \mathcal{B}$   
**then interpret** *kgdd*:  $K\text{-GDD } (B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$  **using**  
*assms by simp*  
**have** *not-ss-0*:  $\neg (\{(x, ix), (y, iy)\} \subseteq (B \times I)) \implies (f B) \text{ index } \{(x, ix), (y, iy)\} = 0$   
**by** (*metis kgdd.points-index-ps-nin*)  
**have**  $x \neq y \implies \neg \{x, y\} \subseteq B$  **using** *ne-index-0 assm points-index-0-left-imp*  
**by** *auto*  
**then have**  $x \neq y \implies \neg (\{(x, ix), (y, iy)\} \subseteq (B \times I))$  **using** *assms*  
**by** (*meson empty-subsetI insert-subset mem-Sigma-iff*)  
**then have** *nexy*:  $x \neq y \implies (f B) \text{ index } \{(x, ix), (y, iy)\} = 0$  **using** *not-ss-0*  
**by** *simp*  
**have**  $x = y \implies (\{(x, ix), (y, iy)\} \subseteq (B \times I)) \implies (f B) \text{ index } \{(x, ix), (y, iy)\} = 0$   
**proof** –  
**assume** *eq*:  $x = y$   
**assume**  $\{(x, ix), (y, iy)\} \subseteq (B \times I)$   
**then obtain**  $g$  **where**  $g \in \{\{x\} \times I \mid x . x \in B\}$  **and**  $(x, ix) \in g$  **and**  $(y, iy) \in g$   
**using** *eq by auto*  
**then show** *?thesis using kgdd.index-together*  
**by** (*smt (verit, best) SigmaD1 SigmaD2 SigmaI assms(4) assms(5) gi mem-Collect-eq xpair ypair*)  
**qed**  
**then show**  $(f B) \text{ index } \{(x, ix), (y, iy)\} = 0$  **using** *not-ss-0 nexy by auto*  
**qed**  
**then have**  $\bigwedge B. B \in \# (\text{image-mset } f \mathcal{B}) \implies B \text{ index } \{(x, ix), (y, iy)\} = 0$  **by**  
*auto*  
**then show**  $(\sum \# (\text{image-mset } f \mathcal{B})) \text{ index } \{X, Y\} = 0$   
**by** (*simp add: points-index-sum xpair ypair*)  
**qed**

**lemma** *wilsons-construction-index-1*:  
**assumes**  $\bigwedge B. B \in \# \mathcal{B} \implies K\text{-GDD } (B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$   
**assumes**  $G1 \in \{G \times I \mid G. G \in \mathcal{G}\}$   
**assumes**  $G2 \in \{G \times I \mid G. G \in \mathcal{G}\}$   
**assumes**  $G1 \neq G2$   
**and**  $(x, ix) \in G1$  **and**  $(y, iy) \in G2$   
**shows**  $(\sum \# (\text{image-mset } f \mathcal{B})) \text{ index } \{(x, ix), (y, iy)\} = (1 :: \text{int})$   
**proof** –  
**obtain**  $G1'$  **where**  $gi1: G1 = G1' \times I$  **and**  $ging1: G1' \in \mathcal{G}$  **using** *assms by auto*  
**obtain**  $G2'$  **where**  $gi2: G2 = G2' \times I$  **and**  $ging2: G2' \in \mathcal{G}$  **using** *assms by auto*

**have**  $xing: x \in G1'$  **using**  $assms\ gi1$  **by**  $simp$   
**have**  $ying: y \in G2'$  **using**  $assms\ gi2$  **by**  $simp$   
**have**  $gne: G1' \neq G2'$  **using**  $assms\ gi1\ gi2$  **by**  $auto$   
**then have**  $xyne: x \neq y$  **using**  $xing\ ying\ ging1\ ging2\ point-in-one-group$  **by**  $blast$   
**have**  $\exists! bl. bl \in \# \mathcal{B} \wedge \{x, y\} \subseteq bl$  **using**  $index-distinct\ points-index-one-unique-block$   
**by**  $(metis\ ging1\ ging2\ gne\ of-nat-1-eq-iff\ xing\ ying)$   
**then obtain**  $bl$  **where**  $blinb: bl \in \# \mathcal{B}$  **and**  $xyblss: \{x, y\} \subseteq bl$  **by**  $auto$   
**then have**  $\bigwedge b. b \in \# \mathcal{B} - \{\#bl\# \} \implies \neg \{x, y\} \subseteq b$  **using**  $points-index-one-not-unique-block$   
**by**  $(metis\ ging1\ ging2\ gne\ index-distinct\ int-ops(2)\ nat-int-comparison(1)\ xing\ ying)$   
**then have**  $not-ss: \bigwedge b. b \in \# \mathcal{B} - \{\#bl\# \} \implies \neg (\{(x, ix), (y, iy)\} \subseteq (b \times I))$   
**using**  $assms$   
**by**  $(meson\ SigmaD1\ empty-subsetI\ insert-subset)$   
**then have**  $pi0: \bigwedge b. b \in \# \mathcal{B} - \{\#bl\# \} \implies (f\ b)\ index\ \{(x, ix), (y, iy)\} = 0$   
**proof** –  
**fix**  $b$   
**assume**  $assm: b \in \# \mathcal{B} - \{\#bl\# \}$   
**then have**  $b \in \# \mathcal{B}$  **by**  $(meson\ in-diffD)$   
**then interpret**  $kgdd: K-GDD\ (b \times I)\ (f\ b)\ \mathcal{K}'\ \{\{x\} \times I \mid x. x \in b\}$  **using**  
 $assms\ by\ simp$   
**show**  $(f\ b)\ index\ \{(x, ix), (y, iy)\} = 0$   
**using**  $assm\ not-ss\ by\ (metis\ kgdd.points-index-ps-nin)$   
**qed**  
**let**  $?G = \{\{x\} \times I \mid x. x \in bl\}$   
**interpret**  $bkgdd: K-GDD\ (bl \times I)\ (f\ bl)\ \mathcal{K}'\ ?G$  **using**  $assms\ blinb\ by\ simp$   
**obtain**  $g1\ g2$  **where**  $xing1: (x, ix) \in g1$  **and**  $ying2: (y, iy) \in g2$  **and**  $g1g: g1 \in ?G$   
**and**  $g2g: g2 \in ?G$  **using**  $assms(5)\ assms(6)\ gi1\ gi2$   
**by**  $(metis\ (no-types,\ lifting)\ bkgdd.point-has-unique-group\ insert-subset\ mem-Sigma-iff\ xyblss)$   
**then have**  $g1 \neq g2$  **using**  $xyne\ by\ blast$   
**then have**  $pi1: (f\ bl)\ index\ \{(x, ix), (y, iy)\} = 1$   
**using**  $bkgdd.index-distinct\ xing1\ ying2\ g1g\ g2g\ by\ simp$   
**have**  $(\sum \# (image-mset\ f\ \mathcal{B}))\ index\ \{(x, ix), (y, iy)\} =$   
 $(\sum B \in \# \mathcal{B}. (f\ B)\ index\ \{(x, ix), (y, iy)\})$   
**by**  $(simp\ add: points-index-sum)$   
**then have**  $(\sum \# (image-mset\ f\ \mathcal{B}))\ index\ \{(x, ix), (y, iy)\} =$   
 $(\sum B \in \# (\mathcal{B} - \{\#bl\# \}). (f\ B)\ index\ \{(x, ix), (y, iy)\}) + (f\ bl)\ index\ \{(x,$   
 $ix), (y, iy)\}$   
**by**  $(metis\ (no-types,\ lifting)\ add.commute\ blinb\ insert-DiffM\ sum-mset.insert)$   
**thus**  $?thesis$  **using**  $pi0\ pi1\ by\ simp$   
**qed**

**theorem** *Wilson's-Construction:*

**assumes**  $card\ I = w$   
**assumes**  $w > 0$   
**assumes**  $\bigwedge n. n \in \mathcal{K}' \implies n \geq 2$   
**assumes**  $\bigwedge B. B \in \# \mathcal{B} \implies K-GDD\ (B \times I)\ (f\ B)\ \mathcal{K}'\ \{\{x\} \times I \mid x. x \in B\}$   
**shows**  $K-GDD\ (\mathcal{V} \times I)\ (\sum B \in \# \mathcal{B}. (f\ B))\ \mathcal{K}'\ \{G \times I \mid G. G \in \mathcal{G}\}$

```

proof –
  let ?Y =  $\mathcal{V} \times I$  and ?H =  $\{G \times I \mid G . G \in \mathcal{G}\}$  and ?B =  $\sum B \in \# \mathcal{B} . (f B)$ 
  interpret pd: proper-design ?Y ?B using wilsons-construction-proper assms by
  auto
  have  $\bigwedge bl . bl \in \# (\sum B \in \# \mathcal{B} . (f B)) \implies \text{card } bl \in \mathcal{K}'$ 
    using assms pair-construction-block-sizes by blast
  then interpret kdes: K-block-design ?Y ?B  $\mathcal{K}'$ 
    using assms(3) by (unfold-locales) (simp-all,fastforce)
  interpret gdd: GDD ?Y ?B ?H 1:: int
  proof (unfold-locales)
    show partition-on  $(\mathcal{V} \times I) \{G \times I \mid G . G \in \mathcal{G}\}$ 
      using assms groups-not-empty design-points-nempty group-partitions
      by (simp add: partition-on-cart-prod)
    have inj-on  $(\lambda G . G \times I) \mathcal{G}$ 
      using inj-on-def pd.design-points-nempty by auto
    then have  $\text{card } \{G \times I \mid G . G \in \mathcal{G}\} = \text{card } \mathcal{G}$  using card-image by (simp add:
  Setcompr-eq-image)
    then show  $1 < \text{card } \{G \times I \mid G . G \in \mathcal{G}\}$  using groups-size by linarith
    show  $(1::\text{int}) \leq 1$  by simp
    have gdd-fact:  $\bigwedge B . B \in \# \mathcal{B} \implies K\text{-GDD } (B \times I) (f B) \mathcal{K}' \{\{x\} \times I \mid x . x \in B\}$ 
      using assms by simp
    show  $\bigwedge G X Y . G \in \{GG \times I \mid GG . GG \in \mathcal{G}\} \implies X \in G \implies Y \in G \implies X \neq Y$ 
       $\implies (\sum \# (\text{image-mset } f \mathcal{B})) \text{ index } \{X, Y\} = 0$ 
      using wilsons-construction-index-0[OF assms(4)] by auto
    show  $\bigwedge G1 G2 X Y . G1 \in \{G \times I \mid G . G \in \mathcal{G}\} \implies G2 \in \{G \times I \mid G . G \in \mathcal{G}\}$ 
       $\implies G1 \neq G2 \implies X \in G1 \implies Y \in G2 \implies ((\sum \# (\text{image-mset } f \mathcal{B})) \text{ index } \{X, Y\}) = (1::\text{int})$ 
      using wilsons-construction-index-1[OF assms(4)] by blast
    qed
    show ?thesis by (unfold-locales)
  qed

end

context pairwise-balance
begin

lemma PBD-by-deleting-point:
  assumes  $v > 2$ 
  assumes  $\bigwedge bl . bl \in \# \mathcal{B} \implies \text{card } bl \geq 2$ 
  shows pairwise-balance (del-point x) (del-point-blocks x)  $\wedge$ 
proof (cases x  $\in \mathcal{V}$ )
  case True
  interpret des: design del-point x del-point-blocks x
    using delete-point-design assms by blast
  show ?thesis using assms design-blocks-nempty del-point-def del-point-blocks-def
  proof (unfold-locales, simp-all)

```

```

show  $2 < v \implies (\bigwedge bl. bl \in \# \mathcal{B} \implies 2 \leq \text{card } bl) \implies 2 \leq (\text{card } (\mathcal{V} - \{x\}))$ 
using card-Diff-singleton-if diff-diff-cancel diff-le-mono2 finite-sets less-one
by (metis diff-is-0-eq neq0-conv t-lt-order zero-less-diff)
have  $\bigwedge ps. ps \subseteq \mathcal{V} - \{x\} \implies ps \subseteq \mathcal{V}$  by auto
then show  $\bigwedge ps. ps \subseteq \mathcal{V} - \{x\} \implies \text{card } ps = 2 \implies \{\#bl - \{x\}. bl \in \# \mathcal{B}\# \}$ 
index ps =  $\Lambda$ 
using delete-point-index-eq del-point-def del-point-blocks-def by simp
qed
next
case False
then show ?thesis
by (simp add: del-invalid-point del-invalid-point-blocks pairwise-balance-axioms)
qed
end

```

```

context k-GDD
begin

```

```

lemma bibd-from-kGDD:

```

```

assumes  $k > 1$ 
assumes  $\bigwedge g. g \in \mathcal{G} \implies \text{card } g = k - 1$ 
assumes  $x \notin \mathcal{V}$ 
shows bibd (add-point x) ( $\mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}$ ) ( $k$ ) 1
proof -
have  $\bigwedge k. k \in \{k\} \implies k = k$  by blast
then have kge:  $\bigwedge k. k \in \{k\} \implies k \geq 2$  using assms(1) by simp
have  $\bigwedge g. g \in \mathcal{G} \implies \text{card } g + 1 = k$  using assms k-non-zero by auto
then have s:  $(\{k\} \cup \{(\text{card } g) + 1 \mid g. g \in \mathcal{G}\}) = \{k\}$  by auto
then interpret pbd: PBD (add-point x)  $\mathcal{B} + \text{mset-set } \{ \text{insert } x \ g \mid g. g \in \mathcal{G} \}$ 
 $\{k\}$ 
using PBD-by-adjoining-point[of x] kge assms by argo
show ?thesis using assms pbd.block-sizes block-size-lt-v finite-sets add-point-def
by (unfold-locales) (simp-all)
qed

```

```

end

```

```

context PBD
begin

```

```

lemma pbd-points-index1:  $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies \mathcal{B} \text{ index } ps = 1$ 
using balanced by simp

```

```

lemma pbd-index1-points-imply-unique-block:

```

```

assumes  $b1 \in \# \mathcal{B}$  and  $b2 \in \# \mathcal{B}$  and  $b1 \neq b2$ 
assumes  $x \neq y$  and  $\{x, y\} \subseteq b1$  and  $x \in b2$ 
shows  $y \notin b2$ 
proof (rule ccontr)
let ?ps =  $\{\# b \in \# \mathcal{B} . \{x, y\} \subseteq b\# \}$ 

```

```

assume  $\neg y \in b2$ 
then have  $a: y \in b2$  by linarith
then have  $\{x, y\} \subseteq b2$ 
  by (simp add: assms(6))
then have  $b1 \in \# ?ps$  and  $b2 \in \# ?ps$  using assms by auto
then have  $ss: \{\#b1, b2\} \subseteq \# ?ps$  using assms
  by (metis insert-noteq-member mset-add mset-subset-eq-add-mset-cancel single-subset-iff)
have  $size \{\#b1, b2\} = 2$  using assms by auto
then have  $ge2: size ?ps \geq 2$  using assms ss by (metis size-mset-mono)
have  $pair: card \{x, y\} = 2$  using assms by auto
have  $\{x, y\} \subseteq \mathcal{V}$  using assms wellformed by auto
then have  $\mathcal{B} \text{ index } \{x, y\} = 1$  using pbid-points-index1 pair by simp
then show False using points-index-def ge2
  by (metis numeral-le-one-iff semiring-norm(69))
qed

```

```

lemma strong-delete-point-groups-index-zero:
  assumes  $G \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  assumes  $xa \in G$  and  $y \in G$  and  $xa \neq y$ 
  shows (str-del-point-blocks x)  $index \{xa, y\} = 0$ 
proof (auto simp add: points-index-0-iff str-del-point-blocks-def)
  fix  $b$ 
  assume  $a1: b \in \# \mathcal{B}$  and  $a2: x \notin b$  and  $a3: xa \in b$  and  $a4: y \in b$ 
  obtain  $b'$  where  $G = b' - \{x\}$  and  $b' \in \# \mathcal{B}$  and  $x \in b'$  using assms by blast
  then show False using  $a1 a2 a3 a4$  assms pbid-index1-points-imply-unique-block
    by fastforce
qed

```

```

lemma strong-delete-point-groups-index-one:
  assumes  $G1 \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  assumes  $G2 \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  assumes  $G1 \neq G2$  and  $xa \in G1$  and  $y \in G2$ 
  shows (str-del-point-blocks x)  $index \{xa, y\} = 1$ 
proof -
  obtain  $b1$  where  $g1: G1 = b1 - \{x\}$  and  $b1in: b1 \in \# \mathcal{B}$  and  $xin1: x \in b1$ 
using assms by blast
  obtain  $b2$  where  $g2: G2 = b2 - \{x\}$  and  $b2in: b2 \in \# \mathcal{B}$  and  $xin2: x \in b2$ 
using assms by blast
  have  $bneq: b1 \neq b2$  using assms(3) g1 g2 by auto
  have  $xa \neq y$  using  $g1 b1in xin1 g2 b2in xin2$  assms(3) assms(4) assms(5)
insert-subset
  by (smt (verit, best) Diff-eq-empty-iff Diff-iff empty-Diff insertCI pbid-index1-points-imply-unique-block)

  then have  $pair: card \{xa, y\} = 2$  by simp
  have  $inv: \{xa, y\} \subseteq \mathcal{V}$  using  $g1 b1in g2 b2in$  assms(4) assms(5)
  by (metis Diff-cancel Diff-subset insert-Diff insert-subset wellformed)
  have  $\{\# b1 \in \# \mathcal{B} . x \in b1\} \text{ index } \{xa, y\} = 0$ 
  proof (auto simp add: points-index-0-iff)

```

```

fix  $b$  assume  $a1: b \in \# \mathcal{B}$  and  $a2: x \in b$  and  $a3: xa \in b$  and  $a4: y \in b$ 
then have  $yxss: \{y, x\} \subseteq b2$ 
  using  $assms(5)$   $gb2$   $xin2$  by  $blast$ 
have  $\{xa, x\} \subseteq b1$ 
  using  $assms(4)$   $gb1$   $xin1$  by  $auto$ 
then have  $xa \notin b2$  using  $pbd-index1-points-imply-unique-block$ 
  by  $(metis\ DiffE\ assms(4)\ b1in\ b2in\ bneq\ gb1\ singletonI\ xin2)$ 
then have  $b2 \neq b$  using  $a3$  by  $auto$ 
then show  $False$  using  $pbd-index1-points-imply-unique-block$ 
  by  $(metis\ DiffD2\ yxss\ a1\ a2\ a4\ assms(5)\ b2in\ gb2\ insertI1)$ 
qed
then have  $(str-del-point-blocks\ x)\ index\ \{xa, y\} = \mathcal{B}\ index\ \{xa, y\}$ 
by  $(metis\ multiset-partition\ plus-nat.add-0\ point-index-distrib\ str-del-point-blocks-def)$ 

thus  $?thesis$  using  $pbd-points-index1\ pair\ inv$  by  $fastforce$ 
qed

lemma  $blocks-with-x-partition$ :
assumes  $x \in \mathcal{V}$ 
shows  $partition-on\ (\mathcal{V} - \{x\})\ \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
proof  $(intro\ partition-onI)$ 
have  $gtt: \bigwedge bl. bl \in \# \mathcal{B} \implies card\ bl \geq 2$  using  $block-size-gt-t$ 
  using  $block-sizes$  by  $blast$ 
show  $\bigwedge p. p \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\} \implies p \neq \{\}$ 
proof  $-$ 
  fix  $p$  assume  $p \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  then obtain  $b$  where  $ptx: p = b - \{x\}$  and  $b \in \# \mathcal{B}$  and  $xinb: x \in b$  by  $blast$ 
  then have  $ge2: card\ b \geq 2$  using  $gtt$  by  $simp$ 
  then have  $finite\ b$  by  $(metis\ card.infinite\ not-numeral-le-zero)$ 
  then have  $card\ p = card\ b - 1$  using  $xinb\ ptx$  by  $simp$ 
  then have  $card\ p \geq 1$  using  $ge2$  by  $linarith$ 
  thus  $p \neq \{\}$  by  $auto$ 
qed
show  $\bigcup \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\} = \mathcal{V} - \{x\}$ 
proof  $(intro\ subset-antisym\ subsetI)$ 
  fix  $xa$ 
  assume  $xa \in \bigcup \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  then obtain  $b$  where  $xa \in b$  and  $b \in \# \mathcal{B}$  and  $x \in b$  and  $xa \neq x$  by  $auto$ 
  then show  $xa \in \mathcal{V} - \{x\}$  using  $wf-invalid-point$  by  $blast$ 
next
  fix  $xa$ 
  assume  $a: xa \in \mathcal{V} - \{x\}$ 
  then have  $nex: xa \neq x$  by  $simp$ 
  then have  $pair: card\ \{xa, x\} = 2$  by  $simp$ 
  have  $\{xa, x\} \subseteq \mathcal{V}$  using  $a\ assms$  by  $auto$ 
  then have  $card\ \{b \in design-support . \{xa, x\} \subseteq b\} = 1$ 
    using  $balanced\ points-index-simple-def\ pbd-points-index1\ assms$  by  $(metis\ pair)$ 
  then obtain  $b$  where  $des: b \in design-support$  and  $ss: \{xa, x\} \subseteq b$ 

```

```

    by (metis (no-types, lifting) card-1-singletonE mem-Collect-eq singletonI)
  then show  $xa \in \bigcup \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
    using des ss nex design-support-def by auto
qed
show  $\bigwedge p p'. p \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\} \implies p' \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\} \implies$ 
   $p \neq p' \implies p \cap p' = \{\}$ 
proof -
  fix p p'
  assume p1:  $p \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$  and p2:  $p' \in \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  and pne:  $p \neq p'$ 
  then obtain b where b1:  $p = b - \{x\}$  and b1in:  $b \in \# \mathcal{B}$  and xinb1:  $x \in b$  by blast
  then obtain b' where b2:  $p' = b' - \{x\}$  and b2in:  $b' \in \# \mathcal{B}$  and xinb2:  $x \in b'$ 
    using p2 by blast
  then have  $b \neq b'$  using pne b1 by auto
  then have  $\bigwedge y. y \in b \implies y \neq x \implies y \notin b'$ 
    using b1in b2in xinb1 xinb2 pbd-index1-points-imply-unique-block
  by (meson empty-subsetI insert-subset)
  then have  $\bigwedge y. y \in p \implies y \notin p'$ 
  by (metis Diff-iff b1 b2 insertI1)
  then show  $p \cap p' = \{\}$  using disjoint-iff by auto
qed
qed

```

lemma *KGDD-by-deleting-point*:

```

  assumes  $x \in \mathcal{V}$ 
  assumes  $\mathcal{B}$  rep  $x < b$ 
  assumes  $\mathcal{B}$  rep  $x > 1$ 
  shows  $K\text{-GDD}(\text{del-point } x) (\text{str-del-point-blocks } x) \mathcal{K} \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$ 
proof -
  have  $\bigwedge bl. bl \in \# \mathcal{B} \implies \text{card } bl \geq 2$  using block-size-gt-t
  by (simp add: block-sizes)
  then interpret des: proper-design (del-point x) (str-del-point-blocks x)
  using strong-delete-point-proper assms by blast
  show ?thesis using blocks-with-x-partition strong-delete-point-groups-index-zero
    strong-delete-point-groups-index-one str-del-point-blocks-def del-point-def
  proof (unfold locales, simp-all add: block-sizes positive-ints assms)
  have ge1:  $\text{card} \{b. b \in \# \mathcal{B} \wedge x \in b\} > 1$ 
  using assms(3) replication-num-simple-def design-support-def by auto
  have fin: finite  $\{b. b \in \# \mathcal{B} \wedge x \in b\}$  by simp
  have inj: inj-on  $(\lambda b. b - \{x\}) \{b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  using assms(2) inj-on-def mem-Collect-eq by auto
  then have  $\text{card} \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\} = \text{card} \{b. b \in \# \mathcal{B} \wedge x \in b\}$ 
  using card-image fin by (simp add: inj card-image setcompr-eq-image)
  then show  $\text{Suc } 0 < \text{card} \{b - \{x\} \mid b. b \in \# \mathcal{B} \wedge x \in b\}$  using ge1
  
```

```

    by presburger
  qed
qed

lemma card-singletons-eq: card  $\{\{a\} \mid a . a \in A\} = \text{card } A$ 
  by (simp add: card-image Setcompr-eq-image)

lemma KGDD-from-PBD:  $K\text{-GDD } \mathcal{V} \mathcal{B} \mathcal{K} \{\{x\} \mid x . x \in \mathcal{V}\}$ 
proof (unfold-locales, auto simp add: Setcompr-eq-image partition-on-singletons)
  have card  $((\lambda x. \{x\}) ' \mathcal{V}) \geq 2$  using t-lt-order card-singletons-eq
    by (metis Collect-mem-eq setcompr-eq-image)
  then show  $\text{Suc } 0 < \text{card } ((\lambda x. \{x\}) ' \mathcal{V})$  by linarith
  show  $\bigwedge xa \ xb. xa \in \mathcal{V} \implies xb \in \mathcal{V} \implies \mathcal{B} \text{ index } \{xa, xb\} \neq \text{Suc } 0 \implies xa = xb$ 
  proof (rule ccontr)
    fix xa xb
    assume ain:  $xa \in \mathcal{V}$  and bin:  $xb \in \mathcal{V}$  and ne1:  $\mathcal{B} \text{ index } \{xa, xb\} \neq \text{Suc } 0$ 
    assume  $xa \neq xb$ 
    then have  $\text{card } \{xa, xb\} = 2$  by auto
    then have  $\mathcal{B} \text{ index } \{xa, xb\} = 1$ 
      by (simp add: ain bin)
    thus False using ne1 by linarith
  qed
qed
end

context bibd
begin
lemma kGDD-from-bibd:
  assumes  $\Lambda = 1$ 
  assumes  $x \in \mathcal{V}$ 
  shows  $k\text{-GDD } (\text{del-point } x) (\text{str-del-point-blocks } x) \mathcal{k} \{b - \{x\} \mid b . b \in \# \mathcal{B} \wedge x \in b\}$ 
proof -
  interpret pbd:  $PBD \mathcal{V} \mathcal{B} \{\mathcal{k}\}$  using assms
    using PBD.intro  $\Lambda$ -PBD-axioms by auto
  have lt:  $\mathcal{B} \text{ rep } x < b$  using block-num-gt-rep
    by (simp add: assms(2))
  have  $\mathcal{B} \text{ rep } x > 1$  using r-ge-two assms by simp
  then interpret kgdd:  $K\text{-GDD } (\text{del-point } x) \text{str-del-point-blocks } x$ 
     $\{\mathcal{k}\} \{b - \{x\} \mid b . b \in \# \mathcal{B} \wedge x \in b\}$ 
    using pbd.KGDD-by-deleting-point lt assms by blast
  show ?thesis using del-point-def str-del-point-blocks-def by (unfold-locales) (simp-all)
qed

end
end

```

## 8 Graphs and Designs

There are many links between graphs and design - most fundamentally that graphs are designs

```
theory Designs-And-Graphs imports Block-Designs Graph-Theory.Digraph Graph-Theory.Digraph-Componen  
begin
```

### 8.1 Non-empty digraphs

First, we define the concept of a non-empty digraph. This mirrors the idea of a "proper design" in the design theory library

```
locale non-empty-digraph = wf-digraph +  
  assumes arcs-not-empty: arcs G  $\neq$  {}
```

```
begin
```

```
lemma verts-not-empty: verts G  $\neq$  {}  
  using wf arcs-not-empty head-in-verts by fastforce
```

```
end
```

### 8.2 Arcs to Blocks

A digraph uses a pair of points to define an ordered edge. In the case of simple graphs, both possible orderings will be in the arcs set. Blocks are inherently unordered, and as such a method is required to convert between the two representations

```
context graph  
begin
```

```
definition arc-to-block :: 'b  $\Rightarrow$  'a set where  
  arc-to-block e  $\equiv$  {tail G e, head G e}
```

```
lemma arc-to-block-to-ends: {fst (arc-to-ends G e), snd (arc-to-ends G e)} =  
arc-to-block e  
  by (simp add: arc-to-ends-def arc-to-block-def)
```

```
lemma arc-to-block-to-ends-swap: {snd (arc-to-ends G e), fst (arc-to-ends G e)}  
= arc-to-block e  
  using arc-to-block-to-ends  
  by (simp add: arc-to-block-to-ends insert-commute)
```

```
lemma arc-to-ends-to-block: arc-to-block e = {x, y}  $\Longrightarrow$   
  arc-to-ends G e = (x, y)  $\vee$  arc-to-ends G e = (y, x)  
  by (metis arc-to-block-def arc-to-ends-def doubleton-eq-iff)
```

```
lemma arc-to-block-sym: arc-to-ends G e1 = (u, v)  $\Longrightarrow$  arc-to-ends G e2 = (v,  
u)  $\Longrightarrow$ 
```

*arc-to-block e1 = arc-to-block e2*  
**by** (*simp add: arc-to-block-def arc-to-ends-def insert-commute*)

**definition** *arcs-blocks* :: 'a set multiset **where**  
*arcs-blocks*  $\equiv$  *mset-set (arc-to-block ' (arcs G))*

**lemma** *arcs-blocks-ends*:  $(x, y) \in \text{arcs-ends } G \implies \{x, y\} \in \# \text{arcs-blocks}$

**proof** (*auto simp add: arcs-ends-def arcs-blocks-def*)

**fix** *xa*

**assume** *assm1*:  $(x, y) = \text{arc-to-ends } G \text{ xa}$  **and** *assm2*:  $xa \in \text{arcs } G$

**obtain** *z* **where** *zin*:  $z \in (\text{arc-to-block ' (arcs G)})$  **and**  $z = \text{arc-to-block } xa$

**using** *assm2* **by** *blast*

**thus**  $\{x, y\} \in \text{arc-to-block ' (arcs G)}$  **using** *assm1 arc-to-block-to-ends*

**by** (*metis fst-conv snd-conv*)

**qed**

**lemma** *arc-ends-blocks-subset*:  $E \subseteq \text{arcs } G \implies (x, y) \in ((\text{arc-to-ends } G) ' E) \implies$

$\{x, y\} \in (\text{arc-to-block ' } E)$

**by** (*auto simp add: arc-to-ends-def arc-to-block-def*)

**lemma** *arc-blocks-end-subset*: **assumes**  $E \subseteq \text{arcs } G$  **and**  $\{x, y\} \in (\text{arc-to-block ' } E)$

**shows**  $(x, y) \in ((\text{arc-to-ends } G) ' E) \vee (y, x) \in ((\text{arc-to-ends } G) ' E)$

**proof** –

**obtain** *e* **where**  $e \in E$  **and**  $\text{arc-to-block } e = \{x, y\}$  **using** *assms*

**by** *fastforce*

**then have**  $\text{arc-to-ends } G \text{ e} = (x, y) \vee \text{arc-to-ends } G \text{ e} = (y, x)$

**using** *arc-to-ends-to-block by simp*

**thus** *?thesis*

**by** (*metis 'e ∈ E' image-iff*)

**qed**

**lemma** *arcs-ends-blocks*:  $\{x, y\} \in \# \text{arcs-blocks} \implies (x, y) \in \text{arcs-ends } G \wedge (y, x) \in \text{arcs-ends } G$

**proof** (*auto simp add: arcs-ends-def arcs-blocks-def*)

**fix** *xa*

**assume** *assm1*:  $\{x, y\} = \text{arc-to-block } xa$  **and** *assm2*:  $xa \in (\text{arcs } G)$

**obtain** *z* **where** *zin*:  $z \in (\text{arc-to-ends } G ' (\text{arcs } G))$  **and**  $z = \text{arc-to-ends } G \text{ xa}$

**using** *assm2* **by** *blast*

**then have**  $z = (x, y) \vee z = (y, x)$  **using** *arc-to-block-to-ends assm1*

**by** (*metis arc-to-ends-def doubleton-eq-iff fst-conv snd-conv*)

**thus**  $(x, y) \in \text{arc-to-ends } G ' (\text{arcs } G)$  **using** *assm2*

**by** (*metis arcs-ends-def arcs-ends-symmetric sym-arcs zin*)

**next**

**fix** *xa*

**assume** *assm1*:  $\{x, y\} = \text{arc-to-block } xa$  **and** *assm2*:  $xa \in (\text{arcs } G)$

**thus**  $(y, x) \in \text{arc-to-ends } G ' \text{arcs } G$  **using** *arcs-ends-def*

**by** (*metis dual-order.refl graph.arc-blocks-end-subset graph-axioms graph-symmetric*)

*imageI*)  
**qed**

**lemma** *arcs-blocks-iff*:  $\{x, y\} \in \# \text{ arcs-blocks} \longleftrightarrow (x, y) \in \text{arcs-ends } G \wedge (y, x) \in \text{arcs-ends } G$   
**using** *arcs-ends-blocks arcs-blocks-ends* **by** *blast*

**lemma** *arcs-ends-wf*:  $(x, y) \in \text{arcs-ends } G \implies x \in \text{verts } G \wedge y \in \text{verts } G$   
**by** *auto*

**lemma** *arcs-blocks-elem*:  $bl \in \# \text{ arcs-blocks} \implies \exists x y . bl = \{x, y\}$   
**apply** (*auto simp add: arcs-blocks-def*)  
**by** (*meson arc-to-block-def*)

**lemma** *arcs-ends-blocks-wf*:  
**assumes**  $bl \in \# \text{ arcs-blocks}$   
**shows**  $bl \subseteq \text{verts } G$

**proof** –

**obtain**  $x y$  **where**  $blpair: bl = \{x, y\}$  **using** *arcs-blocks-elem assms*  
**by** *fastforce*

**then have**  $(x, y) \in \text{arcs-ends } G$  **using** *arcs-ends-blocks assms* **by** *simp*  
**thus** *?thesis* **using** *arcs-ends-wf blpair* **by** *auto*

**qed**

**lemma** *arcs-blocks-simple*:  $bl \in \# \text{ arcs-blocks} \implies \text{count} (\text{arcs-blocks}) bl = 1$   
**by** (*simp add: arcs-blocks-def*)

**lemma** *arcs-blocks-ne*:  $\text{arcs } G \neq \{\} \implies \text{arcs-blocks} \neq \{\#\}$   
**by** (*simp add: arcs-blocks-iff arcs-blocks-def mset-set-empty-iff*)

**end**

### 8.3 Graphs are designs

Prove that a graph is a number of different types of designs

**sublocale** *graph*  $\subseteq$  *design* *verts*  $G$  *arcs-blocks*  
**using** *arcs-ends-blocks-wf arcs-blocks-elem* **by** (*unfold-locales*) (*auto*)

**sublocale** *graph*  $\subseteq$  *simple-design* *verts*  $G$  *arcs-blocks*  
**using** *arcs-ends-blocks-wf arcs-blocks-elem arcs-blocks-simple* **by** (*unfold-locales*)  
(*auto*)

**locale** *non-empty-graph* = *graph* + *non-empty-digraph*

**sublocale** *non-empty-graph*  $\subseteq$  *proper-design* *verts*  $G$  *arcs-blocks*  
**using** *arcs-blocks-ne arcs-not-empty* **by** (*unfold-locales*) *simp*

**lemma** (*in graph*) *graph-block-size*: **assumes**  $bl \in \# \text{ arcs-blocks}$  **shows**  $\text{card } bl = 2$

**proof** –  
**obtain**  $x\ y$  **where**  $blrep: bl = \{x, y\}$  **using** *assms arcs-blocks-elem*  
**by** *fastforce*  
**then have**  $(x, y) \in arcs\text{-ends } G$  **using** *arcs-ends-blocks assms* **by** *simp*  
**then have**  $x \neq y$  **using** *no-loops* **using** *adj-not-same* **by** *blast*  
**thus** *?thesis* **using** *blrep* **by** *simp*  
**qed**

**sublocale** *non-empty-graph*  $\subseteq$  *block-design* *verts G arcs-blocks 2*  
**using** *arcs-not-empty graph-block-size arcs-blocks-ne* **by** (*unfold-locales*) *simp-all*

## 8.4 R-regular graphs

To reason on r-regular graphs and their link to designs, we require a number of extensions to lemmas reasoning around the degrees of vertices

**context** *sym-digraph*  
**begin**

**lemma** *in-out-arcs-reflexive*:  $v \in verts\ G \implies (e \in (in\text{-arcs } G\ v) \implies$   
 $\exists e' . (e' \in (out\text{-arcs } G\ v) \wedge head\ G\ e' = tail\ G\ e))$   
**using** *symmetric-conv sym-arcs* **by** *fastforce*

**lemma** *out-in-arcs-reflexive*:  $v \in verts\ G \implies (e \in (out\text{-arcs } G\ v) \implies$   
 $\exists e' . (e' \in (in\text{-arcs } G\ v) \wedge tail\ G\ e' = head\ G\ e))$   
**using** *symmetric-conv sym-arcs* **by** *fastforce*

**end**

**context** *nomulti-digraph*  
**begin**

**lemma** *in-arcs-single-per-vert*:  
**assumes**  $v \in verts\ G$  **and**  $u \in verts\ G$   
**assumes**  $e1 \in in\text{-arcs } G\ v$  **and**  $e2 \in in\text{-arcs } G\ v$   
**assumes**  $tail\ G\ e1 = u$  **and**  $tail\ G\ e2 = u$   
**shows**  $e1 = e2$

**proof** –  
**have** *in-arcs1*:  $e1 \in arcs\ G$  **and** *in-arcs2*:  $e2 \in arcs\ G$  **using** *assms* **by** *auto*  
**have** *arc-to-ends*  $G\ e1 = arc\text{-to-ends } G\ e2$  **using** *assms arc-to-ends-def*  
**by** (*metis in-in-arcs-conv*)  
**thus** *?thesis* **using** *in-arcs1 in-arcs2 no-multi-arcs* **by** *simp*  
**qed**

**lemma** *out-arcs-single-per-vert*:  
**assumes**  $v \in verts\ G$  **and**  $u \in verts\ G$   
**assumes**  $e1 \in out\text{-arcs } G\ v$  **and**  $e2 \in out\text{-arcs } G\ v$   
**assumes**  $head\ G\ e1 = u$  **and**  $head\ G\ e2 = u$   
**shows**  $e1 = e2$

**proof** –

```

have in-arcs1:  $e1 \in \text{arcs } G$  and in-arcs2:  $e2 \in \text{arcs } G$  using assms by auto
have arc-to-ends  $G$   $e1 = \text{arc-to-ends } G$   $e2$  using assms arc-to-ends-def
by (metis in-out-arcs-conv)
thus ?thesis using in-arcs1 in-arcs2 no-multi-arcs by simp
qed

```

**end**

Some helpers on the transformation arc definition

```

context graph
begin

```

```

lemma arc-to-block-is-inj-in-arcs: inj-on arc-to-block (in-arcs  $G$   $v$ )
apply (auto simp add: arc-to-block-def inj-on-def)
by (metis arc-to-ends-def doubleton-eq-iff no-multi-arcs)

```

```

lemma arc-to-block-is-inj-out-arcs: inj-on arc-to-block (out-arcs  $G$   $v$ )
apply (auto simp add: arc-to-block-def inj-on-def)
by (metis arc-to-ends-def doubleton-eq-iff no-multi-arcs)

```

```

lemma in-out-arcs-reflexive-uniq:  $v \in \text{verts } G \implies (e \in (\text{in-arcs } G$   $v) \implies$ 
 $\exists! e' . (e' \in (\text{out-arcs } G$   $v) \wedge \text{head } G$   $e' = \text{tail } G$   $e))$ 
apply auto
using symmetric-conv apply fastforce
using out-arcs-single-per-vert by (metis head-in-verts in-out-arcs-conv)

```

```

lemma out-in-arcs-reflexive-uniq:  $v \in \text{verts } G \implies e \in (\text{out-arcs } G$   $v) \implies$ 
 $\exists! e' . (e' \in (\text{in-arcs } G$   $v) \wedge \text{tail } G$   $e' = \text{head } G$   $e)$ 
apply auto
using symmetric-conv apply fastforce
using in-arcs-single-per-vert by (metis tail-in-verts in-in-arcs-conv)

```

```

lemma in-eq-out-arc-ends:  $(u, v) \in ((\text{arc-to-ends } G) \text{ ' } (\text{in-arcs } G$   $v)) \iff$ 
 $(v, u) \in ((\text{arc-to-ends } G) \text{ ' } (\text{out-arcs } G$   $v))$ 
unfolding arc-to-ends-def in-in-arcs-conv in-out-arcs-conv
using arcs-ends-conv graph-symmetric by fastforce

```

```

lemma in-degree-eq-card-arc-ends:  $\text{in-degree } G$   $v = \text{card } ((\text{arc-to-ends } G) \text{ ' } (\text{in-arcs}$ 
 $G$   $v))$ 
apply (simp add: in-degree-def)
using no-multi-arcs by (metis card-image in-arcs-in-arcs inj-onI)

```

```

lemma in-degree-eq-card-arc-blocks:  $\text{in-degree } G$   $v = \text{card } (\text{arc-to-block} \text{ ' } (\text{in-arcs}$ 
 $G$   $v))$ 
apply (simp add: in-degree-def)
using no-multi-arcs arc-to-block-is-inj-in-arcs by (simp add: card-image)

```

```

lemma out-degree-eq-card-arc-blocks:  $\text{out-degree } G$   $v = \text{card } (\text{arc-to-block} \text{ ' } (\text{out-arcs}$ 
 $G$   $v))$ 

```

```

apply (simp add: out-degree-def)
using no-multi-arcs arc-to-block-is-inj-out-arcs by (simp add: card-image)

lemma out-degree-eq-card-arc-ends: out-degree  $G v = \text{card } ((\text{arc-to-ends } G) \text{ ` } (\text{out-arcs } G v))$ 
apply (simp add: out-degree-def)
using no-multi-arcs by (metis card-image out-arcs-in-arcs inj-onI)

lemma bij-betw-in-out-arcs: bij-betw  $(\lambda (u, v) . (v, u)) ((\text{arc-to-ends } G) \text{ ` } (\text{in-arcs } G v))$ 
   $((\text{arc-to-ends } G) \text{ ` } (\text{out-arcs } G v))$ 
apply (auto simp add: bij-betw-def)
apply (simp add: swap-inj-on)
apply (metis Pair-inject arc-to-ends-def image-eqI in-eq-out-arc-ends in-in-arcs-conv)
by (metis arc-to-ends-def imageI in-eq-out-arc-ends in-out-arcs-conv pair-imageI)

lemma in-eq-out-degree: in-degree  $G v = \text{out-degree } G v$ 
using bij-betw-in-out-arcs bij-betw-same-card in-degree-eq-card-arc-ends
  out-degree-eq-card-arc-ends by auto

lemma in-out-arcs-blocks: arc-to-block  $\text{ ` } (\text{in-arcs } G v) = \text{arc-to-block } \text{ ` } (\text{out-arcs } G v)$ 
proof (auto)
  fix  $xa$ 
  assume  $a1: xa \in \text{arcs } G$  and  $a2: v = \text{head } G xa$ 
  then have  $xa \in \text{in-arcs } G v$  by simp
  then obtain  $e$  where  $out: e \in \text{out-arcs } G v$  and  $\text{head } G e = \text{tail } G xa$ 
    using out-in-arcs-reflexive-uniq by force
  then have  $\text{arc-to-ends } G e = (v, \text{tail } G xa)$ 
    by (simp add: arc-to-ends-def)
  then have  $\text{arc-to-block } xa = \text{arc-to-block } e$ 
    using arc-to-block-sym by (metis a2 arc-to-ends-def)
  then show  $\text{arc-to-block } xa \in \text{arc-to-block } \text{ ` } \text{out-arcs } G (\text{head } G xa)$ 
    using out a2 by blast
next
  fix  $xa$ 
  assume  $a1: xa \in \text{arcs } G$  and  $a2: v = \text{tail } G xa$ 
  then have  $xa \in \text{out-arcs } G v$  by simp
  then obtain  $e$  where  $ina: e \in \text{in-arcs } G v$  and  $\text{tail } G e = \text{head } G xa$ 
    using out-in-arcs-reflexive-uniq by force
  then have  $\text{arc-to-ends } G e = (\text{head } G xa, v)$ 
    by (simp add: arc-to-ends-def)
  then have  $\text{arc-to-block } xa = \text{arc-to-block } e$ 
    using arc-to-block-sym by (metis a2 arc-to-ends-def)
  then show  $\text{arc-to-block } xa \in \text{arc-to-block } \text{ ` } \text{in-arcs } G (\text{tail } G xa)$ 
    using ina a2 by blast
qed

end

```

A regular digraph is defined as one where the in degree equals the out degree which in turn equals some fixed integer  $r$

**locale** *regular-digraph* = *wf-digraph* +

**fixes**  $r :: \text{nat}$

**assumes** *in-deg-r*:  $v \in \text{verts } G \implies \text{in-degree } G \ v = r$

**assumes** *out-deg-r*:  $v \in \text{verts } G \implies \text{out-degree } G \ v = r$

**locale** *regular-graph* = *graph* + *regular-digraph*

**begin**

**lemma** *rep-vertices-in-blocks* [*simp*]:

**assumes**  $x \in \text{verts } G$

**shows**  $\text{size } \{\# \ e \in \# \ \text{arcs-blocks} \ . \ x \in e \ \#\} = r$

**proof** –

**have**  $\bigwedge e . e \in (\text{arc-to-block } ' (\text{arcs } G)) \implies x \in e \implies e \in (\text{arc-to-block } ' \text{in-arcs } G \ x)$

**using** *arc-to-block-def in-in-arcs-conv insert-commute insert-iff singleton-iff sym-arcs*

*symmetric-conv* **by** *fastforce*

**then have**  $\{ e \in (\text{arc-to-block } ' (\text{arcs } G)) . x \in e \} = (\text{arc-to-block } ' (\text{in-arcs } G \ x))$

**using** *arc-to-block-def* **by** *auto*

**then have**  $\text{card } \{ e \in (\text{arc-to-block } ' (\text{arcs } G)) . x \in e \} = r$

**using** *in-deg-r in-degree-eq-card-arc-blocks assms* **by** *auto*

**thus** *?thesis*

**using** *arcs-blocks-def finite-arcs* **by** *force*

**qed**

**end**

Intro rules for regular graphs

**lemma** *graph-in-degree-r-imp-reg*[*intro*]: **assumes** *graph*  $G$

**assumes**  $(\bigwedge v . v \in (\text{verts } G) \implies \text{in-degree } G \ v = r)$

**shows** *regular-graph*  $G \ r$

**proof** –

**interpret**  $g$ : *graph*  $G$  **using** *assms* **by** *simp*

**interpret**  $wf$ : *wf-digraph*  $G$  **by** (*simp add: g.wf-digraph-axioms*)

**show** *?thesis*

**using** *assms(2) g.in-eq-out-degree* **by** (*unfold-locales simp-all*)

**qed**

**lemma** *graph-out-degree-r-imp-reg*[*intro*]: **assumes** *graph*  $G$

**assumes**  $(\bigwedge v . v \in (\text{verts } G) \implies \text{out-degree } G \ v = r)$

**shows** *regular-graph*  $G \ r$

**proof** –

**interpret**  $g$ : *graph*  $G$  **using** *assms* **by** *simp*

**interpret**  $wf$ : *wf-digraph*  $G$  **by** (*simp add: g.wf-digraph-axioms*)

**show** *?thesis*

**using** *assms(2) g.in-eq-out-degree* **by** (*unfold-locales simp-all*)

**qed**

Regular graphs (non-empty) can be shown to be a constant rep design

**locale** *non-empty-regular-graph* = *regular-graph* + *non-empty-digraph*

**sublocale** *non-empty-regular-graph*  $\subseteq$  *non-empty-graph*  
**by** *unfold-locales*

**sublocale** *non-empty-regular-graph*  $\subseteq$  *constant-rep-design* *verts* *G* *arcs-blocks* *r*  
**using** *arcs-blocks-ne* *arcs-not-empty*  
**by** (*unfold-locales*)(*simp-all* *add: point-replication-number-def*)

**end**

## 9 Sub-designs

Sub designs are a relationship between two designs using the subset and submultiset relations This theory defines the concept at the incidence system level, before extending to defining on well defined designs

**theory** *Sub-Designs* **imports** *Design-Operations*  
**begin**

### 9.1 Sub-system and Sub-design Locales

**locale** *sub-set-system* = *incidence-system*  $\mathcal{V}$   $\mathcal{B}$   
**for**  $\mathcal{U}$  **and**  $\mathcal{A}$  **and**  $\mathcal{V}$  **and**  $\mathcal{B}$  +  
**assumes** *points-subset*:  $\mathcal{U} \subseteq \mathcal{V}$   
**assumes** *blocks-subset*:  $\mathcal{A} \subseteq \# \mathcal{B}$   
**begin**

**lemma** *sub-points*:  $x \in \mathcal{U} \implies x \in \mathcal{V}$   
**using** *points-subset* **by** *auto*

**lemma** *sub-blocks*:  $bl \in \# \mathcal{A} \implies bl \in \# \mathcal{B}$   
**using** *blocks-subset* **by** *auto*

**lemma** *sub-blocks-count*:  $\text{count } \mathcal{A} \leq \text{count } \mathcal{B}$   
**by** (*simp* *add: mset-subset-eq-count* *blocks-subset*)

**end**

**locale** *sub-incidence-system* = *sub-set-system* + *ins: incidence-system*  $\mathcal{U}$   $\mathcal{A}$

**locale** *sub-design* = *sub-incidence-system* + *des: design*  $\mathcal{V}$   $\mathcal{B}$   
**begin**

**lemma** *sub-non-empty-blocks*:  $A \in \# \mathcal{A} \implies A \neq \{\}$   
**using** *des.blocks-nempty* *sub-blocks* **by** *simp*

```

sublocale sub-des: design  $\mathcal{U} \mathcal{A}$ 
  using des.finite-sets finite-subset points-subset sub-non-empty-blocks
  by (unfold-locales) (auto)

end

locale proper-sub-set-system = incidence-system  $\mathcal{V} \mathcal{B}$ 
  for  $\mathcal{U}$  and  $\mathcal{A}$  and  $\mathcal{V}$  and  $\mathcal{B}$  +
  assumes points-psubset:  $\mathcal{U} \subseteq \mathcal{V}$ 
  assumes blocks-subset:  $\mathcal{A} \subseteq \# \mathcal{B}$ 
begin

lemma point-sets-ne:  $\mathcal{U} \neq \mathcal{V}$ 
  using points-psubset by auto

end

sublocale proper-sub-set-system  $\subseteq$  sub-set-system
  using points-psubset blocks-subset by (unfold-locales) simp-all

context sub-set-system
begin

lemma sub-is-proper:  $\mathcal{U} \neq \mathcal{V} \implies$  proper-sub-set-system  $\mathcal{U} \mathcal{A} \mathcal{V} \mathcal{B}$ 
  using blocks-subset incidence-system-axioms
  by (simp add: points-subset proper-sub-set-system.intro proper-sub-set-system-axioms-def
psubsetI)

end

locale proper-sub-incidence-system = proper-sub-set-system + ins: incidence-system
 $\mathcal{U} \mathcal{A}$ 

sublocale proper-sub-incidence-system  $\subseteq$  sub-incidence-system
  by (unfold-locales)

context sub-incidence-system
begin
lemma sub-is-proper:  $\mathcal{U} \neq \mathcal{V} \implies$  proper-sub-incidence-system  $\mathcal{U} \mathcal{A} \mathcal{V} \mathcal{B}$ 
  by (simp add: ins.incidence-system-axioms proper-sub-incidence-system-def sub-is-proper)

end

locale proper-sub-design = proper-sub-incidence-system + des: design  $\mathcal{V} \mathcal{B}$ 

sublocale proper-sub-design  $\subseteq$  sub-design
  by (unfold-locales)

```

**context** *sub-design*

**begin**

**lemma** *sub-is-proper*:  $\mathcal{U} \neq \mathcal{V} \implies \text{proper-sub-design } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$

**by** (*simp add: des.wf-design proper-sub-design.intro sub-is-proper*)

**end**

**lemma** *ss-proper-implies-sub* [*intro*]:  $\text{proper-sub-set-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B} \implies \text{sub-set-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$

**using** *proper-sub-set-system.axioms(1) proper-sub-set-system.blocks-subset psubsetE*

**by** (*metis proper-sub-set-system.points-psubset sub-set-system.intro sub-set-system-axioms-def*)

**lemma** *sub-ssI* [*intro!*]:  $\text{incidence-system } \mathcal{V} \ \mathcal{B} \implies \mathcal{U} \subseteq \mathcal{V} \implies \mathcal{A} \subseteq_{\#} \mathcal{B} \implies \text{sub-set-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$

**using** *incidence-system-def subset-iff*

**by** (*unfold-locales (simp-all add: incidence-system.wellformed)*)

**lemma** *sub-ss-equality*:

**assumes** *sub-set-system*  $\mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$

**and** *sub-set-system*  $\mathcal{V} \ \mathcal{B} \ \mathcal{U} \ \mathcal{A}$

**shows**  $\mathcal{U} = \mathcal{V}$  **and**  $\mathcal{A} = \mathcal{B}$

**using** *assms(1) assms(2) sub-set-system.points-subset apply blast*

**by** (*meson assms(1) assms(2) sub-set-system.blocks-subset subset-mset.eq-iff*)

## 9.2 Reasoning on Sub-designs

### 9.2.1 Reasoning on Incidence Sys property relationships

**context** *sub-incidence-system*

**begin**

**lemma** *sub-sys-block-sizes*:  $\text{ins.sys-block-sizes} \subseteq \text{sys-block-sizes}$

**by** (*auto simp add: sys-block-sizes-def ins.sys-block-sizes-def blocks-subset sub-blocks*)

**lemma** *sub-point-rep-number-le*:  $x \in \mathcal{U} \implies \mathcal{A} \text{ rep } x \leq \mathcal{B} \text{ rep } x$

**by** (*simp add: point-replication-number-def blocks-subset multiset-filter-mono size-mset-mono*)

**lemma** *sub-point-index-le*:  $ps \subseteq \mathcal{U} \implies \mathcal{A} \text{ index } ps \leq \mathcal{B} \text{ index } ps$

**by** (*simp add: points-index-def blocks-subset multiset-filter-mono size-mset-mono*)

**lemma** *sub-sys-intersection-numbers*:  $\text{ins.intersection-numbers} \subseteq \text{intersection-numbers}$

**apply** (*auto simp add: intersection-numbers-def ins.intersection-numbers-def*)

**by** (*metis blocks-subset insert-DiffM insert-subset-eq-iff*)

**end**

### 9.2.2 Reasoning on Incidence Sys/Design operations

**context** *incidence-system*

**begin**

**lemma** *sub-set-sysI*[*intro*]:  $\mathcal{U} \subseteq \mathcal{V} \implies \mathcal{A} \subseteq \# \mathcal{B} \implies \text{sub-set-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$   
**by** (*simp add: sub-ssI incidence-system-axioms*)

**lemma** *sub-inc-sysI*[*intro*]:  $\text{incidence-system } \mathcal{U} \ \mathcal{A} \implies \mathcal{U} \subseteq \mathcal{V} \implies \mathcal{A} \subseteq \# \mathcal{B} \implies$   
 $\text{sub-incidence-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$   
**by** (*simp add: sub-incidence-system.intro sub-set-sysI*)

**lemma** *multiple-orig-sub-system*:  
**assumes**  $n > 0$   
**shows**  $\text{sub-incidence-system } \mathcal{V} \ \mathcal{B} \ \mathcal{V}$  (*multiple-blocks n*)  
**using** *multiple-block-in-original wellformed multiple-blocks-sub assms*  
**by** (*unfold-locales simp-all*)

**lemma** *add-point-sub-sys*:  $\text{sub-incidence-system } \mathcal{V} \ \mathcal{B} \ (\text{add-point } p) \ \mathcal{B}$   
**using** *add-point-wf add-point-def*  
**by** (*simp add: sub-ssI subset-insertI incidence-system-axioms sub-incidence-system.intro*)

**lemma** *strong-del-point-sub-sys*:  $\text{sub-incidence-system } (\text{del-point } p) \ (\text{str-del-point-blocks } p) \ \mathcal{V} \ \mathcal{B}$   
**using** *strong-del-point-incidence-wf wf-invalid-point del-point-def str-del-point-blocks-def*  
  
**by** (*unfold-locales auto*)

**lemma** *add-block-sub-sys*:  $\text{sub-incidence-system } \mathcal{V} \ \mathcal{B} \ (\mathcal{V} \cup b) \ (\text{add-block } b)$   
**using** *add-block-wf wf-invalid-point add-block-def* **by** (*unfold-locales auto*)

**lemma** *delete-block-sub-sys*:  $\text{sub-incidence-system } \mathcal{V} \ (\text{del-block } b) \ \mathcal{V} \ \mathcal{B}$   
**using** *delete-block-wf delete-block-subset incidence-system-def* **by** (*unfold-locales, auto*)

**end**

**lemma** (*in two-set-systems*) *combine-sub-sys*:  $\text{sub-incidence-system } \mathcal{V} \ \mathcal{B} \ \mathcal{V}^+ \ \mathcal{B}^+$   
**by** (*unfold-locales simp-all*)

**lemma** (*in two-set-systems*) *combine-sub-sys-alt*:  $\text{sub-incidence-system } \mathcal{V}' \ \mathcal{B}' \ \mathcal{V}^+ \ \mathcal{B}^+$   
**by** (*unfold-locales simp-all*)

**context** *design*

**begin**

**lemma** *sub-designI* [*intro*]:  $\text{design } \mathcal{U} \ \mathcal{A} \implies \text{sub-incidence-system } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B} \implies$   
 $\text{sub-design } \mathcal{U} \ \mathcal{A} \ \mathcal{V} \ \mathcal{B}$   
**by** (*simp add: sub-design.intro wf-design*)

**lemma** *sub-designII* [intro]:  $design\ U\ \mathcal{A} \implies sub-incidence-system\ \mathcal{V}\ \mathcal{B}\ U\ \mathcal{A} \implies sub-design\ \mathcal{V}\ \mathcal{B}\ U\ \mathcal{A}$

**by** (*simp add: sub-design-def*)

**lemma** *multiple-orig-sub-des*:

**assumes**  $n > 0$

**shows**  $sub-design\ \mathcal{V}\ \mathcal{B}\ \mathcal{V}\ (multiple-blocks\ n)$

**using** *multiple-is-design assms multiple-orig-sub-system* **by** (*simp add: sub-design.intro*)

**lemma** *add-point-sub-des*:  $sub-design\ \mathcal{V}\ \mathcal{B}\ (add-point\ p)\ \mathcal{B}$

**using** *add-point-design add-point-sub-sys sub-design.intro* **by** *fastforce*

**lemma** *strong-del-point-sub-des*:  $sub-design\ (del-point\ p)\ (str-del-point-blocks\ p)\ \mathcal{V}\ \mathcal{B}$

**using** *strong-del-point-sub-sys sub-design.intro wf-design* **by** *blast*

**lemma** *add-block-sub-des*:  $finite\ b \implies b \neq \{\} \implies sub-design\ \mathcal{V}\ \mathcal{B}\ (\mathcal{V} \cup b)\ (add-block\ b)$

**using** *add-block-sub-sys add-block-design sub-designII* **by** *fastforce*

**lemma** *delete-block-sub-des*:  $sub-design\ \mathcal{V}\ (del-block\ b)\ \mathcal{V}\ \mathcal{B}$

**using** *delete-block-design delete-block-sub-sys sub-designI* **by** *auto*

**end**

**lemma** (*in two-designs*) *combine-sub-des*:  $sub-design\ \mathcal{V}\ \mathcal{B}\ \mathcal{V}^+\ \mathcal{B}^+$

**by** (*unfold-locales simp-all*)

**lemma** (*in two-designs*) *combine-sub-des-alt*:  $sub-design\ \mathcal{V}'\ \mathcal{B}'\ \mathcal{V}^+\ \mathcal{B}^+$

**by** (*unfold-locales simp-all*)

**end**

## 10 Design Isomorphisms

**theory** *Design-Isomorphisms* **imports** *Design-Basics Sub-Designs*

**begin**

### 10.1 Images of Set Systems

We loosely define the concept of taking the "image" of a set system, as done in isomorphisms. Note that this is not based off mathematical theory, but is for ease of notation

**definition** *blocks-image* ::  $'a\ set\ multiset \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b\ set\ multiset$  **where**  
*blocks-image*  $B\ f \equiv image-mset\ ((\cdot)\ f)\ B$

**lemma** *image-block-set-constant-size*:  $size\ (B) = size\ (blocks-image\ B\ f)$

by (simp add: blocks-image-def)

**lemma** (in incidence-system) image-set-system-wellformed:  
 incidence-system (f ' V) (blocks-image B f)  
 by (unfold-locales, auto simp add: blocks-image-def) (meson image-eqI wf-invalid-point)

**lemma** (in finite-incidence-system) image-set-system-finite:  
 finite-incidence-system (f ' V) (blocks-image B f)  
 using image-set-system-wellformed finite-sets  
 by (intro-locales) (simp-all add: blocks-image-def finite-incidence-system-axioms.intro)

## 10.2 Incidence System Isomorphisms

Isomorphism's are defined by the Handbook of Combinatorial Designs [3]

**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** iso-eq-order: card V = card V'  
 using bij bij-betw-same-card by auto

**lemma** iso-eq-block-num: size B = size B'  
 using block-img by (metis size-image-mset)

**lemma** iso-block-img-alt-rep: {# π ' bl . bl ∈# B#} = B'  
 using block-img by simp

**lemma** inv-iso-block-img: image-mset ((⟨⟩) (inv-into V π)) B' = B  
**proof** –

have  $\bigwedge x. x \in V \implies ((inv-into V \pi) \circ \pi) x = x$   
 using bij bij-betw-inv-into-left comp-apply by fastforce  
 then have  $\bigwedge bl x. bl \in \# B \implies x \in bl \implies ((inv-into V \pi) \circ \pi) x = x$   
 using source.wellformed by blast  
 then have  $img: \bigwedge bl. bl \in \# B \implies image ((inv-into V \pi) \circ \pi) bl = bl$   
 by simp  
 have  $image-mset ((⟨⟩) (inv-into V \pi)) B' = image-mset ((⟨⟩) (inv-into V \pi))$   
 $(image-mset ((⟨⟩) \pi) B)$   
 using block-img by simp  
 then have  $image-mset ((⟨⟩) (inv-into V \pi)) B' = image-mset ((⟨⟩) ((inv-into V \pi)$   
 $\circ \pi)) B$   
 by (metis (no-types, opaque-lifting) block-img comp-apply image-comp multi-  
 set.map-comp multiset.map-cong0)  
 thus ?thesis using img by simp  
**qed**

**lemma** inverse-incidence-sys-iso: incidence-system-isomorphism V' B' V B (inv-into

$\mathcal{V} \pi$ )  
**using** *bij bij-betw-inv-into inv-iso-block-img* **by** (*unfold-locales*) *simp*

**lemma** *iso-points-map*:  $\pi \text{ ' } \mathcal{V} = \mathcal{V}'$   
**using** *bij* **by** (*simp add: bij-betw-imp-surj-on*)

**lemma** *iso-points-inv-map*:  $(\text{inv-into } \mathcal{V} \pi) \text{ ' } \mathcal{V}' = \mathcal{V}$   
**using** *incidence-system-isomorphism.iso-points-map inverse-incidence-sys-iso* **by** *blast*

**lemma** *iso-points-ss-card*:  
**assumes**  $ps \subseteq \mathcal{V}$   
**shows**  $\text{card } ps = \text{card } (\pi \text{ ' } ps)$   
**using** *assms bij bij-betw-same-card bij-betw-subset* **by** *blast*

**lemma** *iso-block-in*:  $bl \in \# \mathcal{B} \implies (\pi \text{ ' } bl) \in \# \mathcal{B}'$   
**using** *iso-block-img-alt-rep*  
**by** (*metis image-eqI in-image-mset*)

**lemma** *iso-inv-block-in*:  $x \in \# \mathcal{B}' \implies x \in ({}^{\circ}) \pi \text{ ' } \text{set-mset } \mathcal{B}$   
**by** (*metis block-img in-image-mset*)

**lemma** *iso-img-block-orig-exists*:  $x \in \# \mathcal{B}' \implies \exists bl . bl \in \# \mathcal{B} \wedge x = \pi \text{ ' } bl$   
**using** *iso-inv-block-in* **by** *blast*

**lemma** *iso-blocks-map-inj*:  $x \in \# \mathcal{B} \implies y \in \# \mathcal{B} \implies \pi \text{ ' } x = \pi \text{ ' } y \implies x = y$   
**using** *image-inv-into-cancel incidence-system.wellformed iso-points-inv-map iso-points-map*  
**by** (*metis (no-types, lifting) source.incidence-system-axioms subset-image-iff*)

**lemma** *iso-bij-betwn-block-sets*: *bij-betw*  $(({}^{\circ}) \pi) (\text{set-mset } \mathcal{B}) (\text{set-mset } \mathcal{B}')$   
**apply** (*simp add: bij-betw-def inj-on-def*)  
**using** *iso-block-in iso-inv-block-in iso-blocks-map-inj* **by** *auto*

**lemma** *iso-bij-betwn-block-sets-inv*: *bij-betw*  $(({}^{\circ}) (\text{inv-into } \mathcal{V} \pi)) (\text{set-mset } \mathcal{B}') (\text{set-mset } \mathcal{B})$   
**using** *incidence-system-isomorphism.iso-bij-betwn-block-sets inverse-incidence-sys-iso*  
**by** *blast*

**lemma** *iso-bij-betw-individual-blocks*:  $bl \in \# \mathcal{B} \implies \text{bij-betw } \pi \text{ ' } bl (\pi \text{ ' } bl)$   
**using** *bij bij-betw-subset source.wellformed* **by** *blast*

**lemma** *iso-bij-betw-individual-blocks-inv*:  $bl \in \# \mathcal{B} \implies \text{bij-betw } (\text{inv-into } \mathcal{V} \pi) (\pi \text{ ' } bl) bl$   
**using** *bij bij-betw-subset source.wellformed bij-betw-inv-into-subset* **by** *fastforce*

**lemma** *iso-bij-betw-individual-blocks-inv-alt*:  
 $bl \in \# \mathcal{B}' \implies \text{bij-betw } (\text{inv-into } \mathcal{V} \pi) bl ((\text{inv-into } \mathcal{V} \pi) \text{ ' } bl)$   
**using** *incidence-system-isomorphism.iso-bij-betw-individual-blocks inverse-incidence-sys-iso*  
**by** *blast*

**lemma** *iso-inv-block-in-alt*:  $(\pi \text{ ' } bl) \in \# \mathcal{B}' \implies bl \subseteq \mathcal{V} \implies bl \in \# \mathcal{B}$   
**using** *image-eqI image-inv-into-cancel inv-iso-block-img iso-points-inv-map*  
**by** (*metis (no-types, lifting) iso-points-map multiset.set-map subset-image-iff*)

**lemma** *iso-img-block-not-in*:  
**assumes**  $x \notin \# \mathcal{B}$   
**assumes**  $x \subseteq \mathcal{V}$   
**shows**  $(\pi \text{ ' } x) \notin \# \mathcal{B}'$   
**proof** (*rule ccontr*)  
**assume**  $a: \neg \pi \text{ ' } x \notin \# \mathcal{B}'$   
**then have**  $a: \pi \text{ ' } x \in \# \mathcal{B}'$  **by** *simp*  
**then have**  $\bigwedge y. y \in (\pi \text{ ' } x) \implies (\text{inv-into } \mathcal{V} \ \pi) \ y \in \mathcal{V}$   
**using** *target.wf-invalid-point iso-points-inv-map* **by** *auto*  
**then have**  $((\text{'}) (\text{inv-into } \mathcal{V} \ \pi)) (\pi \text{ ' } x) \in \# \mathcal{B}$   
**using** *iso-bij-betwn-block-sets-inv* **by** (*meson a bij-betw-apply*)  
**thus** *False*  
**using**  $a$  *assms(1) assms(2) iso-inv-block-in-alt* **by** *blast*  
**qed**

**lemma** *iso-block-multiplicity*:  
**assumes**  $bl \subseteq \mathcal{V}$   
**shows**  $\text{source.multiplicity } bl = \text{target.multiplicity } (\pi \text{ ' } bl)$   
**proof** (*cases bl \in \# \mathcal{B}*)  
**case** *True*  
**have** *inj-on*  $((\text{'}) \ \pi) (\text{set-mset } \mathcal{B})$   
**using** *bij-betw-imp-inj-on iso-bij-betwn-block-sets* **by** *auto*  
**then have**  $\text{count } \mathcal{B} \ bl = \text{count } \mathcal{B}' (\pi \text{ ' } bl)$   
**using** *count-image-mset-le-count-inj-on count-image-mset-ge-count True block-img inv-into-f-f*  
*less-le-not-le order.not-eq-order-implies-strict* **by** *metis*  
**thus** *?thesis* **by** *simp*  
**next**  
**case** *False*  
**have**  $s\text{-mult}: \text{source.multiplicity } bl = 0$   
**by** (*simp add: False count-eq-zero-iff*)  
**then have**  $\text{target.multiplicity } (\pi \text{ ' } bl) = 0$   
**using** *False count-inI iso-inv-block-in-alt*  
**by** (*metis assms*)  
**thus** *?thesis*  
**using**  $s\text{-mult}$  **by** *simp*  
**qed**

**lemma** *iso-point-in-block-img-iff*:  $p \in \mathcal{V} \implies bl \in \# \mathcal{B} \implies p \in bl \longleftrightarrow (\pi \ p) \in (\pi \text{ ' } bl)$   
**by** (*metis bij bij-betw-imp-surj-on iso-bij-betw-individual-blocks-inv bij-betw-inv-into-left imageI*)

**lemma** *iso-point-subset-block-iff*:  $p \subseteq \mathcal{V} \implies bl \in \# \mathcal{B} \implies p \subseteq bl \longleftrightarrow (\pi \text{ ' } p) \subseteq$

( $\pi \text{ ' } bl$ )  
**apply** *auto*  
**using** *image-subset-iff iso-point-in-block-img-iff subset-iff* **by** *metis*

**lemma** *iso-is-image-block*:  $\mathcal{B}' = \text{blocks-image } \mathcal{B} \ \pi$   
**unfolding** *blocks-image-def* **by** (*simp add: block-img iso-points-map*)

**end**

### 10.3 Design Isomorphisms

Apply the concept of isomorphisms to designs only

**locale** *design-isomorphism* = *incidence-system-isomorphism*  $\mathcal{V} \ \mathcal{B} \ \mathcal{V}' \ \mathcal{B}' \ \pi$  + *source*:  
*design*  $\mathcal{V} \ \mathcal{B}$  +  
*target*: *design*  $\mathcal{V}' \ \mathcal{B}'$  **for**  $\mathcal{V}$  **and**  $\mathcal{B}$  **and**  $\mathcal{V}'$  **and**  $\mathcal{B}'$  **and** *bij-map* ( $\langle \pi \rangle$ )

**context** *design-isomorphism*  
**begin**

**lemma** *inverse-design-isomorphism*: *design-isomorphism*  $\mathcal{V}' \ \mathcal{B}' \ \mathcal{V} \ \mathcal{B}$  (*inv-into*  $\mathcal{V} \ \pi$ )  
**using** *inverse-incidence-sys-iso source.wf-design target.wf-design*  
**by** (*simp add: design-isomorphism.intro*)

**end**

#### 10.3.1 Isomorphism Operation

Define the concept of isomorphic designs outside the scope of locale

**definition** *isomorphic-designs* (**infixl**  $\langle \cong_D \rangle$  50) **where**  
 $\mathcal{D} \cong_D \mathcal{D}' \longleftrightarrow (\exists \ \pi . \text{design-isomorphism } (fst \ \mathcal{D}) \ (snd \ \mathcal{D}) \ (fst \ \mathcal{D}') \ (snd \ \mathcal{D}') \ \pi)$

**lemma** *isomorphic-designs-symmetric*:  $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \implies (\mathcal{V}', \mathcal{B}') \cong_D (\mathcal{V}, \mathcal{B})$   
**using** *isomorphic-designs-def design-isomorphism.inverse-design-isomorphism*  
**by** *metis*

**lemma** *isomorphic-designs-implies-bij*:  $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \implies \exists \ \pi . \text{bij-betw } \pi \ \mathcal{V} \ \mathcal{V}'$   
**using** *incidence-system-isomorphism.bij isomorphic-designs-def*  
**by** (*metis design-isomorphism.axioms(1) fst-conv*)

**lemma** *isomorphic-designs-implies-block-map*:  $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \implies \exists \ \pi . \text{image-mset } ((\cdot) \ \pi) \ \mathcal{B} = \mathcal{B}'$   
**using** *incidence-system-isomorphism.block-img isomorphic-designs-def*  
**using** *design-isomorphism.axioms(1)* **by** *fastforce*

**context** *design*  
**begin**

**lemma** *isomorphic-designsI* [intro]:  $\text{design } \mathcal{V}' \mathcal{B}' \implies \text{bij-betw } \pi \mathcal{V} \mathcal{V}' \implies \text{image-mset } ((\cdot) \pi) \mathcal{B} = \mathcal{B}'$   
 $\implies (\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}')$   
**using** *design-isomorphism.intro isomorphic-designs-def wf-design image-set-system-wellformed*  
**by** (*metis bij-betw-imp-surj-on blocks-image-def fst-conv incidence-system-axioms*)

*incidence-system-isomorphism.intro incidence-system-isomorphism-axioms-def snd-conv*)

**lemma** *eq-designs-isomorphic*:

**assumes**  $\mathcal{V} = \mathcal{V}'$

**assumes**  $\mathcal{B} = \mathcal{B}'$

**shows**  $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}')$

**proof** –

**interpret** *d1: design*  $\mathcal{V} \mathcal{B}$  **using** *assms*

**using** *wf-design* **by** *auto*

**interpret** *d2: design*  $\mathcal{V}' \mathcal{B}'$  **using** *assms*

**using** *wf-design* **by** *blast*

**have** *design-isomorphism*  $\mathcal{V} \mathcal{B} \mathcal{V}' \mathcal{B}'$  *id* **using** *assms* **by** (*unfold-locales*) *simp-all*

**thus** *?thesis* **unfolding** *isomorphic-designs-def* **by** *auto*

**qed**

**end**

**context** *design-isomorphism*

**begin**

### 10.3.2 Design Properties/Operations under Isomorphism

**lemma** *design-iso-point-rep-num-eq*:

**assumes**  $p \in \mathcal{V}$

**shows**  $\mathcal{B} \text{ rep } p = \mathcal{B}' \text{ rep } (\pi p)$

**proof** –

**have**  $\{\#b \in \# \mathcal{B} . p \in b\# \} = \{\#b \in \# \mathcal{B} . \pi p \in \pi ' b\# \}$

**using** *assms filter-mset-cong iso-point-in-block-img-iff* *assms* **by** *force*

**then have**  $\{\#b \in \# \mathcal{B}' . \pi p \in b\# \} = \text{image-mset } ((\cdot) \pi) \{\#b \in \# \mathcal{B} . p \in b\# \}$

**by** (*simp add: image-mset-filter-swap block-img*)

**thus** *?thesis*

**by** (*simp add: point-replication-number-def*)

**qed**

**lemma** *design-iso-rep-numbers-eq*: *source.replication-numbers = target.replication-numbers*

**apply** (*simp add: source.replication-numbers-def target.replication-numbers-def*)

**using** *design-iso-point-rep-num-eq design-isomorphism.design-iso-point-rep-num-eq*

*iso-points-map*

**by** (*metis (no-types, lifting) inverse-design-isomorphism iso-points-inv-map rev-image-eqI*)

**lemma** *design-iso-block-size-eq*:  $bl \in \# \mathcal{B} \implies \text{card } bl = \text{card } (\pi ' bl)$

**using** *card-image-le finite-subset-image image-inv-into-cancel*

**by** (*metis iso-points-inv-map iso-points-map le-antisym source.finite-blocks source.wellformed*)

**lemma** *design-iso-block-sizes-eq*: *source.sys-block-sizes = target.sys-block-sizes*  
**apply** (*simp add: source.sys-block-sizes-def target.sys-block-sizes-def*)  
**by** (*metis (no-types, lifting) design-iso-block-size-eq iso-block-in iso-img-block-orig-exists*)

**lemma** *design-iso-points-index-eq*:  
**assumes**  $ps \subseteq \mathcal{V}$   
**shows**  $\mathcal{B} \text{ index } ps = \mathcal{B}' \text{ index } (\pi \text{ ' } ps)$   
**proof** –  
**have**  $\bigwedge b . b \in \# \mathcal{B} \implies ((ps \subseteq b) = ((\pi \text{ ' } ps) \subseteq \pi \text{ ' } b))$   
**using** *iso-point-subset-block-iff* **assms** **by** *blast*  
**then have**  $\{\#b \in \# \mathcal{B} . ps \subseteq b\# \} = \{\#b \in \# \mathcal{B} . (\pi \text{ ' } ps) \subseteq (\pi \text{ ' } b)\#\}$   
**using** *assms filter-mset-cong* **by** *force*  
**then have**  $\{\#b \in \# \mathcal{B}' . \pi \text{ ' } ps \subseteq b\#\} = \text{image-mset } ((\text{'}) \pi) \{\#b \in \# \mathcal{B} . ps \subseteq b\#\}$   
**by** (*simp add: image-mset-filter-swap block-img*)  
**thus** *?thesis*  
**by** (*simp add: points-index-def*)  
**qed**

**lemma** *design-iso-points-indices-imp*:  
**assumes**  $x \in \text{source.point-indices } t$   
**shows**  $x \in \text{target.point-indices } t$   
**proof** –  
**obtain**  $ps$  **where**  $t: \text{card } ps = t$  **and**  $ss: ps \subseteq \mathcal{V}$  **and**  $x: \mathcal{B} \text{ index } ps = x$  **using** *assms*  
**by** (*auto simp add: source.point-indices-def*)  
**then have**  $x\text{-val}: x = \mathcal{B}' \text{ index } (\pi \text{ ' } ps)$  **using** *design-iso-points-index-eq* **by** *auto*  
**have**  $x\text{-img}: (\pi \text{ ' } ps) \subseteq \mathcal{V}'$   
**using** *ss bij iso-points-map* **by** *fastforce*  
**then have**  $\text{card } (\pi \text{ ' } ps) = t$  **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**

**lemma** *design-iso-points-indices-eq*: *source.point-indices t = target.point-indices t*  
**using** *inverse-design-isomorphism design-isomorphism.design-iso-points-indices-imp design-iso-points-indices-imp* **by** *blast*

**lemma** *design-iso-block-intersect-num-eq*:  
**assumes**  $b1 \in \# \mathcal{B}$   
**assumes**  $b2 \in \# \mathcal{B}$   
**shows**  $b1 \text{ |}\cap\text{| } b2 = (\pi \text{ ' } b1) \text{ |}\cap\text{| } (\pi \text{ ' } b2)$   
**proof** –  
**have**  $\text{split}: \pi \text{ ' } (b1 \cap b2) = (\pi \text{ ' } b1) \cap (\pi \text{ ' } b2)$  **using** *assms bij bij-betw-inter-subsets*  
**by** (*metis source.wellformed*)  
**thus** *?thesis* **using** *source.wellformed*  
**by** (*simp add: intersection-number-def iso-points-ss-card split assms(2) inf.coboundedI2*)

qed

**lemma** *design-iso-inter-numbers-imp*:

**assumes**  $x \in \text{source.intersection-numbers}$

**shows**  $x \in \text{target.intersection-numbers}$

**proof** –

**obtain**  $b1\ b2$  **where**  $1: b1 \in\# \mathcal{B}$  **and**  $2: b2 \in\# (\text{remove1-mset } b1\ \mathcal{B})$  **and**  $xval:$   
 $x = b1 \mid\cap\mid b2$

**using** *assms* **by** (*auto simp add: source.intersection-numbers-def*)

**then have**  $pi1: \pi \text{ ' } b1 \in\# \mathcal{B}'$  **by** (*simp add: iso-block-in*)

**have**  $pi2: \pi \text{ ' } b2 \in\# (\text{remove1-mset } (\pi \text{ ' } b1)\ \mathcal{B}')$  **using** *iso-block-in 2*

**by** (*metis (no-types, lifting) 1 block-ing image-mset-remove1-mset-if in-remove1-mset-neq*)

*iso-blocks-map-inj more-than-one-mset-mset-diff multiset.set-map*)

**have**  $x = (\pi \text{ ' } b1) \mid\cap\mid (\pi \text{ ' } b2)$  **using** *1 2 design-iso-block-intersect-num-eq*

**by** (*metis in-diffD xval*)

**then have**  $x \in \{b1 \mid\cap\mid b2 \mid b1\ b2 . b1 \in\# \mathcal{B}' \wedge b2 \in\# (\mathcal{B}' - \{\#b1\#})\}$

**using**  $pi1\ pi2$  **by** *blast*

**then show** *?thesis* **by** (*simp add: target.intersection-numbers-def*)

qed

**lemma** *design-iso-intersection-numbers: source.intersection-numbers = target.intersection-numbers*

**using** *inverse-design-isomorphism design-isomorphism.design-iso-inter-numbers-imp*

*design-iso-inter-numbers-imp* **by** *blast*

**lemma** *design-iso-n-intersect-num*:

**assumes**  $b1 \in\# \mathcal{B}$

**assumes**  $b2 \in\# \mathcal{B}$

**shows**  $b1 \mid\cap\mid_n b2 = ((\pi \text{ ' } b1) \mid\cap\mid_n (\pi \text{ ' } b2))$

**proof** –

**let**  $?A = \{x . x \subseteq b1 \wedge x \subseteq b2 \wedge \text{card } x = n\}$

**let**  $?B = \{y . y \subseteq (\pi \text{ ' } b1) \wedge y \subseteq (\pi \text{ ' } b2) \wedge \text{card } y = n\}$

**have**  $b1v: b1 \subseteq \mathcal{V}$  **by** (*simp add: assms(1) source.wellformed*)

**have**  $b2v: b2 \subseteq \mathcal{V}$  **by** (*simp add: assms(2) source.wellformed*)

**then have**  $\bigwedge x\ y . x \subseteq b1 \implies x \subseteq b2 \implies y \subseteq b1 \implies y \subseteq b2 \implies \pi \text{ ' } x = \pi \text{ ' } y \implies x = y$

**using**  $b1v\ b2v$  **bij** **by** (*metis bij-betw-imp-surj-on bij-betw-inv-into-subset dual-order.trans*)

**then have** *inj*: *inj-on*  $((\text{'}) \pi)$   $?A$  **by** (*simp add: inj-on-def*)

**have** *eqcard*:  $\bigwedge xa . xa \subseteq b1 \implies xa \subseteq b2 \implies \text{card } (\pi \text{ ' } xa) = \text{card } xa$  **using**  $b1v$   
 $b2v$  *bij*

**using** *iso-points-ss-card* **by** *auto*

**have** *surj*:  $\bigwedge x . x \subseteq \pi \text{ ' } b1 \implies x \subseteq \pi \text{ ' } b2 \implies$

$x \in \{(\pi \text{ ' } xa) \mid xa . xa \subseteq b1 \wedge xa \subseteq b2 \wedge \text{card } xa = \text{card } x\}$

**proof** –

**fix**  $x$

**assume**  $x1: x \subseteq \pi \text{ ' } b1$  **and**  $x2: x \subseteq \pi \text{ ' } b2$

**then obtain**  $xa$  **where** *eq-x*:  $\pi \text{ ' } xa = x$  **and** *ss*:  $xa \subseteq \mathcal{V}$

**by** (*metis b1v dual-order.trans subset-imageE*)

**then have**  $f1: xa \subseteq b1$  **by** (*simp add: x1 assms(1) iso-point-subset-block-iff*)  
**then have**  $f2: xa \subseteq b2$  **by** (*simp add: eq-x ss assms(2) iso-point-subset-block-iff*  
 $x2$ )  
**then have**  $f3: \text{card } xa = \text{card } x$  **using** *bij* **by** (*simp add: eq-x ss iso-points-ss-card*)  
**then show**  $x \in \{(\pi \text{ ' } xa) \mid xa . xa \subseteq b1 \wedge xa \subseteq b2 \wedge \text{card } xa = \text{card } x\}$   
**using**  $f1 f2 f3 \langle \pi \text{ ' } xa = x \rangle$  **by** *auto*  
**qed**  
**have** *bij-betw* ( $(\text{ ' } \pi) ?A ?B$ )  
**proof** (*auto simp add: bij-betw-def*)  
**show** *inj-on* ( $(\text{ ' } \pi) \{x. x \subseteq b1 \wedge x \subseteq b2 \wedge \text{card } x = n\}$ ) **using** *inj* **by** *simp*  
**show**  $\bigwedge xa. xa \subseteq b1 \implies xa \subseteq b2 \implies n = \text{card } xa \implies \text{card } (\pi \text{ ' } xa) = \text{card } xa$   
**using** *eqcard* **by** *simp*  
**show**  $\bigwedge x. x \subseteq \pi \text{ ' } b1 \implies x \subseteq \pi \text{ ' } b2 \implies n = \text{card } x \implies$   
 $x \in (\text{ ' } \pi) \text{ ' } \{xa. xa \subseteq b1 \wedge xa \subseteq b2 \wedge \text{card } xa = \text{card } x\}$   
**using** *surj* **by** (*simp add: setcompr-eq-image*)  
**qed**  
**thus** *?thesis*  
**using** *bij-betw-same-card* **by** (*auto simp add: n-intersect-number-def*)  
**qed**

**lemma** *subdesign-iso-implies:*

**assumes** *sub-set-system*  $V B \mathcal{V} \mathcal{B}$   
**shows** *sub-set-system*  $(\pi \text{ ' } V) (\text{blocks-image } B \pi) \mathcal{V}' \mathcal{B}'$   
**proof** (*unfold-locales*)  
**show**  $\pi \text{ ' } V \subseteq \mathcal{V}'$   
**by** (*metis assms image-mono iso-points-map sub-set-system.points-subset*)  
**show**  $\text{blocks-image } B \pi \subseteq \# \mathcal{B}'$   
**by** (*metis assms block-img blocks-image-def image-mset-subseteq-mono sub-set-system.blocks-subset*)

**qed**

**lemma** *subdesign-image-is-design:*

**assumes** *sub-set-system*  $V B \mathcal{V} \mathcal{B}$   
**assumes** *design*  $V B$   
**shows** *design*  $(\pi \text{ ' } V) (\text{blocks-image } B \pi)$   
**proof** –  
**interpret** *fin*: *finite-incidence-system*  $(\pi \text{ ' } V) (\text{blocks-image } B \pi)$  **using** *assms(2)*  
**by** (*simp add: design.axioms(1) finite-incidence-system.image-set-system-finite*)  
**interpret** *des*: *sub-design*  $V B \mathcal{V} \mathcal{B}$  **using** *assms design.wf-design-iff*  
**by** (*unfold-locales, auto simp add: sub-set-system.points-subset sub-set-system.blocks-subset*)  
**have** *bl-img*:  $\text{blocks-image } B \pi \subseteq \# \mathcal{B}'$   
**by** (*simp add: blocks-image-def des.blocks-subset image-mset-subseteq-mono iso-is-image-block*)  
**then show** *?thesis*  
**proof** (*unfold-locales, auto*)  
**show**  $\{\} \in \# \text{blocks-image } B \pi \implies \text{False}$   
**using** *assms subdesign-iso-implies target.blocks-nempty bl-img* **by** *auto*  
**qed**  
**qed**

```

lemma sub-design-isomorphism:
  assumes sub-set-system  $V B \mathcal{V} \mathcal{B}$ 
  assumes design  $V B$ 
  shows design-isomorphism  $V B (\pi \text{ ' } V)$  (blocks-image  $B \pi$ )  $\pi$ 
proof –
  interpret design  $(\pi \text{ ' } V)$  (blocks-image  $B \pi$ )
    by (simp add: assms(1) assms(2) subdesign-image-is-design)
  interpret des: design  $V B$  by fact
  show ?thesis
  proof (unfold-locales)
    show bij-betw  $\pi V (\pi \text{ ' } V)$  using bij
      by (metis assms(1) bij-betw-subset sub-set-system.points-subset)
    show image-mset  $((\text{'}) \pi) B = \text{blocks-image } B \pi$  by (simp add: blocks-image-def)
  qed
qed

end
end

```

**theory** *Design-Theory-Root*

**imports**

*Multisets-Extras*

*Design-Basics*

*Design-Operations*

*Block-Designs*

*BIBD*

*Resolvable-Designs*

*Group-Divisible-Designs*

*Designs-And-Graphs*

*Design-Isomorphisms*

*Sub-Designs*

**begin**

**end**

## References

- [1] E. A. Bender. Partitions of multisets. *Discrete Mathematics*, 9(4):301–311, Oct. 1974.
- [2] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*. Number v. 45 in North-Holland Mathematical Library. North Holland : Distributors for the U.S.A. and Canada, Elsevier Science Pub. Co, Amsterdam ; New York, 1989.

- [3] C. J. Colbourn and J. H. Dinitz. *Handbook of Combinatorial Designs / Edited by Charles J. Colbourn, Jeffrey H. Dinitz*. Chapman & Hall/CRC, 2nd edition, 2007.
- [4] S. Herke. Math3301 lecture notes in combinatorial design theory, July 2016.
- [5] L. H. Soicher. Designs, Groups and Computing. In A. Detinko, D. Flannery, and E. O'Brien, editors, *Probabilistic Group Theory, Combinatorics, and Computing*, Lecture Notes in Mathematics 2070, pages 83–107. Springer, 2013.
- [6] D. Stinson. *Combinatorial Designs: Constructions and Analysis*. Springer, 2004.