Design Theory

Chelsea Edmonds and Lawrence Paulson

March 17, 2025

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

1 Micellanious 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 *card* $\{x \in Pow\ A\ .\ card\ x = 1\}$ = *card* (A)
using *assms*
proof (*induct rule*: *finite-induct*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x F*)
have *Pow* (*insert x F*) = *Pow F* ∪ *insert x ' Pow F*
by (*simp add*: *Pow-insert*)
then have *split*: $\{y \in Pow\ (insert\ x\ F)\ .\ card\ y = 1\}$ =
$\{y \in (Pow\ F)\ .\ card\ y = 1\} \cup \{y \in (insert\ x\ `\ Pow\ F)\ .\ card\ y = 1\}$
by *blast*
have $\bigwedge\ y\ .\ y \in (insert\ x\ `\ Pow\ F) \Longrightarrow$ *finite y*
using *finite-subset insert.hyps*(*1*) **by** *fastforce*
then have *single*: $\bigwedge\ y\ .\ y \in (insert\ x\ `\ Pow\ F) \Longrightarrow card\ y = 1 \Longrightarrow y = \{x\}$
by (*metis card-1-singletonE empty-iff image-iff insertCI insertE*)
then have *card* $\{y \in (insert\ x\ `\ Pow\ F)\ .\ card\ y = 1\} = 1$
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 $\{y \in (insert\ x\ `\ Pow\ F)\ .\ card\ y = 1\} = \{\{x\}\}$
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*:$\{y \in Pow\ (insert\ x\ F)\ .\ card\ y = 1\} = \{y \in (Pow\ F)\ .\ card\ y = 1\} \cup \{\{x\}\}$
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 $\{y \in (Pow\ F)\ .\ card\ y = 1\} \cap \{\{x\}\} = \{\}$ **by** *blast*
then have *fact*:*card* $\{y \in Pow\ (insert\ x\ F)\ .\ card\ y = 1\}$ =
card $\{y \in (Pow\ F)\ .\ card\ y = 1\}$ + *card* $\{\{x\}\}$
using *split2 card-Un-disjoint insert.hyps*(*1*) **by** *auto*
have *card* (*insert x F*) = *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 *card A* > *0*
obtains *x* **where** *x* ∈ *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*: *finite* $A \Longrightarrow$ *card* $ps >$ *card* $A \Longrightarrow \neg\ (ps \subseteq A)$
using *card-mono leD* **by** *auto*

lemma *card-inter-lt-single*: *finite* $A \Longrightarrow$ *finite* $B \Longrightarrow$ *card* $(A \cap B) \leq$ *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*: $\bigwedge x\ .\ x \in A \Longrightarrow x \in C$ **by** *blast*
thus *False* **using** *a assms* **by** *blast*
qed

lemma *set-card-diff-ge-zero*: *finite* $A \Longrightarrow$ *finite* $B \Longrightarrow A \neq B \Longrightarrow$ *card* $A =$ *card* $B \Longrightarrow$
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*: *card* $(\{a \in A\ .\ P\ a\ \} - \{x\})$ = *card* $\{a \in (A - \{x\})\ .\ (P\ a\ )\}$
by (*simp add*: *set-filter-diff*)

lemma *obtain-subset-with-card-int-n*:
assumes $(n ::int) \leq$ *of-nat* (*card* S)
assumes $(n ::int) \geq 0$
obtains T **where** $T \subseteq S$ *of-nat* (*card* T) $= (n ::int)$ *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 $\bigwedge g\ .\ g \in G \Longrightarrow g \neq \{\}$
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 $\bigwedge g\ .\ g \in G \Longrightarrow g \neq \{x\} \longleftrightarrow$ *?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*: *bij-betw f A B* $\Longrightarrow$ *a1* $\subseteq$ *A* $\Longrightarrow$ *a2* $\subseteq$ *A*
$\Longrightarrow$ *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*} | *g*. *g* ∈ *G* ∧ *g* ≠ {*x*}}
proof (*intro partition-onI*)
show ⋀*p*. *p* ∈ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}} $\Longrightarrow$ *p* ≠ {}
using *assms partition-onD3 subset-singletonD* **by** *force*
let *?f* = (λ *g* . *g* − {*x*})
have *un-img*: ⋃({ *?f g* | *g*. *g* ∈ *G* }) = *?f* (⋃ *G*) **by** *blast*
have *empty*: ⋃ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}} = ⋃({*g* − {*x*} | *g*. *g* ∈ *G* } − {{}})
by *blast*
then have ⋃({*g* − {*x*} | *g*. *g* ∈ *G* } − {{}}) = ⋃({*g* − {*x*} | *g*. *g* ∈ *G* }) **by** *blast*
then show ⋃ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}} = *A* − {*x*} **using** *partition-onD1 assms un-img*
by (*metis empty*)
then show ⋀*p p′*.
p ∈ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}} $\Longrightarrow$
p′ ∈ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}} $\Longrightarrow$ *p* ≠ *p′* $\Longrightarrow$ *p* ∩ *p′* = {}
proof −
fix *p1 p2*
assume *p1*: *p1* ∈ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}}
and *p2*: *p2* ∈ {*g* − {*x*} |*g*. *g* ∈ *G* ∧ *g* ≠ {*x*}}
and *ne*: *p1* ≠ *p2*
obtain *p1′ p2′* **where** *orig1*: *p1* = *p1′* − {*x*} **and** *orig2*: *p2* = *p2′* − {*x*}
and *origne*: *p1′* ≠ *p2′* **and** *ne1*: *p1′* ≠ {*x*} **and** *ne2*:*p2′* ≠ {*x*} **and** *ing1*: *p1′* ∈ *G*
and *ing2*: *p2′* ∈ *G*
using *p1 p2* **using** *mem-Collect-eq ne* **by** *blast*
then have *p1′* ∩ *p2′* = {} **using** *assms partition-onD2 ing1 ing2 origne disjointD* **by** *blast*
thus *p1* ∩ *p2* = {} **using** *orig1 orig2* **by** *blast*
qed
qed

lemma *partition-on-cart-prod*:
assumes *card I* > *0*
assumes *A* ≠ {}
assumes *G* ≠ {}
assumes *partition-on A G*
shows *partition-on* (*A* × *I*) {*g* × *I* |*g*. *g* ∈ *G*}
proof (*intro partition-onI*)
show ⋀*p*. *p* ∈ {*g* × *I* |*g*. *g* ∈ *G*} $\Longrightarrow$ *p* ≠ {}

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 $\bigwedge p\ p'.\ p \in \{g \times I \mid g.\ g \in G\} \Longrightarrow p' \in \{g \times I \mid g.\ g \in G\} \Longrightarrow p \neq p' \Longrightarrow p \cap p' = \{\}$
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*: $A \neq \{\#\} \longleftrightarrow (set\text{-}mset\ A) \neq \{\}$
by *simp*

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

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

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

lemma *size-multiset-set-mset-const-count*:
assumes $card\ (set\text{-}mset\ A) = ca$
assumes $\bigwedge p.\ p \in\#\ A \Longrightarrow count\ A\ p = ca2$
shows $size\ A = (ca * ca2)$
proof −
have $size\ A = (\sum\ p \in (set\text{-}mset\ A)\ .\ count\ A\ p)$ **using** *size-multiset-overloaded-eq* **by** *auto*
then have $size\ A = (\sum\ p \in (set\text{-}mset\ A)\ .\ ca2)$ **using** *assms* **by** *simp*
thus *?thesis* **using** *assms(1)* **by** *auto*
qed

lemma *size-multiset-int-count*:
assumes $of\text{-}nat\ (card\ (set\text{-}mset\ A)) = (ca :: int)$
assumes $\bigwedge p.\ p \in\#\ A \Longrightarrow of\text{-}nat\ (count\ A\ p) = (ca2 :: int)$
shows $of\text{-}nat\ (size\ A) = ((ca :: int) * ca2)$
proof −
have $size\ A = (\sum\ p \in (set\text{-}mset\ A)\ .\ count\ A\ p)$ **using** *size-multiset-overloaded-eq* **by** *auto*
then have $of\text{-}nat\ (size\ A) = (\sum\ p \in (set\text{-}mset\ A)\ .\ ca2)$ **using** *assms* **by** *simp*
thus *?thesis* **using** *assms(1)* **by** *auto*
qed

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

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

Lemmas for repeat_mset

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

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

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

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

lemma *elem-in-original-in-repeat*: $n > 0 \Longrightarrow a \in\# A \Longrightarrow a \in\#$ *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*: *size* $\{\#\ a \in\# (A1 + A2)\ .\ P\ a\ \#\} =$ *size* $\{\#\ a \in\# A1\ .\ P\ a\ \#\} +$
size $\{\#\ a \in\# A2\ .\ P\ a\ \#\}$
by *simp*

lemma *size-filter-neg*: *size* $\{\#a \in\# A\ .\ P\ a\ \#\} =$ *size* $A -$ *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 $\bigwedge a\ .\ P\ a \Longrightarrow Q\ a$
shows *filter-mset P A* = *filter-mset P* (*filter-mset Q A*)
proof −
have *filter-mset P* (*filter-mset Q A*) = *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-mset* (λ *a* . $\{x, y\} \subseteq a$) *A* =
filter-mset (λ *a* . $\{x, y\} \subseteq a$) (*filter-mset* (λ *a* . $x \in a$) *A*)
proof −
have *filter*: *filter-mset* (λ *a* . $\{x, y\} \subseteq a$) (*filter-mset* (λ *a* . $x \in a$) *A*) =
filter-mset (λ *a* . $x \in a \wedge \{x, y\} \subseteq a$) *A* **by** (*simp add*: *filter-filter-mset*)
have $\bigwedge$ *a*. $\{x, y\} \subseteq a \Longrightarrow x \in a$ **by** *simp*
thus *?thesis* **using** *filter* **by** *auto*
qed

lemma *multiset-image-do-nothing*: ($\bigwedge$ *x* .$x \in\# A \Longrightarrow f\ x = x$) $\Longrightarrow$ *image-mset f A* = *A*
by (*induct A*) *auto*

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

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

lemma *filter-mset-image-mset*:
filter-mset P (*image-mset f A*) = *image-mset f* (*filter-mset* (λ*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-mset* (λ *x* . *f x*) (*image-mset* (λ *y* . *g y*) *A*) =
image-mset (λ *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 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*]: *size* (*A* ×# {#*b*#}) = *size A*
by (*simp add*: *Times-mset-single-right*)

lemma *size-cartesian-product-empty* [*simp*]: *size* (*A* ×# {#}) = *0*
by *simp*

lemma *size-add-elem-step-eq*:
assumes *size* (*A* ×# *B*) = *size A* ∗ *size B*
shows *size* (*add-mset x A* ×# *B*) = *size* (*add-mset x A*) ∗ *size B*
proof −
have (*add-mset x A* ×# *B*) = *A* ×# *B* + {#*x*#} ×# *B*
by (*metis Sigma-mset-plus-distrib1 add-mset-add-single*)
then have *size* (*add-mset x A* ×# *B*) = *size* (*A* ×# *B*) + *size B* **by** *auto*
also have ... = *size A* ∗ *size B* + *size B*
by (*simp add*: *assms*)
finally have *size* (*add-mset x A* ×# *B*) = (*size A* + *1*) ∗ *size B*
by *auto*
thus *?thesis* **by** *simp*
qed

lemma *size-cartesian-product*: *size* (*A* ×# *B*) = *size A* ∗ *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* ×# *B*)
unfolding *distinct-mset-count-less-1*
proof (*rule allI*)
fix *x*
have *count-mult*: *count* (*A* ×# *B*) *x* = *count A* (*fst x*) ∗ *count B* (*snd x*)
using *count-Sigma-mset* **by** (*metis prod.exhaust-sel*)
then have *count A* (*fst x*) ∗ *count B* (*snd x*) ≤ *1* **using** *assm1 assm2*
unfolding *distinct-mset-count-less-1* **using** *mult-le-one* **by** *blast*
thus *count* (*A* ×# *B*) *x* ≤ *1* **using** *count-mult* **by** *simp*
qed

lemma *cart-product-single-intersect*: *x1* ≠ *x2* ⟹ ({#*x1*#} ×# *A*) ∩# ({#*x2*#} ×# *B*) = {#}
using *multiset-inter-single* **by** *fastforce*

lemma *size-union-distinct-cart-prod*: *x1* ≠ *x2* ⟹ *size* (({#*x1*#} ×# *A*) ∪# ({#*x2*#} ×# *B*)) =
size ({#*x1*#} ×# *A*) + *size* ({#*x2*#} ×# *B*)
by (*simp add*: *cart-product-single-intersect size-Un-disjoint*)

lemma *size-Union-distinct-cart-prod*: *distinct-mset M* ⟹

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* $M \Longrightarrow$
($\bigwedge$ p . $p \in\#$ $M \Longrightarrow$ *size* ($\{\#$ $b \in\#$ B . P p b $\#\}$) = c) $\Longrightarrow$
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* $V \Longrightarrow$
($\bigwedge$ b . $b \in\#$ $B \Longrightarrow$ *size* ($\{\#$ $v \in\#$ V . P v b $\#\}$) = c) $\Longrightarrow$
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 $\times\#$ $\{\#b\#\}$) $\Longrightarrow$ 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-mset* x A $\times\#$ $\{\#b\#\}$ = *add-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-mset Q $\{\#x\#\}$ $\times\#$ $\{\#b\#\}$
by (*metis Sigma-mset-plus-distrib1 add-mset-add-single filter-union-mset*)
have *filter-mset* P $\{\#(x, b)\#\}$ = *filter-mset* Q $\{\#x\#\}$ $\times\#$ $\{\#b\#\}$
using *add.prems* **by** *fastforce*
then show *?case* **using** *lhs rhs add.IH add.prems* **by** *force*

qed

lemma *cart-prod-singleton-left-gen*:
assumes $\bigwedge x\ .\ x \in\# (\{\#a\#\} \times\# B) \Longrightarrow P\ x \longleftrightarrow Q\ (snd\ x)$
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*: *filter-mset P* ($\{\#a\#\} \times\#$ *add-mset x B*) = *filter-mset P* ($\{\#a\#\} \times\#$ *B*) +
filter-mset P $\{\#(a, x)\#\}$
by (*simp add*: *cart-product-add-1-filter2*)
have *rhs*: $\{\#a\#\} \times\#$ *filter-mset Q* (*add-mset x B*) = $\{\#a\#\} \times\#$ *filter-mset Q* *B* +
$\{\#a\#\} \times\#$ *filter-mset Q* $\{\#x\#\}$
using *add-mset-add-single filter-union-mset* **by** (*metis Times-mset-single-left image-mset-union*)
have *filter-mset P* $\{\#(a, x)\#\} = \{\#a\#\} \times\#$ *filter-mset Q* $\{\#x\#\}$
using *add.prems* **by** *fastforce*
then show *?case* **using** *lhs rhs add.IH add.prems* **by** *force*
qed

lemma *cart-product-singleton-left*: $\{\#m \in\# (\{\#a\#\} \times\#\ N)\ .\ fst\ m \in snd\ m\ \#\}$ =
($\{\#a\#\} \times\# \{\#\ n \in\# N\ .\ a \in n\ \#\}$) (**is** *?A = ?B*)
proof −
have *stmt*: $\bigwedge m.\ m \in\# (\{\#a\#\} \times\# N) \Longrightarrow fst\ m \in snd\ m \longleftrightarrow a \in snd\ m$
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*: $\{\#m \in\# (N \times\# \{\#b\#\})\ .\ fst\ m \in snd\ m\ \#\} =$
($\{\#\ n \in\# N\ .\ n \in b\ \#\} \times\# \{\#\ b\ \#\}$) (**is** *?A = ?B*)
proof −
have *stmt*: $\bigwedge m.\ m \in\# (N \times\# \{\#b\#\}) \Longrightarrow fst\ m \in snd\ m \longleftrightarrow fst\ m \in b$
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*: $\{\#m \in\# ((add\text{-}mset\ a\ M) \times\# N)\ .\ (fst\ m \in snd\ m)\ \#\} =$
$\{\#m \in\# (M \times\# N)\ .\ (fst\ m \in snd\ m)\ \#\} + (\{\#a\#\} \times\# \{\#\ n \in\# N\ .\ a \in n\ \#\})$
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*: $\{\#m \in\# M \times\# (\textit{add-mset } b\ N)\ .\ (\textit{fst}\ m \in \textit{snd}\ m)\ \#\} =$
$\{\#m \in\# (M \times\# N)\ .\ (\textit{fst}\ m \in \textit{snd}\ m)\ \#\} + (\{\#\ n \in\# M\ .\ n \in b\ \#\} \times\# \{\#b\#\})$
unfolding *add-mset-add-single* [*of b N*] *Sigma-mset-plus-distrib1*
by (*metis* (*no-types*) *add-mset-add-single cart-product-add-1-filter2 cart-product-singleton-right*)

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

lemma *set-break-down-right*:
shows $\{\#\ x \in\# M \times\# N\ .\ (\textit{fst}\ x) \in (\textit{snd}\ x)\ \ \#\} = (\sum n \in\# N.\ (\{\#m \in\# M.\ m \in n\#\} \times\# \{\#n\#\}))$
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 $\bigwedge x\ .\ x \in\# A \Longrightarrow f\ x = g\ x$
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 $\bigwedge x\ .\ x \in\# A \Longrightarrow f\ x \geq g\ x$
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*:$(\bigwedge\ x\ .\ f\ x \geq 0) \Longrightarrow (\sum\ x \in\# A.\ f\ x\ ) \geq 0$

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.prems*)
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 (⋀ *x* . *f x* ≥ *0*)
shows (∑ *x* ∈# *A*. *f x*) = *0* ⟷ (∀ *x* ∈ *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 $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$(y * (count\ (add\text{-}mset\ y\ A)\ y)) + (\sum x{\in}(set\text{-}mset\ A) - \{y\}.\ x * count\ A\ x)$
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*: $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$y + y * (count\ A\ y) + (\sum x{\in}(set\text{-}mset\ A) - \{y\}.\ x * count\ A\ x)$
using *y-count* **by** *simp*
then have $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$y + (\sum x{\in}insert\ y\ ((set\text{-}mset\ A) - \{y\})\ .\ x * count\ A\ x)$
by (*simp add*: *sum.insert-remove*)
then have $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$y + (\sum x{\in}(set\text{-}mset\ A)\ .\ x * count\ A\ x)$
by (*simp add*: *True insert-absorb*)
then show *?thesis* **using** *lhs add.IH*
by *linarith*
next
case *False*
have *x-val*: $\bigwedge x\ .\ x \in set\text{-}mset\ A \Longrightarrow x{*}\ count\ (add\text{-}mset\ y\ A)\ x = x * (count\ A\ x)$
using *False* **by** *auto*
have *y-count*: $count\ (add\text{-}mset\ y\ A)\ y = 1$ **using** *False count-inI* **by** *fastforce*
have *lhs*: $(\sum x{\in}\#add\text{-}mset\ y\ A.\ x) = (\sum x{\in}\#\ A.\ x) + y$ **by** *simp*
have $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$(y * count\ (add\text{-}mset\ y\ A)\ y) + (\sum x{\in}set\text{-}mset\ A.\ x * count\ A\ x)$
using *x-val rhs* **by** (*metis* (*no-types*, *lifting*) *False finite-set-mset sum.cong sum.insert*)
then have $(\sum x{\in}set\text{-}mset\ (add\text{-}mset\ y\ A).\ x * count\ (add\text{-}mset\ y\ A)\ x) =$
$y + (\sum x{\in}set\text{-}mset\ A.\ x * count\ A\ x)$
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*: $size\ \{\#\ a \in\#\ A\ .\ P\ a \vee Q\ a\ \#\} =$
$size\ \{\#\ a \in\#\ A\ .\ P\ a\ \#\} + size\ \{\#\ a \in\#\ A\ .\ Q\ a\ \#\} - size\ \{\#\ a \in\#\ A\ .\ P\ a \wedge Q\ a\ \#\}$
by (*simp add*: *mset-bunion-filter mset-inter-filter mset-union-size-inter*)

definition *partition-on-mset* :: *'a multiset* ⇒ *'a multiset multiset* ⇒ *bool* **where**

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

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

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

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

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

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

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

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

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

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

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

lemma *partition-on-mset-card*: **assumes** *partition-on-mset A P* **shows** *size P* $\leq$ *size A*
proof (*rule ccontr*)
assume $\neg$ *size P* $\leq$ *size A*
then have *a*: *size P* $>$ *size A* **by** *simp*
have $\bigwedge$ *x* . *x* $\in\#$ *P* $\Longrightarrow$ *size x* $>$ *0* **using** *partition-on-msetD2*
using *assms nonempty-has-size* **by** *auto*
then have ($\sum x \in\#$ *P*. *size x*) $\geq$ *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* $\Longrightarrow$ *a* $\in\#$ *A* $\Longrightarrow$
(*sum x* $\in\#$ *P*. *count x a*) = *count A a*
by (*metis count-sum-mset partition-on-msetD1*)

lemma *partition-on-mset-subsets*: *partition-on-mset A P* $\Longrightarrow$ *x* $\in\#$ *P* $\Longrightarrow$ *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* $\Longrightarrow$ (*sum p* $\in\#$ *P*. *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* $\Longrightarrow$ *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* $\Longrightarrow$ *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 $x \in\# \sum_\# (\textit{mset-set}\ (\textit{mset-set}\ `\ P))$
shows $\textit{count}\ (\sum_\#\ (\textit{mset-set}\ (\textit{mset-set}\ `\ P)))\ x = 1$
proof −
have *disj*: *disjoint P* **using** *assms partition-onD2* **by** *auto*
then obtain *p* **where** *pin*: $p \in\# \textit{mset-set}\ (\textit{mset-set}\ `\ P)$ **and** *xin*: $x \in\# p$
using *assms* **by** *blast*
then have $\textit{count}\ (\textit{mset-set}\ (\textit{mset-set}\ `\ P))\ p = 1$
by (*meson count-eq-zero-iff count-mset-set′*)
then have *filter*: $\bigwedge p'.\ p' \in\# ((\textit{mset-set}\ (\textit{mset-set}`\ P)) - \{\#p\#\}) \Longrightarrow p \neq p'$
using *count-eq-zero-iff count-single* **by** *fastforce*
have *zero*: $\bigwedge p'.\ p' \in\# \textit{mset-set}\ (\textit{mset-set}\ `\ P) \Longrightarrow p' \neq p \Longrightarrow \textit{count}\ p'\ x = 0$
proof (*rule ccontr*)
fix p'
assume *assm*: $p' \in\# \textit{mset-set}\ (\textit{mset-set}\ `\ P)$ **and** *ne*: $p' \neq p$ **and** *n0*: $\textit{count}\ p'\ x \neq 0$
then have *xin2*: $x \in\# p'$ **by** *auto*
obtain *p1 p2* **where** *p1in*: $p1 \in P$ **and** *p2in*: $p2 \in P$ **and** *p1eq*: $\textit{mset-set}\ p1 = p$
and *p2eq*: $\textit{mset-set}\ p2 = p'$ **using** *assm assms*(1) *assms*(2) *pin*
by (*metis* (*no-types*, *lifting*) *elem-mset-set finite-elements finite-imageI image-iff*)
have *origne*: $p1 \neq p2$ **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*: $\textit{count}\ p\ x = 1$ **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 $\textit{count}\ (\sum_\#\ (\textit{mset-set}\ (\textit{mset-set}\ `\ P)))\ x = (\sum p' \in\# (\textit{mset-set}\ (\textit{mset-set}\ `\ P)).\ \textit{count}\ p'\ x)$
using *count-sum-mset* **by** *auto*
also have $\ldots = (\textit{count}\ p\ x) + (\sum p' \in\# ((\textit{mset-set}\ (\textit{mset-set}\ `\ P)) - \{\#p\#\}).\ \textit{count}\ p'\ x)$
by (*metis* (*mono-tags*, *lifting*) *insert-DiffM pin sum-mset.insert*)
also have $\ldots = 1 + (\sum p' \in\# ((\textit{mset-set}\ (\textit{mset-set}\ `\ P)) - \{\#p\#\}).\ \textit{count}\ p'\ x)$
using *one* **by** *presburger*
finally have $\textit{count}\ (\sum_\#\ (\textit{mset-set}\ (\textit{mset-set}\ `\ P)))\ x = 1 + (\sum p' \in\# ((\textit{mset-set}\ (\textit{mset-set}\ `\ P)) - \{\#p\#\}).\ 0)$
using *zero filter* **by** (*metis* (*mono-tags*, *lifting*) *in-diffD sum-over-fun-eq*)
then show $\textit{count}\ (\sum_\#\ (\textit{mset-set}\ (\textit{mset-set}\ `\ P)))\ x = 1$ **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*: $\bigcup P = A$ **using** *assms partition-onD1* **by** *auto*
have *imp*: $\bigwedge x.\ x \in\# \sum_{\#}$ (*mset-set* (*mset-set* ‘ *P*)) $\Longrightarrow$ $x \in\#$ *mset-set A*
proof −
fix *x*
assume $x \in\# \sum_{\#}$ (*mset-set* (*mset-set* ‘ *P*))
then obtain *p* **where** $p \in$ (*mset-set* ‘ *P*) **and** *xin*: $x \in\# p$
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 $x \in\#$ *mset-set A*
using *partd1 xin assms*(*1*) **by** *auto*
qed
have *imp2*: $\bigwedge x\ .\ x \in\#$ *mset-set A* $\Longrightarrow$ $x \in\# \sum_{\#}$ (*mset-set* (*mset-set* ‘ *P*))
proof −
fix *x*
assume $x \in\#$ *mset-set A*
then have $x \in A$ **by** (*simp add*: *assms*(*1*))
then obtain *p* **where** $p \in P$ **and** $x \in p$ **using** *assms*(*2*) **using** *partd1* **by** *blast*
then obtain *p′* **where** $p' \in$ (*mset-set* ‘ *P*) **and** $p' =$ *mset-set p* **by** *blast*
thus $x \in\# \sum_{\#}$ (*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*: $\bigwedge\ x\ .\ x \in\#$ *mset-set A* $\Longrightarrow$ *count* (*mset-set A*) *x* = *1*
using *assms*(*1*) **by** *fastforce*
then show $\sum_{\#}$ (*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 $\bigwedge$ *p*. $p \in P \Longrightarrow\ p \neq \{\}$ **using** *assms partition-onD3* **by** *auto*
then have $\bigwedge$ *p*. $p \in P \Longrightarrow$ *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 $p \in\# \mathit{image\text{-}mset\ set\text{-}mset\ P}$
assumes $p' \in\# \mathit{image\text{-}mset\ set\text{-}mset\ P}$
assumes $p \neq p'$
shows $p \cap p' = \{\}$
proof −
obtain *p1* **where** *p1in*: $p1 \in\# P$ **and** *p1eq*: *set-mset p1* $= p$ **using** *assms*(*3*)
by *blast*
obtain *p2* **where** *p2in*: $p2 \in\# P$ **and** *p2eq*: *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 $\bigwedge p.\ p \in\# \mathit{image\text{-}mset\ set\text{-}mset\ P} \Longrightarrow p \neq \{\}$
using *assms*(*1*) *partition-on-msetD2* **by** *fastforce*
next
have $\bigwedge\ x\ .\ x \in \mathit{set\text{-}mset\ A} \Longrightarrow x \in \bigcup\ (\mathit{set\text{-}mset}\ (\mathit{image\text{-}mset\ set\text{-}mset\ P}))$
by (*metis Union-iff assms*(*1*) *image-eqI mset-big-union-obtain partition-on-msetD1 set-image-mset*)
then show $\bigcup\ (\mathit{set\text{-}mset}\ (\mathit{image\text{-}mset\ set\text{-}mset\ P})) = \mathit{set\text{-}mset\ A}$
using *set-eqI*′ *partition-on-mset-elems assms* **by** *auto*
show $\bigwedge p\ p'.\ p \in\# \mathit{image\text{-}mset\ set\text{-}mset\ P} \Longrightarrow p' \in\# \mathit{image\text{-}mset\ set\text{-}mset\ P} \Longrightarrow$

$p \neq p' \Longrightarrow p \cap p' = \{\}$
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* ⇒ *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* (‹$\mathcal{V}$›)
  fixes *block-collection* :: *'a set multiset* (‹$\mathcal{B}$›)
  assumes *wellformed*: $b \in\# \mathcal{B} \Longrightarrow b \subseteq \mathcal{V}$
begin

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

definition *incident* :: *'a* ⇒ *'a set* ⇒ *bool* **where**
incident p b ≡ $(p, b) \in \mathcal{I}$

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

abbreviation v ≡ *card* $\mathcal{V}$

abbreviation b ≡ *size* $\mathcal{B}$

Basic incidence lemmas

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

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

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

Abbreviations for all incidence systems

abbreviation *multiplicity* :: *'a set* ⇒ *nat* **where**
multiplicity b ≡ *count* $\mathcal{B}$ *b*

abbreviation *incomplete-block* :: *'a set* ⇒ *bool* **where**
incomplete-block bl ≡ *card bl* < *card* $\mathcal{V}$ ∧ *bl* ∈# $\mathcal{B}$

lemma *incomplete-alt-size*: *incomplete-block bl* $\Longrightarrow$ *card bl* $<$ v
by *simp*

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

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

definition *design-support* :: *'a set set* **where**
design-support $\equiv$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *card bl* $\leq$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *bl* $\neq$ {}

begin

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

lemma *wf-design-iff*: *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *design* $\mathcal{V}$ $\mathcal{B}$ $\longleftrightarrow$ (*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*: $\mathcal{B}$ $\neq$ {#} $\Longrightarrow$ $\mathcal{V}$ $\neq$ {}
using *wf-design wf-design-iff* **by** *auto*

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

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

Size lemmas

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

lemma *block-size-gt-0*: *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *b* $\subseteq$ *V*)
assumes $\bigwedge$ *b* . *b* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *b* $\neq$ {}
assumes *finite V*
assumes $\mathcal{B}$ $\neq$ {#}
assumes *V* $\neq$ {}
shows *design V* $\mathcal{B}$
using *assms* **by** (*unfold-locales*) *simp-all*

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

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

$\Longrightarrow \mathcal{V} \neq \{\} \Longrightarrow$ *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* ⇒ *'a* ⇒ *nat* (**infix** ‹*rep*› *75*)
where
B rep x ≡ *size* {#*b* ∈# *B* . *x* ∈ *b*#}

lemma *max-point-rep*: *B rep x* ≤ *size B*
using *size-filter-mset-lesseq* **by** (*simp add*: *point-replication-number-def*)

lemma *rep-number-g0-exists*:
assumes *B rep x* > *0*
obtains *b* **where** *b* ∈# *B* **and** *x* ∈ *b*
proof −
have *size* {#*b* ∈# *B* . *x* ∈ *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*: *finite B* ⟹ (*mset-set B*) *rep x* = *card* {*b* ∈ *B* . *x* ∈ *b*}
by (*simp add*: *point-replication-number-def*)

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

lemma *point-rep-singleton-val* [*simp*]: *x* ∈ *b* ⟹ {#*b*#} *rep x* = *1*
by (*simp add*: *point-replication-number-def*)

lemma *point-rep-singleton-inval* [*simp*]: *x* ∉ *b* ⟹ {#*b*#} *rep x* = *0*
by (*simp add*: *point-replication-number-def*)

context *incidence-system*
begin

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

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

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

lemma *point-rep-number-inv*: *size* $\{\#\ b \in\#\ \mathcal{B}\ .\ x \notin b\ \#\}$ = b − ($\mathcal{B}$ *rep* x)
proof −
have b = *size* $\{\#\ b \in\#\ \mathcal{B}\ .\ x \notin b\ \#\}$ + *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*: ($\mathcal{B}$ *rep* x) < b $\Longrightarrow \mathcal{B} \neq \{\#\} \Longrightarrow \{\#\ b \in\#$
$\mathcal{B}\ .\ x \notin b\ \#\} \neq \{\#\}$
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* ⇒ *'a set* ⇒ *nat* (**infix** ‹*index*› *75*) **where**
B index ps ≡ *size* $\{\#b \in\#\ B\ .\ ps \subseteq b\#\}$

lemma *points-index-empty* [*simp*]: $\{\#\}$ *index ps* = *0*
by (*simp add*: *points-index-def*)

lemma *point-index-distrib*: (*B1* + *B2*) *index ps* = *B1 index ps* + *B2 index ps*
by (*simp add*: *points-index-def*)

lemma *point-index-diff*: *B1 index ps* = (*B1* + *B2*) *index ps* − *B2 index ps*
by (*simp add*: *points-index-def*)

lemma *points-index-singleton*: $\{\#b\#\}$ *index ps* = *1* ⟷ *ps* ⊆ *b*
by (*simp add*: *points-index-def*)

lemma *points-index-singleton-zero*: ¬ (*ps* ⊆ *b*) ⟹ $\{\#b\#\}$ *index ps* = *0*
by (*simp add*: *points-index-def*)

lemma *points-index-sum*: ($\sum_\#$ *B*) *index ps* = ($\sum$ *b* ∈# *B* . (*b index ps*))
using *points-index-empty* **by** (*induction B*) (*auto simp add*: *point-index-distrib*)

lemma *points-index-block-image-add-eq*:
assumes *x* ∉ *ps*
assumes *B index ps* = *l*
shows $\{\#$ *insert x b* . *b* ∈# *B*$\#\}$ *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 (*mset-set B*) *index ps* = *card* {*b* ∈ *B*. *ps* ⊆ *b*}
by (*simp add*: *points-index-def assms*)

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

lemma *points-index-pair-rep-num*:
assumes ⋀ *b*. *b* ∈# *B* ⟹ *x* ∈ *b*
shows *B index* {*x*, *y*} = *B 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 index ps* = *0*
assumes *b* ∈# *B*
shows ¬ (*ps* ⊆ *b*)
proof (*rule ccontr*)
assume ¬ ¬ *ps* ⊆ *b*
then have *a*: *ps* ⊆ *b* **by** *auto*
then have *b* ∈# {#*bl* ∈# *B* . *ps* ⊆ *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 ⋀ *b* . *b* ∈# *B* ⟹ (¬ *ps* ⊆ *b*)
shows *B index ps* = *0*
using *assms* **by** (*simp add*: *filter-mset-empty-conv points-index-def*)

lemma *points-index-0-iff*: *B index ps* = *0* ⟷ (∀ *b*. *b* ∈# *B* ⟶ (¬ *ps* ⊆ *b*))
using *points-index-0-left-imp points-index-0-right-imp* **by** *metis*

lemma *points-index-gt0-impl-existance*:
assumes *B index ps* > *0*
shows (∃ *bl* . (*bl* ∈# *B* ∧ *ps* ⊆ *bl*))
proof −
have *size* {#*bl* ∈# *B* . *ps* ⊆ *bl*#} > *0*
by (*metis assms points-index-def*)
then obtain *bl* **where** *bl* ∈# *B* **and** *ps* ⊆ *bl*
by (*metis filter-mset-empty-conv nonempty-has-size*)
thus *?thesis* **by** *auto*
qed

lemma *points-index-one-unique*:
assumes *B index ps* = *1*
assumes *bl* ∈# *B* **and** *ps* ⊆ *bl* **and** *bl′* ∈# *B* **and** *ps* ⊆ *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-existance 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*: $\neg\ (ps \subseteq \mathcal{V}) \Longrightarrow \mathcal{B}$ *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 $\geq n \Longrightarrow ps \subseteq bl \Longrightarrow$ *count* $\{\#bl \in\# \mathcal{B} . ps \subseteq bl\#\}$ $bl \geq n$
by *simp*

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

assumes *card ps > card* $\mathcal{V}$
shows $\mathcal{B}$ *index ps = 0*
proof −
have $\bigwedge$ *b. b* $\in\#$ $\mathcal{B} \Longrightarrow$ *card ps > 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*#} $\Longrightarrow$ *ps* $\subseteq$ $\mathcal{V}$ $\Longrightarrow$ ($\mathcal{B}$ *index ps*) $\geq$ (*size x*)
by (*simp add*: *points-index-def size-mset-mono*)

lemma (**in** *design*) *points-index-count-min*: *multiplicity bl* $\geq$ *n* $\Longrightarrow$ *ps* $\subseteq$ *bl* $\Longrightarrow$ $\mathcal{B}$ *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** ‹|∩|› *70*) **where**
b1 |∩| *b2* $\equiv$ *card* (*b1* $\cap$ *b2*)

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

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

lemma *intersect-num-commute*: *b1* |∩| *b2* = *b2* |∩| *b1*
by (*simp add*: *inf-commute intersection-number-def*)

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

notation *n-intersect-number* (‹(- |∩|_ -)› [*52*, *51*, *52*] *50*)

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

lemma *n-inter-num-one*: *finite b1* $\Longrightarrow$ *finite b2* $\Longrightarrow$ *b1* $|\cap|_1$ *b2* = *b1* |∩| *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* $\Longrightarrow$ *finite b2* $\Longrightarrow$ *b1* $|\cap|_n$ *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*: $x \in A \Longrightarrow \{a \in A \,.\, a = x\} = \{x\}$
by *auto*

lemma (**in** *design*) *n-inter-num-zero*:
assumes *b1* $\in\#$ $\mathcal{B}$ **and** *b2* $\in\#$ $\mathcal{B}$
shows *b1* $|\cap|_0$ *b2* = *1*
proof −
have *empty*: $\bigwedge x$. *finite x* $\Longrightarrow$ *card x* = *0* $\Longrightarrow$ *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*) $\Longrightarrow$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *b2* $\in\#$ $\mathcal{B}$
$\Longrightarrow$ *b1* $|\cap|_n$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *b2* $\in\#$ $\mathcal{B}$
$\Longrightarrow$ *b1* $|\cap|_n$ *b2* = (*nat* (*b1* $|\cap|$ *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* | *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* $\Longrightarrow$ $\exists$ *x*. *x* $\in$ $\mathcal{V}$ $\wedge$ $\mathcal{B}$ *rep x* = *r*
using *replication-numbers-def* **by** *auto*

lemma *point-rep-number-in-set*: $x \in \mathcal{V} \Longrightarrow (\mathcal{B}\ rep\ x) \in replication\text{-}numbers$
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 $\equiv \{\ card\ bl \mid bl.\ bl \in\# \mathcal{B}\}$

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

lemma *sys-block-sizes-obtain-bl*: $x \in sys\text{-}block\text{-}sizes \Longrightarrow (\exists\ bl \in\# \mathcal{B}.\ card\ bl = x)$
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 $\equiv \{\ b1\ |\cap|\ b2 \mid b1\ b2\ .\ b1 \in\# \mathcal{B} \wedge b2 \in\# (\mathcal{B} - \{\#b1\#\})\}$

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

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

The set of all possible point indices

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

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

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

lemma *point-indices-alt-def*: *point-indices t* = { $\mathcal{B}$ *index ps* | *ps. card ps* = *t* ∧ *ps* ⊆ $\mathcal{V}$}
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* ⇒ *'a set* (‹-^{c}› [*56*] *55*) **where**
block-complement b ≡ $\mathcal{V}$ − *b*

definition *complement-blocks* :: *'a set multiset* (‹($\mathcal{B}^C$)›)**where**
complement-blocks ≡ {# *bl*^{c} . *bl* ∈# $\mathcal{B}$ #}

lemma *block-complement-elem-iff*:
assumes *ps* ⊆ $\mathcal{V}$
shows *ps* ⊆ *bl*^{c} ⟷ (∀ *x* ∈ *ps. x* ∉ *bl*)
using *assms block-complement-def* **by** (*auto*)

lemma *block-complement-inter-empty*: *bl1*^{c} = *bl2* ⟹ *bl1* ∩ *bl2* = {}
using *block-complement-def* **by** *auto*

lemma *block-complement-inv*:
assumes *bl* ∈# $\mathcal{B}$
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* ⊆ (*bl*^{c}) ⟹ *ps* ⊆ $\mathcal{V}$
using *block-complement-def* **by** *blast*

lemma *obtain-comp-block-orig*:
assumes *bl1* ∈# $\mathcal{B}^C$
obtains *bl2* **where** *bl2* ∈# $\mathcal{B}$ **and** *bl1* = *bl2*^{c}
using *wellformed assms* **by** (*auto simp add*: *complement-blocks-def*)

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

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

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

lemma *complement-index*:
assumes $ps \subseteq \mathcal{V}$
shows $\mathcal{B}^C$ *index ps* = *size* $\{\# b \in\# \mathcal{B} . (\forall x \in ps . x \notin b) \#\}$
proof −
have $\mathcal{B}^C$ *index ps* = *size* $\{\# b \in\# \{\# bl^c . bl \in\# \mathcal{B}\#\}. ps \subseteq b \#\}$
by (*simp add*: *complement-blocks-def points-index-def*)
then have $\mathcal{B}^C$ *index ps* = *size* $\{\# bl^c \mid bl \in\# \mathcal{B} . ps \subseteq bl^c \#\}$
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 $\mathcal{B}^C$ *index* $\{x, y\}$ = *size* $\{\# b \in\# \mathcal{B} . x \notin b \wedge y \notin b \#\}$
proof −
have *a*: $\bigwedge b. b \in\# \mathcal{B} \Longrightarrow \forall x' \in \{x, y\} . x' \notin b \Longrightarrow x \notin b \wedge y \notin b$
by *simp*
have $\bigwedge b. b \in\# \mathcal{B} \Longrightarrow x \notin b \wedge y \notin b \Longrightarrow \forall x' \in \{x, y\} . x' \notin b$
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** $\mathcal{B}$ *rep x* = *r*
shows $\mathcal{B}^C$ *rep x* = b − *r*
proof −
have *r*: *size* $\{\#b \in\# \mathcal{B} . x \in b\#\} = r$ **using** *assms* **by** (*simp add*: *point-replication-number-def*)
then have *a*: $\bigwedge b . b \in\# \mathcal{B} \Longrightarrow x \in b \Longrightarrow x \notin b^c$
by (*simp add*: *block-complement-def*)
have $\bigwedge b . b \in\# \mathcal{B} \Longrightarrow x \notin b \Longrightarrow x \in b^c$
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 b = *size* $\{\#b \in\# \mathcal{B} . x \in b\#\}$ + *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*: $bl \in\# \mathcal{B}^C \Longrightarrow bl \subseteq \mathcal{V}$
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*: $b \subseteq \mathcal{V} \Longrightarrow card\ (b^c) = card\ \mathcal{V} - card\ b$
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* $\Longrightarrow$ *card* $(bl^c) > 0$
using *block-complement-size* **by** (*simp add*: *wellformed*)

lemma *block-comp-incomplete-nempty*: *incomplete-block bl* $\Longrightarrow$ $bl^c \neq \{\}$
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* $\Longrightarrow$ $bl \subset \mathcal{V}$
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 $\bigwedge bl\ .\ bl \in\# \mathcal{B} \Longrightarrow$ *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* ⇒ *'a set multiset* **where**
multiple-blocks n ≡ *repeat-mset n* $\mathcal{B}$

lemma *multiple-block-in-original*: *b* ∈# *multiple-blocks n* ⟹ *b* ∈# $\mathcal{B}$
by (*simp add*: *elem-in-repeat-in-original*)

lemma *multiple-block-in*: *n* > *0* ⟹ *b* ∈# $\mathcal{B}$ ⟹ *b* ∈# *multiple-blocks n*
by (*simp add*: *elem-in-original-in-repeat*)

lemma *multiple-blocks-gt*: *n* > *0* ⟹ *size* (*multiple-blocks n*) ≥ *size* $\mathcal{B}$
by (*simp*)

lemma *block-original-count-le*: *n* > *0* ⟹ *count* $\mathcal{B}$ *b* ≤ *count* (*multiple-blocks n*) *b*
using *count-repeat-mset* **by** *simp*

lemma *multiple-blocks-sub*: *n* > *0* ⟹ $\mathcal{B}$ ⊆# (*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.prems* **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*: $\bigwedge b\ x.\ b \in\# multiple\text{-}blocks\ \ n \Longrightarrow x \in b \Longrightarrow x \in \mathcal{V}$
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* ∈# (*multiple-blocks n*)}
by (*simp add*: *mult-sys.sys-block-sizes-def*)
then have *eq*: $\bigwedge$ *bl*. *bl* ∈# (*multiple-blocks n*) ⟷ *bl* ∈# $\mathcal{B}$
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-nempty*
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* ∈# $\mathcal{B}$ ⟹ *multiplicity bl* = *1*

begin

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

lemma *simple-blocks-eq-sup*: *mset-set* (*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 = *card* (*design-support*)
  by (*metis simple-blocks-eq-sup size-mset-set*)

lemma *points-index-simple-def*: $\mathcal{B}$ *index ps* = *card* {*b* ∈ *design-support* . *ps* ⊆ *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*: $\mathcal{B}$ *rep x* = *card* {*b* ∈ *design-support* . *x* ∈ *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* ∈# $\mathcal{B}$ ⟹ *multiplicity b* > *1* ⟹ ¬ *simple-incidence-system* $\mathcal{V}$ $\mathcal{B}$
  using *simple-incidence-system-def simple-incidence-system-axioms-def* **by** (*metis nat-neq-iff*)

lemma *multiple-not-simple*:
  assumes *n* > *1*
  assumes $\mathcal{B}$ ≠ {#}
  shows ¬ *simple-incidence-system* $\mathcal{V}$ (*multiple-blocks n*)
proof (*rule ccontr*, *simp*)

assume *simple-incidence-system* $\mathcal{V}$ (*multiple-blocks n*)
then have $\bigwedge$ *bl*. *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *count* (*multiple-blocks n*) *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-nempty*: $\mathcal{V}$ $\neq$ {}
using *v-non-zero* **by** *auto*

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

end

Intro rules for a proper design

lemma (**in** *design*) *proper-designI*[*intro*]: b $\neq$ *0* $\Longrightarrow$ *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* $\in\#$ $\mathcal{B} \Longrightarrow$ *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* ⇒ *'a set* **where**
add-point p ≡ *insert p* $\mathcal{V}$

lemma *add-existing-point* [*simp*]: $p \in \mathcal{V} \Longrightarrow$ *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* ⇒ *'a set set* ⇒ *'a set multiset* **where**

add-point-to-blocks p bs ≡ {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *bs*#} + {# *b* ∈# $\mathcal{B}$. *b* ∉ *bs*#}

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

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

lemma *add-new-point-rep-number*:
 assumes *p* ∉ $\mathcal{V}$
 shows (*add-point-to-blocks p bs*) *rep p* = *size* {#*b* ∈# $\mathcal{B}$. *b* ∈ *bs*#}
proof −
 have ⋀ *b*. *b* ∈# $\mathcal{B}$ ⟹ *b* ∉ *bs* ⟹ *p* ∉ *b*
  by (*simp add*: *assms wf-invalid-point*)
 then have *zero*: {# *b* ∈# $\mathcal{B}$. *b* ∉ *bs*#} *rep p* = *0*
  by (*simp add*: *filter-mset-empty-conv point-replication-number-def*)
 have (*add-point-to-blocks p bs*) *rep p* = {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *bs*#} *rep p* + {# *b* ∈# $\mathcal{B}$. *b* ∉ *bs*#} *rep p*
  by (*simp add*: *add-point-to-blocks-def*)
 then have *eq*: (*add-point-to-blocks p bs*) *rep p* = {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *bs*#} *rep p*
  using *zero* **by** *simp*
 have ⋀ *bl* . *bl* ∈# {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *bs*#} ⟹ *p* ∈ *bl* **by** *auto*
 then have {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *bs*#} *rep p* = *size* {# (*insert p b*) | *b* ∈# $\mathcal{B}$. *b* ∈ *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-system* (*add-point p*) (*add-point-to-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* ⇒ *'a set* **where**
del-point p ≡ $\mathcal{V}$ − {*p*}

definition *del-point-blocks*:: *'a* ⇒ *'a set multiset* **where**
del-point-blocks p ≡ {# (*bl* − {*p*}) . *bl* ∈# $\mathcal{B}$ #}

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* ∉ $\mathcal{V}$ ⟹ *bl* ∈# $\mathcal{B}$ ⟹ *bl* − {*p*} = *bl*
using *wf-invalid-point* **by** *blast*

lemma *del-invalid-point*: *p* ∉ $\mathcal{V}$ ⟹ (*del-point p*) = $\mathcal{V}$
by (*simp add*: *del-point-def*)

lemma *del-invalid-point-blocks*: *p* ∉ $\mathcal{V}$ ⟹ (*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*: (⋀ *bl*. *bl* ∈# $\mathcal{B}$ ⟹ *p* ∉ *bl*) ⟹ (*del-point-blocks p*) = $\mathcal{B}$
by (*simp add*: *del-point-blocks-def*)

lemma *delete-point-blocks-wf*: *b* ∈# (*del-point-blocks p*) ⟹ *b* ⊆ $\mathcal{V}$ − {*p*}
unfolding *del-point-blocks-def* **using** *wellformed* **by** *auto*

lemma *delete-point-blocks-sub*:
assumes *b* ∈# (*del-point-blocks p*)
obtains *bl* **where** *bl* ∈# $\mathcal{B}$ ∧ *b* ⊆ *bl*
using *assms* **by** (*auto simp add*: *del-point-blocks-def*)

lemma *delete-point-split-blocks*: *del-point-blocks p* =
{# *bl* ∈#$\mathcal{B}$. *p* ∉ *bl*#} + {# *bl* − {*p*} | *bl* ∈# $\mathcal{B}$. *p* ∈ *bl*#}
proof −
have *sm*: ⋀ *bl* . *p* ∉ *bl* ⟹ *bl* − {*p*} = *bl* **by** *simp*
have *del-point-blocks p* = {# (*bl* − {*p*}) . *bl* ∈# $\mathcal{B}$ #} **by** (*simp add*: *del-point-blocks-def*)
also have ... = {# (*bl* − {*p*}) | *bl* ∈# $\mathcal{B}$. *p* ∉ *bl* #} + {# (*bl* − {*p*}) | *bl* ∈# $\mathcal{B}$. *p* ∈ *bl* #}
using *multiset-partition* **by** (*metis image-mset-union union-commute*)
finally have *del-point-blocks p* = {#*bl* | *bl* ∈# $\mathcal{B}$. *p* ∉ *bl*#} +
{# (*bl* − {*p*}) | *bl* ∈# $\mathcal{B}$. *p* ∈ *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* ⊆ (*del-point p*)
shows (*del-point-blocks p*) *index ps* = $\mathcal{B}$ *index ps*
proof −
have *eq*: *filter-mset* ((⊆) *ps*) {#*bl* − {*p*}. *bl* ∈# $\mathcal{B}$#} =

image-mset (λ *b* . *b* − {*p*}) (*filter-mset* (λ *b*. *ps* ⊆ *b* − {*p*}) $\mathcal{B}$)
using *filter-mset-image-mset* **by** *blast*
have *p* ∉ *ps* **using** *assms del-point-def* **by** *blast*
then have $\bigwedge$ *bl* . *ps* ⊆ *bl* ⟷ *ps* ⊆ *bl* − {*p*} **by** *blast*
then have ((*filter-mset* (λ *b*. *ps* ⊆ *b* − {*p*}) $\mathcal{B}$)) = (*filter-mset* (λ *b* . *ps* ⊆ *b*) $\mathcal{B}$)
by *auto*
then have *size* (*image-mset* (λ *b* . *b* − {*p*}) (*filter-mset* (λ *b*. *ps* ⊆ *b* − {*p*}) $\mathcal{B}$))

= $\mathcal{B}$ *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* ∈# $\mathcal{B}$. *p* ∉ *bl*#}

lemma *str-del-point-blocks-alt*: *str-del-point-blocks p* = $\mathcal{B}$ − {# *bl* ∈# $\mathcal{B}$. *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* ∈# $\mathcal{B}$ ⟹ *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* ∈# $\mathcal{B}$ ⟹ *bl* ∈# *str-del-point-blocks p* ⟷ *p* ∉ *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* ⊆# $\mathcal{B}$
by (*simp add*: *str-del-point-blocks-def*)

lemma *delete-point-strong-block-in-orig*: *bl* ∈# *str-del-point-blocks p* ⟹ *bl* ∈# $\mathcal{B}$
using *delete-point-strong-block-subset* **by** (*metis mset-subset-eqD*)

lemma *delete-invalid-pt-strong-eq*: *p* ∉ $\mathcal{V}$ ⟹ $\mathcal{B}$ = *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* ⊆ (*del-point p*)
shows (*str-del-point-blocks p*) *index ps* =
$\mathcal{B}$ *index ps* − {# *bl* ∈# $\mathcal{B}$. *p* ∈ *bl*#} *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* ⇒ *'a set multiset* **where**
add-block b ≡ $\mathcal{B}$ + {#*b*#}

lemma *add-block-alt*: *add-block b* = *add-mset b* $\mathcal{B}$
by (*simp add*: *add-block-def*)

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

lemma *add-block-rep-number-not-in*: *x* ∉ *b* ⟹ (*add-block b*) *rep x* = $\mathcal{B}$ *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* ⊆ *b*
shows (*add-block b*) *index ps* = $\mathcal{B}$ *index ps* + *1*
proof −
have (*add-block b*) = {#*b*#} + $\mathcal{B}$ **by** (*simp add*: *add-block-def*)
then have *split*: (*add-block b*) *index ps* = {#*b*#} *index ps* + $\mathcal{B}$ *index ps*
by (*metis point-index-distrib*)
have {#*b*#} *index ps* = *1* **using** *assms points-index-singleton* **by** *auto*
thus *?thesis* **using** *split* **by** *simp*
qed

lemma *add-block-index-not-in*: ¬ (*ps* ⊆ *b*) ⟹ (*add-block b*) *index ps* = $\mathcal{B}$ *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 then 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*: $b \subseteq \mathcal{V} \Longrightarrow$ *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*: (*del-block* b) $\subseteq\#$ $\mathcal{B}$
by (*simp add*: *del-block-def*)

lemma *delete-invalid-block-eq*: $b \notin\# \mathcal{B} \Longrightarrow$ *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* $b = \{\#\ bl - b\ .\ bl \in\#$ *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* $\{x\}$ = (*del-point-blocks* x)
proof −
have *neqx*: $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow bl \neq \{x\}$ **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* $\mathcal{B}$
using *add-point-wf* **by** *simp*

lemma *add-point-sys-rep-numbers*: *add-point-sys.replication-numbers p* =
replication-numbers ∪ {$\mathcal{B}$ *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* $\mathcal{V}$ ∪ *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* ⊆ $\mathcal{V}$
shows *add-block-sys.del-block bl bl* = $\mathcal{B}$
using *add-block-sys.del-block-def add-block-def* **by** *simp*

lemma *del-add-block-inv*: *bl* ∈# $\mathcal{B}$ ⟹ *del-block-sys.add-block bl bl* = $\mathcal{B}$
using *del-block-sys.add-block-def del-block-def wellformed* **by** *fastforce*

lemma *del-invalid-add-block-eq*: *bl* ∉# $\mathcal{B}$ ⟹ *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* ∉ $\mathcal{V}$
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 *card* (*del-point p*) = v − *1*
proof −
have *vg0*: *card* $\mathcal{V} > 0$ **using** *assms finite-sets card-gt-0-iff* **by** *auto*
then have *card* (*del-point p*) = *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* $\Longrightarrow$ *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*: $b \subseteq \mathcal{V} \Longrightarrow$ *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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *p* $\in$ *bl* $\Longrightarrow$ *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* $\in\#$ (*del-point-blocks p*) $\Longrightarrow$ *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'* $\Longrightarrow$ *bl* $\neq$ {} **using** *block blocks-nempty* **by** (*simp add*: *rem*)
have *p* $\in$ *bl'* $\Longrightarrow$ *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* $\in\#$ (*del-block bl*) $\Longrightarrow$ *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* $\in\#$ $\mathcal{B} \Longrightarrow \neg$ (*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* $\in\#$ $\mathcal{B} \Longrightarrow$ *p* $\in$ *bl* $\Longrightarrow$ *2* $\leq$ *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* $\in\#$ $\mathcal{B} \Longrightarrow$ *p* $\in$ *bl* $\Longrightarrow$ *2* $\leq$ *card bl*
assumes $\mathcal{B}$ *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 set*) **and** $\mathcal{B}$ **and** $\mathcal{V}'$ :: ('*a 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*: $\mathcal{V} = \mathcal{V}' \Longrightarrow \mathcal{B} = \mathcal{B}' \Longrightarrow \mathcal{B}^+$ = *sys1.multiple-blocks 2*
by *auto*

lemma *combine-multiple2*: $\mathcal{V} = \mathcal{V}' \Longrightarrow \mathcal{B} = \mathcal{B}' \Longrightarrow \mathcal{B}^+$ = *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 ∪ *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*: *card* ($\mathcal{V}^+$) $\geq$ *card* $\mathcal{V}$
  by (*meson Un-upper1 card-mono combine-fin-sys.finite-sets*)

lemma *combine-order-2*: *card* ($\mathcal{V}^+$) $\geq$ *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*: $\mathcal{B} \neq \{\#\} \Longrightarrow$ *proper-design* $\mathcal{V}$ $\mathcal{B}$
  by (*unfold-locales*) (*simp*)

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

lemma *min-one-proper-design*: *proper-design* $\mathcal{V}$ $\mathcal{B}$ $\vee$ *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* (‹$\mathcal{K}$›)
  assumes *block-sizes*: *bl* ∈# $\mathcal{B}$ ⟹ *card bl* ∈ $\mathcal{K}$
  assumes *positive-ints*: $x \in \mathcal{K} \Longrightarrow x > 0$
begin

lemma *sys-block-size-subset*: *sys-block-sizes* ⊆ $\mathcal{K}$
  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* (‹k›)
  assumes *uniform* [*simp*]: *bl* ∈# $\mathcal{B}$ ⟹ *card bl* = k
begin

lemma *k-non-zero*: k ≥ *1*
proof −
  obtain *bl* **where** *bl-in*: *bl* ∈# $\mathcal{B}$
    using *design-blocks-nempty* **by** *auto*
  then have *card bl* ≥ *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*: ∀ *bl* ∈# $\mathcal{B}$.*card bl* = k
  using *uniform* **by** *auto*

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

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

lemma *sys-block-sizes-uniform* [*simp*]: $sys\text{-}block\text{-}sizes = \{\mathrm{k}\}$
proof −
have $sys\text{-}block\text{-}sizes = \{bs\ .\ \exists\ bl\ .\ bs = card\ bl \wedge bl \in\# \mathcal{B}\}$ **by** (*simp add*: *sys-block-sizes-def*)
then have $sys\text{-}block\text{-}sizes = \{bs\ .\ bs = \mathrm{k}\}$ **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*: $\mathrm{k} \leq \mathrm{v} - 1 \Longrightarrow bl \in\# \mathcal{B} \Longrightarrow incomplete\text{-}block\ bl$
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 $card\ bl = \mathrm{v} - \mathrm{k}$
proof −
obtain bl' **where** *bl-assm*: $bl = bl'^c \wedge bl' \in\# \mathcal{B}$
using *wellformed assms* **by** (*auto simp add*: *complement-blocks-def*)
then have $int\ (card\ bl') = \mathrm{k}$ **by** *simp*
thus *?thesis* **using** *bl-assm block-complement-size wellformed*
by (*simp add*: *block-size-lt-order of-nat-diff*)
qed

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

lemma *block-size-lt-v*: $\mathrm{k} \leq \mathrm{v}$
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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *card bl* = *k*)
$\Longrightarrow$ *block-design* $\mathcal{V}$ $\mathcal{B}$ *k*
by (*unfold-locales*) (*auto*)

context *block-design*
begin

lemma *block-design-multiple*: *n* > *0* $\Longrightarrow$ *block-design* $\mathcal{V}$ (*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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *incomplete-block bl*
using *incomplete uniform uniform-size-incomp* **by** *fastforce*

lemma *incomplete-imp-proper-subset*: *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *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 $\Longrightarrow$ *incomplete-design*
$\mathcal{V}$ $\mathcal{B}$ k
by *unfold-locales auto*

context *incomplete-design*
begin

lemma *multiple-incomplete*: *n* > *0* $\Longrightarrow$ *incomplete-design* $\mathcal{V}$ (*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 λ_t blocks

locale *t-wise-balance* = *proper-design* +
fixes *grouping* :: *nat* (‹t›) **and** *index* :: *nat* (‹Λ_t›)
assumes *t-non-zero*: t ≥ *1*
assumes *t-lt-order*: t ≤ v
assumes *balanced* [*simp*]: *ps* ⊆ $\mathcal{V}$ ⟹ *card ps* = t ⟹ $\mathcal{B}$ *index ps* = Λ_t
begin

lemma *t-non-zero-suc*: t ≥ *Suc 0*
using *t-non-zero* **by** *auto*

lemma *balanced-alt-def-all*: ∀ *ps* ⊆ $\mathcal{V}$. *card ps* = t ⟶ $\mathcal{B}$ *index ps* = Λ_t
using *balanced* **by** *auto*

end

lemma (**in** *proper-design*) *t-wise-balanceI*[*intro*]: t ≤ v ⟹ t ≥ *1* ⟹
(⋀ *ps* . *ps* ⊆ $\mathcal{V}$ ⟹ *card ps* = t ⟹ $\mathcal{B}$ *index ps* = Λ_t) ⟹ *t-wise-balance* $\mathcal{V}$ $\mathcal{B}$ t Λ_t
by (*unfold-locales*) *auto*

context *t-wise-balance*
begin

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

lemma *multiple-t-wise-balance-index* [*simp*]:
assumes *ps* ⊆ $\mathcal{V}$
assumes *card ps* = t
shows (*multiple-blocks n*) *index ps* = Λ_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 (Λ_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-lt-order multiple-t-wise-balance-index*
by (*unfold-locales*) (*simp-all*)
qed

lemma *twise-set-pair-index*: $ps \subseteq \mathcal{V} \Longrightarrow ps2 \subseteq \mathcal{V} \Longrightarrow ps \neq ps2 \Longrightarrow$ *card ps* = t $\Longrightarrow$ *card ps2* = t
$\Longrightarrow \mathcal{B}$ *index ps* = $\mathcal{B}$ *index ps2*
using *balanced* **by** *simp*

lemma *t-wise-balance-alt*: $ps \subseteq \mathcal{V} \Longrightarrow$ *card ps* = t $\Longrightarrow \mathcal{B}$ *index ps* = *l2*
$\Longrightarrow (\bigwedge ps\ .\ ps \subseteq \mathcal{V} \Longrightarrow$ *card ps* = t $\Longrightarrow \mathcal{B}$ *index ps* = *l2*)
using *twise-set-pair-index* **by** *blast*

lemma *index-1-imp-mult-1* [*simp*]:
assumes Λ_t = *1*
assumes *bl* $\in\#$ $\mathcal{B}$
assumes *card bl* $\geq$ t
shows *multiplicity bl* = *1*
proof (*rule ccontr*)
assume ¬ (*multiplicity bl* = *1*)
then have *not*: *multiplicity bl* ≠ *1* **by** *simp*
have *multiplicity bl* ≠ *0* **using** *assms* **by** *simp*
then have *m*: *multiplicity bl* ≥ *2* **using** *not* **by** *linarith*
obtain *ps* **where** *ps*: $ps \subseteq bl \wedge$ *card ps* = t
using *assms obtain-t-subset-points*
by (*metis obtain-subset-with-card-n*)
then have $\mathcal{B}$ *index ps* ≥ *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* $\mathcal{V}$ $\mathcal{B}$ *2* Λ
for *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *index* (‹Λ›)

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*: $k \in \mathcal{K} \Longrightarrow k \geq$ t

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

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

locale *PBD* = Λ-*PBD* $\mathcal{V}$ $\mathcal{B}$ *1* $\mathcal{K}$ **for** *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *sizes* (‹$\mathcal{K}$›)
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*-Λ-*PBD* = *pairwise-balance* + *block-design* +
  assumes *block-size-t*: *2* $\leq$ k

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

locale *k-PBD* = *k*-Λ-*PBD* $\mathcal{V}$ $\mathcal{B}$ *1* k **for** *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *u-block-size* (‹k›)

sublocale *k-PBD* $\subseteq$ *PBD* $\mathcal{V}$ $\mathcal{B}$ {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* (‹t›)
  fixes *min-index* :: *nat* (‹Λ_t›)
  assumes *covering*: *ps* $\subseteq$ $\mathcal{V}$ $\Longrightarrow$ *card ps* = t $\Longrightarrow$ $\mathcal{B}$ *index ps* $\geq$ Λ_t
  assumes *block-size-t*: t $\leq$ k
  assumes *t-non-zero*: t $\geq$ *1*
begin

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

using *covering* **by** *auto*

end

lemma (**in** *block-design*) *t-covering-designI* [*intro*]: $t \leq \mathrm{k} \Longrightarrow t \geq 1 \Longrightarrow$
$(\bigwedge ps.\ ps \subseteq \mathcal{V} \Longrightarrow card\ ps = t \Longrightarrow \mathcal{B}\ index\ ps \geq \Lambda_t) \Longrightarrow t\text{-}covering\text{-}design\ \mathcal{V}\ \mathcal{B}$ k t Λ_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* (‹t›)
fixes *min-index* :: *nat* (‹Λ_t›)
assumes *packing*: $ps \subseteq \mathcal{V} \Longrightarrow card\ ps = \mathrm{t} \Longrightarrow \mathcal{B}\ index\ ps \leq \Lambda_t$
assumes *block-size-t*: t ≤ k
assumes *t-non-zero*: t ≥ *1*
begin

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

end

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

lemma *packing-covering-imp-balance*:
assumes *t-packing-design V B k t* Λ_t
assumes *t-covering-design V B k t* Λ_t
shows *t-wise-balance V B t* Λ_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* ≤ *t* **using** *assms t-packing-design.t-non-zero* **by** *auto*
show *t* ≤ *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 $\bigwedge ps.\ ps \subseteq V \Longrightarrow card\ ps = t \Longrightarrow B\ index\ ps = \Lambda_t$
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*]: $x \in \mathcal{V} \Longrightarrow \mathcal{B}\ rep\ x = \mathrm{r}$

begin

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

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

lemma *rep-numbers-constant* [*simp*]: *replication-numbers* $= \{\mathrm{r}\}$
  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*: $x1 \in \mathcal{V} \Longrightarrow x2 \in \mathcal{V} \Longrightarrow x1 \neq x2 \Longrightarrow \mathcal{B}\ rep\ x1 = \mathcal{B}\ rep\ x2$
  using *rep-number* **by** *auto*

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

lemma *constant-rep-point-not-0*:
  assumes $x \in \mathcal{V}$
  shows $\mathcal{B}\ rep\ x \neq 0$
proof (*rule ccontr*)
  assume $\neg\ \mathcal{B}\ rep\ x \neq 0$
  then have $\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ rep\ x = 0$ **using** *rep-number assms* **by** *auto*
  then have $\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow size\ \{\#bl \in\# \mathcal{B}\ .\ x \in bl\#\} = 0$
    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*: $\mathrm{r} \neq 0$
  using *rep-number constant-rep-point-not-0 design-points-nempty* **by** *auto*

lemma *r-gzero*: $\mathrm{r} > 0$

using *rep-not-zero* **by** *auto*

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

lemma *complement-rep-number*:
assumes ⋀ *bl* . *bl* ∈# $\mathcal{B}$ ⟹ *incomplete-block bl*
shows *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 *constant-rep-design* $\mathcal{V}$ (*multiple-blocks n*) (r ∗ *n*)
proof −
interpret *d*: *proper-design* $\mathcal{V}$ (*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*]: (⋀ *x* . *x* ∈ $\mathcal{V}$ ⟹ $\mathcal{B}$ *rep x* = r)
⟹ *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 ≤ k
begin

lemma *point-indices-balanced*: *point-indices* t = {Λ_t}
proof −
have *point-indices* t = {*i* . ∃ *ps* . *i* = $\mathcal{B}$ *index ps* ∧ *card ps* = t ∧ *ps* ⊆ $\mathcal{V}$}
by (*simp add*: *point-indices-def*)
then have *point-indices* t = {*i* . *i* = Λ_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* Λ_t
assumes $t \leq k$
shows *t-design V B k t* Λ_t
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *t-design.intro t-design-axioms.intro*)

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

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

lemma *t-design-pack-cov* [*intro*]:
assumes $k <$ *card V*
assumes *t-covering-design V B k t* Λ_t
assumes *t-packing-design V B k t* Λ_t
shows *t-design V B k t* Λ_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* Λ_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* $\mathcal{V}$ $\mathcal{B}$ t Λ_t {k}
using *uniform k-non-zero block-size-t* **by** (*unfold-locales*) *simp-all*

context *t-design*
begin

lemma *multiple-t-design*: $n > 0 \Longrightarrow$ *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* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *u-block-size* (‹k›) **and** *grouping* (‹t›)

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*]: $\Lambda_t = 1 \Longrightarrow$ *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 Λ_t +
des2: *t-wise-balance* $\mathcal{V}$ $\mathcal{B}'$ t $\Lambda_t{}'$ **for** $\mathcal{V}$ $\mathcal{B}$ t Λ_t $\mathcal{B}'$ $\Lambda_t{}'$
begin

lemma *combine-t-wise-balance-index*: *ps* $\subseteq$ $\mathcal{V}$ $\Longrightarrow$ *card ps* = t $\Longrightarrow$ $\mathcal{B}^+$ *index ps* = (Λ_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 (Λ_t + $\Lambda_t{}'$)
proof (*unfold-locales*, *simp add*: *des1.t-non-zero-suc*)
have *card* $\mathcal{V}^+$ $\geq$ *card* $\mathcal{V}$ **by** *simp*
then show t $\leq$ *card* ($\mathcal{V}^+$) **using** *des1.t-lt-order* **by** *linarith*
show $\bigwedge$*ps*. *ps* $\subseteq$ $\mathcal{V}^+$ $\Longrightarrow$ *card ps* = t $\Longrightarrow$ ($\mathcal{B}^+$ *index ps*) = Λ_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 (Λ_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 $\mathcal{V}$ $\mathcal{B}'$
+ *two-t-wise-eq-points* $\mathcal{V}$ $\mathcal{B}$ t Λ_t $\mathcal{B}'$ Λ_t′ + *des1*: *t-design* $\mathcal{V}$ $\mathcal{B}$ k t Λ_t +
des2: *t-design* $\mathcal{V}$ $\mathcal{B}'$ k t Λ_t′ **for** $\mathcal{V}$ $\mathcal{B}$ k $\mathcal{B}'$ t Λ_t Λ_t′
begin

lemma *combine-is-t-des*: *t-design* $\mathcal{V}^+$ $\mathcal{B}^+$ k t (Λ_t + Λ_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 (Λ_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* Λ
for *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›)
and *u-block-size* (‹k›) **and** *index* (‹Λ›)

begin

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

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

lemma *index-one-empty-rm-blv* [*simp*]:
assumes Λ = *1* **and** *blv* ∈# $\mathcal{B}$ **and** *p* ⊆ *blv* **and** *card p* = *2*
shows {#*bl* ∈# *remove1-mset blv* $\mathcal{B}$. *p* ⊆ *bl*#} = {#}
proof −
have *blv-in*: *blv* ∈# *filter-mset* ((⊆) *p*) $\mathcal{B}$
using *assms* **by** *simp*
have *p* ⊆ $\mathcal{V}$ **using** *assms wellformed* **by** *auto*
then have *size* (*filter-mset* ((⊆) *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 Λ = *1* **and** *blv* ∈# $\mathcal{B}$ **and** *p* ⊆ *blv* **and** *card p* = *2*

shows $\bigwedge bl.\ bl \in\# remove1\text{-}mset\ blv\ \mathcal{B} \Longrightarrow \neg\ (p \subseteq bl)$
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, λ)-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 *size* ({# $p \in\#$ (*mset-set* ($\mathcal{V} - \{x\}$) $\times\#$ {# $bl \in\#\ \mathcal{B}\ .\ x \in bl$ #}). *fst p* $\in$ *snd p*#}) = $\Lambda * (\mathrm{v} - 1)$
proof −
let *?M* = *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* $\Longrightarrow y \in \mathcal{V}$ **using** *assms*
by (*simp add*: *finite-sets*)
have *b-contents*: $\bigwedge bl.\ bl \in\#$ *?B* $\Longrightarrow x \in bl$ **using** *assms* **by** *auto*
have $\bigwedge y.\ y \in\#$ *?M* $\Longrightarrow y \neq x$ **using** *assms*
by (*simp add*: *finite-sets*)
then have $\bigwedge y\ .y \in\#$ *?M* $\Longrightarrow$ *size* ({# $bl \in\#$ *?B* . $\{x, y\} \subseteq bl$#}) = Λ
using *points-index-pair filter-filter-mset-ss-member y-point assms finite-sets*
by *metis*
then have $\bigwedge y\ .y \in\#$ *?M* $\Longrightarrow$ *size* ({# $bl \in\#$ *?B* . $x \in bl \wedge y \in bl$#}) = Λ
by *auto*
then have *bl-set-size*: $\bigwedge y\ .\ y \in\#$ *?M* $\Longrightarrow$ *size* ({# $bl \in\#$ *?B* . $y \in bl$#}) = Λ
using *b-contents* **by** (*metis* (*no-types*, *lifting*) *filter-mset-cong*)
then have *final-size*: *size* ($\sum p \in\# ?M$. ({#p#} $\times\#$ {#$bl \in\#$ *?B*. $p \in bl$#})) = *size* (*?M*) $* \Lambda$
using *m-distinct size-Union-distinct-cart-prod-filter bl-set-size* **by** *blast*
have *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 *size* ({# $p \in\#$ (*mset-set* ($\mathcal{V} - \{x\}$) $\times\#$ {# $bl \in\#\ \mathcal{B}\ .\ x \in bl$ #}). *fst p* $\in$ *snd p*#})
= ($\mathcal{B}$ *rep x*) $* (\mathrm{k} - 1)$
(**is** *size* ({# $p \in\#$ (*?M* $\times\#$ *?B*). *fst p* $\in$ *snd p*#}) = ($\mathcal{B}$ *rep x*) $* (\mathrm{k} - 1)$)
proof −
have $\bigwedge y.\ y \in\#$ *?M* $\Longrightarrow y \neq x$ **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*. *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*. *card* $\{p \in (\mathcal{V} - \{x\})$. $p \in bl\}$ = *card* $(\{p \in \mathcal{V}$. $p \in bl\} - \{x\})$
by (*simp add*: *set-filter-diff-card*)
then have $\forall$ *bl* $\in\#$ *?B*. *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* $\Longrightarrow$ *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*: $x \in \mathcal{V} \Longrightarrow (\mathcal{B}$ *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 $(\mathcal{B}$ *rep x*) $= \Lambda *$ (v $-$ *1*) *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*: $\forall$ $x \in \mathcal{V}$. $\mathcal{B}$ *rep x* $= \Lambda *$ (v $-$ *1*) *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*) *div* (k $-$ *1*))
using *r-constant-alt* **by** (*unfold-locales*) *simp-all*

lemma (**in** *t-design*) *bibdI* [*intro*]: t $= 2 \Longrightarrow$ *bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ_t
using *t-lt-order block-size-t* **by** (*unfold-locales*) (*simp-all*)

context *bibd*
begin

abbreviation r $\equiv (\Lambda *$ (v $-$ *1*) *div* (k $-$ *1*))

lemma *necessary-condition-one*:
shows r ∗ (k − *1*) = Λ ∗ (v − *1*)
using *necess-cond-1-rhs necess-cond-1-lhs design-points-nempty rep-number* **by** *auto*

lemma *bibd-point-occ-rep*:
assumes *x* ∈ *bl*
assumes *bl* ∈# $\mathcal{B}$
shows ($\mathcal{B}$ − {#*bl*#}) *rep x* = r − *1*
proof −
have *xin*: *x* ∈ $\mathcal{V}$ **using** *assms wf-invalid-point* **by** *blast*
then have *rep*: *size* {# *blk* ∈# $\mathcal{B}$. *x* ∈ *blk* #} = r **using** *rep-number-unfold-set* **by** *simp*
have ($\mathcal{B}$ − {#*bl*#}) *rep x* = *size* {# *blk* ∈# ($\mathcal{B}$ − {#*bl*#}). *x* ∈ *blk* #}
by (*simp add*: *point-replication-number-def*)
then have ($\mathcal{B}$ − {#*bl*#}) *rep x* = *size* {# *blk* ∈# $\mathcal{B}$. *x* ∈ *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*: *size* {# *x* ∈# (*mset-set* $\mathcal{V}$ ×# $\mathcal{B}$) . (*fst x*) ∈ (*snd x*) #} = v ∗ r
proof −
let *?M* = *mset-set* $\mathcal{V}$
have ⋀ *p* . *p* ∈# *?M* ⟹ *size* ({# *bl* ∈# $\mathcal{B}$. *p* ∈ *bl* #}) = r
using *finite-sets rep-number-unfold-set r-gzero nat-eq-iff2* **by** *auto*
then have *size* (∑ *p*∈#*?M*. ({#*p*#} ×# {#*bl* ∈# $\mathcal{B}$. *p* ∈ *bl*#})) = *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*: *size* {# *x* ∈# (*mset-set* $\mathcal{V}$ ×# $\mathcal{B}$) . (*fst x*) ∈ (*snd x*) #} = b∗k
(**is** *size* {# *x* ∈# (*?M* ×# *?B*). (*fst x*) ∈ (*snd x*) #} = b∗k)
proof −
have ⋀ *bl* . *bl* ∈# *?B* ⟹ *size* ({# *p* ∈# *?M* . *p* ∈ *bl* #}) = k
using *uniform k-non-zero uniform-unfold-point-set-mset* **by** *fastforce*
then have *size* (∑ *bl*∈#*?B*. ({# *p* ∈# *?M* . *p* ∈ *bl* #} ×# {#*bl*#})) = *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*: $b = \Lambda * v * (v - 1)$ *div* $(k * (k-1))$
proof −
have $b * k = (v * r)$ **using** *necessary-condition-two* **by** *simp*
then have *k-dvd*: k *dvd* $(v * r)$ **by** (*metis dvd-triv-right*)
then have $b = (v * r)$ *div* k **using** *necessary-condition-two min-block-size-2* **by** *auto*
then have $b = (v * ((\Lambda * (v - 1)\ div\ (k - 1))))$ *div* k **by** *simp*
then have $b = (v * \Lambda * (v - 1))$ *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*: $\Lambda * (v - 1) = k * (k - 1) \Longrightarrow b = v \wedge r = k$
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 ≠ *0* **using** *k-non-zero b-non-zero* **by** *auto*
then have *rhsnot0*: v ∗ r ≠ *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 ⟹ 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 + Λ − *2*∗r
proof −
have *xin*: $x \in \mathcal{V}$ **and** *yin*: $y \in \mathcal{V}$ **using** *assms* **by** *auto*
have *ge*: *2*∗r ≥ Λ **using** *index-lt-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*: ... = *size* $\mathcal{B}$ − (*size* $\{\# b \in\# \mathcal{B}\ .\ \neg\ (x \notin b \wedge y \notin b)\ \#\}$)
using *size-filter-neg* **by** *blast*
also have ... = *size* $\mathcal{B}$ − (*size* $\{\# b \in\# \mathcal{B}\ .\ x \in b \vee y \in b\#\}$) **by** *auto*

also have ... = b − (*size* {# *b* ∈# $\mathcal{B}$. *x* ∈ *b* ∨ *y* ∈ *b*#}) **by** (*simp add*: *of-nat-diff*)
finally have $\mathcal{B}^C$ *index* {*x*, *y*} = b − (*size* {# *b* ∈# $\mathcal{B}$. *x* ∈ *b*#} +
size {# *b* ∈# $\mathcal{B}$. *y* ∈ *b*#} − *size* {# *b* ∈# $\mathcal{B}$. *x* ∈ *b* ∧ *y* ∈ *b*#})
by (*simp add*: *mset-size-partition-dep s2 s1*)
then have $\mathcal{B}^C$ *index* {*x*, *y*} = b − (r + r − Λ) **using** *rep-number-unfold-set lambda xin yin*
by *presburger*
then have $\mathcal{B}^C$ *index* {*x*, *y*} = b − (*2*∗r − Λ)
using *index-lt-replication* **by** (*metis mult-2*)
thus *?thesis* **using** *ge diff-diff-right* **by** *simp*
qed

lemma *complement-bibd-index*:
assumes *ps* ⊆ $\mathcal{V}$
assumes *card ps* = *2*
shows $\mathcal{B}^C$ *index ps* = b + Λ − *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* ≠ *y* **using** *assms* **by** *auto*
thus *?thesis* **using** *comp-params-index-pair assms*
by (*simp add*: *set*)
qed

lemma *complement-bibd*:
assumes k ≤ v − *2*
shows *bibd* $\mathcal{V}$ $\mathcal{B}^C$ (v − k) (b + Λ − *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* ≤ *des*.v **using** *assms block-size-t* **by** *linarith*
show ⋀*ps*. *ps* ⊆ $\mathcal{V}$ ⟹ *card ps* = *2* ⟹
$\mathcal{B}^C$ *index ps* = b + Λ − *2* ∗ (Λ ∗ (*des*.v − *Suc 0*) *div* (k − *Suc 0*))
using *complement-bibd-index* **by** *simp*
show *2* ≤ *des*.v − k **using** *assms block-size-t* **by** *linarith*
qed
qed

lemma *multiple-bibd*: *n* > *0* ⟹ *bibd* $\mathcal{V}$ (*multiple-blocks n*) k (Λ ∗ *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* Λ Λ′
+ *des1*: *bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ + *des2*: *bibd* $\mathcal{V}$ $\mathcal{B}'$ k Λ′ **for** $\mathcal{V}$ $\mathcal{B}$ k $\mathcal{B}'$ Λ Λ′
begin

lemma *combine-is-bibd*: *bibd* $\mathcal{V}^+$ $\mathcal{B}^+$ k $(\Lambda + \Lambda')$
by (*unfold-locales*)

sublocale *combine-bibd*: *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* (‹*bl*›)
assumes *valid-block*: *bl* $\in\#$ $\mathcal{B}$
begin

definition *derived-blocks* :: *'a set multiset* (‹($\mathcal{B}^D$)›) **where**
$\mathcal{B}^D \equiv \{\#\ bl \cap b\ .\ b \in\#\ (\mathcal{B} - \{\#bl\#\})\ \#\}$

lemma *derive-define-flip*: $\{\#\ b \cap bl\ .\ b \in\#\ (\mathcal{B} - \{\#bl\#\})\ \#\} = \mathcal{B}^D$
by (*simp add*: *derived-blocks-def inf-sup-aci*(*1*))

lemma *derived-points-order*: *card bl* = k
using *uniform valid-block* **by** *simp*

lemma *derived-block-num*: *bl* $\in\#$ $\mathcal{B} \Longrightarrow$ *size* $\mathcal{B}^D$ = b − *1*
by (*simp add*: *derived-blocks-def size-remove1-mset-If valid-block*)

lemma *derived-is-wellformed*: $b \in\#\ \mathcal{B}^D \Longrightarrow b \subseteq bl$
by (*simp add*: *derived-blocks-def valid-block*) (*auto*)

lemma *derived-point-subset-orig*: $ps \subseteq bl \Longrightarrow ps \subset \mathcal{V}$
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** $b = b2 \cap bl$ **and** $b2 \in\#$ *remove1-mset bl* $\mathcal{B}$
using *assms derived-blocks-def* **by** *auto*

sublocale *derived-incidence-sys*: *incidence-system bl* $\mathcal{B}^D$
using *derived-is-wellformed valid-block* **by** (*unfold-locales*) (*auto*)

sublocale *derived-fin-incidence-system*: *finite-incidence-system bl* $\mathcal{B}^D$
using *valid-block finite-blocks* **by** (*unfold-locales*) *simp-all*

lemma *derived-blocks-nempty*:
assumes $\bigwedge$ *b* .*b* $\in\#$ *remove1-mset bl* $\mathcal{B} \Longrightarrow$ *bl* $|\cap|$ *b* > *0*

assumes *bld* $\in\#$ $\mathcal{B}^D$
shows *bld* $\neq$ {}
proof −
obtain *bl2* **where** *inter*: *bld* = *bl2* $\cap$ *bl* **and** *member*: *bl2* $\in\#$ *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* $\in\#$ *remove1-mset bl* $\mathcal{B}$ $\Longrightarrow$ *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* $\in\#$ *remove1-mset bl* $\mathcal{B}$ $\Longrightarrow$ *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* (‹($\mathcal{B}^R$)›) **where**
$\mathcal{B}^R \equiv \{\#\ b - bl\ .\ b \in\#\ (\mathcal{B} - \{\#bl\#\})\ \#\}$

lemma *residual-order*: *card* (bl^c) = v − k
by (*simp add*: *valid-block wellformed block-complement-size*)

lemma *residual-block-num*: *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* $\in\#$ *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*: $b \in\# \mathcal{B}^R \Longrightarrow x \in b \Longrightarrow x \notin bl$
using *residual-obtain-orig-block* **by** *auto*

lemma *residual-is-wellformed*: $b \in\# \mathcal{B}^R \Longrightarrow b \subseteq (bl^c)$
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*: $bl2 \in\# remove1\text{-}mset\ bl\ \mathcal{B}$
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 *card bl2* = *card bl* **using** *uniform valid-block member*
using *in-diffD* **by** *fastforce*
then have *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* $\Longrightarrow$ *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* bl^c $\mathcal{B}^R$ **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*: Λ ∗ (v − *1*) = k ∗ (k − *1*)
using *necessary-condition-one rep-value-sym* **by** *auto*

lemma *sym-design-vk-gt-kl*:
assumes k ≥ Λ + *2*
shows v − k > k − Λ
proof (*rule ccontr*)
define *k l v* **where** *kdef*: *k* ≡ *int* k **and** *ldef*: *l* ≡ *int* Λ **and** *vdef*: *v* ≡ *int* v
assume ¬ (v − k > k − Λ)
then have *a*: ¬ (*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* ≥ *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* ≤ *2* ∗ *k* − *l* **using** *a* **by** *linarith*
then have *v* − *1* ≤ *2* ∗ *k* − *l* − *1* **by** *linarith*
then have *l*∗ (*v* − *1*) ≤ *l*∗(*2* ∗ *k* − *l* − *1*)
using *lge mult-le-cancel-left* **by** *fastforce*
then have *k* ∗ (*k* − *1*) ≤ *l*∗(*2* ∗ *k* − *l* − *1*)
by (*simp add*: *sym*)
then have *k* ∗ (*k* − *1*) − *l*∗(*2* ∗ *k* − *l* − *1*) ≤ *0* **by** *linarith*
then have *k*^*2* − *k* − *l*∗ *2* ∗ *k* + *l*^*2* + *l* ≤ *0*
by (*simp add*: *mult-ac right-diff-distrib*′ *power2-eq-square*)
then have (*k* − *l*)∗(*k* − *l* − *1*) ≤ *0*
by (*simp add*: *mult-ac right-diff-distrib*′ *power2-eq-square*)
then have *k* = *l* ∨ *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 $\Longrightarrow$ *symmetric-bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ
by *unfold-locales simp*

lemma *symmetric-bibdII*: Λ * (v − *1*) = k * (k − *1*) $\Longrightarrow$ *symmetric-bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ
using *symmetric-condition-1* **by** *unfold-locales blast*

lemma *symmetric-not-admissable*: Λ * (v − *1*) ≠ k * (k − *1*) $\Longrightarrow$ ¬ *symmetric-bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ
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* ∈# $\mathcal{B}$
shows *p* ∈# $\sum_{\#}${# *mset-set* (*bl* ∩ *blv*) .*bl* ∈# ($\mathcal{B}$ − {#*blv*#})#} ⟷ *p* ∈ *blv*
proof (*auto*, *simp add*: *assms finite-blocks*)
assume *assm*: *p* ∈ *blv*
then have ($\mathcal{B}$ − {#*blv*#}) *rep p* > *0* **using** *bibd-point-occ-rep r-ge-two assms* **by** *auto*
then obtain *bl* **where** *bl* ∈# ($\mathcal{B}$ − {#*blv*#}) ∧ *p* ∈ *bl* **using** *assms rep-number-g0-exists* **by** *metis*
then show ∃ *x*∈#*remove1-mset blv* $\mathcal{B}$. *p* ∈# *mset-set* (*x* ∩ *blv*)
using *assms assm finite-blocks* **by** *auto*
qed

lemma *intersect-mult-set-block-subset-iff*:
assumes *blv* ∈# $\mathcal{B}$
assumes *p* ∈# $\sum_{\#}${# *mset-set* {*y* .*y* ⊆ *blv* ∩ *b2* ∧ *card y* = *2*} .*b2* ∈# ($\mathcal{B}$ − {#*blv*#})#}
shows *p* ⊆ *blv*
proof (*rule subsetI*)

fix x
assume *asm*: $x \in p$
obtain $b2$ **where** $p \in\# \mathit{mset\text{-}set}\ \{y\ .\ y \subseteq \mathit{blv} \cap b2 \wedge \mathit{card}\ y = 2\} \wedge b2 \in\#(\mathcal{B} - \{\#\mathit{blv}\#\})$
using *assms* **by** *blast*
then have $p \subseteq \mathit{blv} \cap b2$
by (*metis* (*no-types*, *lifting*) *elem-mset-set equals0D infinite-set-mset-mset-set mem-Collect-eq*)
thus $x \in \mathit{blv}$ **using** *asm* **by** *auto*
qed

lemma *intersect-mult-set-block-subset-card*:
assumes $\mathit{blv} \in\# \mathcal{B}$
assumes $p \in\# \sum_{\#}\{\#\ \mathit{mset\text{-}set}\ \{y\ .y \subseteq \mathit{blv} \cap b2 \wedge \mathit{card}\ y = 2\}\ .b2 \in\#\ (\mathcal{B} - \{\#\mathit{blv}\#\})\#\}$
shows $\mathit{card}\ p = 2$
proof −
obtain $b2$ **where** $p \in\# \mathit{mset\text{-}set}\ \{y\ .\ y \subseteq \mathit{blv} \cap b2 \wedge \mathit{card}\ y = 2\} \wedge b2 \in\#(\mathcal{B} - \{\#\mathit{blv}\#\})$
using *assms* **by** *blast*
thus *?thesis*
by (*metis* (*mono-tags*, *lifting*) *elem-mset-set equals0D infinite-set-mset-mset-set mem-Collect-eq*)
qed

lemma *intersect-mult-set-block-with-point-exists*:
assumes $\mathit{blv} \in\# \mathcal{B}$ **and** $p \subseteq \mathit{blv}$ **and** $\Lambda \geq 2$ **and** $\mathit{card}\ p = 2$
shows $\exists x \in\# \mathit{remove1\text{-}mset}\ \mathit{blv}\ \mathcal{B}.\ p \in\# \mathit{mset\text{-}set}\ \{y.\ y \subseteq \mathit{blv} \wedge y \subseteq x \wedge \mathit{card}\ y = 2\}$
proof −
have $\mathit{size}\ \{\#b \in\#\ \mathcal{B}\ .\ p \subseteq b\#\} = \Lambda$ **using** *points-index-def assms*
by (*metis balanced-alt-def-all dual-order.trans wellformed*)
then have $\mathit{size}\ \{\#\mathit{bl} \in\#\ (\mathcal{B} - \{\#\mathit{blv}\#\})\ .\ p \subseteq \mathit{bl}\#\} \geq 1$
using *assms* **by** (*simp add*: *size-Diff-singleton*)
then obtain bl **where** $\mathit{bl} \in\#\ (\mathcal{B} - \{\#\mathit{blv}\#\}) \wedge p \subseteq \mathit{bl}$ **using** *assms filter-mset-empty-conv*
by (*metis diff-diff-cancel diff-is-0-eq′ le-numeral-extra*(*4*) *size-empty zero-neq-one*)

thus *?thesis*
using *assms finite-blocks* **by** *auto*
qed

lemma *intersect-mult-set-block-subset-iff-2*:
assumes $\mathit{blv} \in\# \mathcal{B}$ **and** $p \subseteq \mathit{blv}$ **and** $\Lambda \geq 2$ **and** $\mathit{card}\ p = 2$
shows $p \in\# \sum_{\#}\{\#\ \mathit{mset\text{-}set}\ \{y\ .y \subseteq \mathit{blv} \cap b2 \wedge \mathit{card}\ y = 2\}\ .b2 \in\#\ (\mathcal{B} - \{\#\mathit{blv}\#\})\#\}$
by (*auto simp add*: *intersect-mult-set-block-with-point-exists assms*)

lemma *sym-sum-mset-inter-sets-count*:

assumes *blv* $\in\#$ $\mathcal{B}$
assumes $p \in blv$
shows *count* $(\sum_{\#}\{\#$ *mset-set* $(bl \cap blv)$ *.bl* $\in\#$ $(\mathcal{B} - \{\#blv\#\})\#\})$ $p = \mathrm{r} - 1$
(**is** *count* $(\sum_{\#} ?M)$ $p = \mathrm{r} - 1$)
proof −
have *size-inter*: *size* $\{\#$ *mset-set* $(bl \cap blv) \mid bl \in\#$ $(\mathcal{B} - \{\#blv\#\})$. $p \in bl\#\}$ $= \mathrm{r} - 1$
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* $\in\#$ $(\mathcal{B} - \{\#blv\#\}) \Longrightarrow p \in bl \longrightarrow$ *count* (*mset-set* $(bl \cap blv))$ p $= 1$
using *assms count-mset-set*(1) *inter-finite* **by** *simp*
then have $\bigwedge$ *bl* . *bl* $\in\#$ $\{\#b1 \in\#(\mathcal{B} - \{\#blv\#\})$. $p \in b1\#\} \Longrightarrow$ *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 $\in bl\#\}$
$\Longrightarrow$ *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* $\in\#$ $(\mathcal{B} - \{\#blv\#\})\#\}) = \mathrm{k} * (\mathrm{r} - 1)$
(**is** *size* $(\sum_{\#} ?M) = \mathrm{k}*$ $(\mathrm{r} - 1)$)
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* $(\sum_{\#} ?M)) = \mathrm{k}$
by (*simp add*: *assms*)
have $\bigwedge$ *p*. *p* $\in\#$ $(\sum_{\#} ?M) \Longrightarrow$ *count* $(\sum_{\#} ?M)$ $p = \mathrm{r} - 1$
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* $|\cap|$ *b2*$) = \mathrm{k}*$ $(\mathrm{r} - 1)$
proof −
have $(\sum b2 \in\#(\mathcal{B} - \{\#b1\#\}).$ *b1* $|\cap|$ *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* ($\sum_\#$ *?M*) = (k *choose 2*) $*$ (Λ -1))
proof (*cases* Λ = *1*)
case *True*
have *empty*: $\bigwedge$ *b2* . *b2* $\in\#$ *remove1-mset blv* $\mathcal{B} \Longrightarrow \{y$.*y* $\subseteq$ *blv* $\wedge$ *y* $\subseteq$ *b2* $\wedge$ *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*: $\Lambda \geq 2$ **using** *index-not-zero* **by** *linarith*
have *subset-card*: $\bigwedge$ *x* . *x* $\in\#$ ($\sum_\#$ *?M*) $\Longrightarrow$ *card x* = *2*
proof −
fix *x*
assume *a*: *x* $\in\#$ ($\sum_\#$ *?M*)
then obtain *b2* **where** *x* $\in\#$ *mset-set* $\{y$. *y* $\subseteq$ *blv* $\cap$ *b2* $\wedge$ *card y* = *2*$\}$ $\wedge$ *b2* $\in\#(\mathcal{B} - \{\#blv\#\})$
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* ($\sum_\#$ *?M*) = $\{bl$. *bl* $\subseteq$ *blv* $\wedge$ *card bl* = *2*$\}$
proof
show *set-mset* ($\sum_\#$ *?M*) $\subseteq$ $\{bl$. *bl* $\subseteq$ *blv* $\wedge$ *card bl* = *2*$\}$
using *subset-card intersect-mult-set-block-subset-iff assms* **by** *blast*
show $\{bl$. *bl* $\subseteq$ *blv* $\wedge$ *card bl* = *2*$\}$ $\subseteq$ *set-mset* ($\sum_\#$ *?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* ($\sum_\#$ *?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* $\in\#$ $\mathcal{B}$
shows ($\sum$ *b2* $\in\#$ ($\mathcal{B}$ − $\{\#b1\#\}$). ((*b1* $|\cap|$ *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 ($\sum$ *b2* $\in\#$ ($\mathcal{B}$ − $\{\#b1\#\}$). ((*b1* $|\cap|$ *b2*) *choose 2*)) = ($\sum$ *b2* $\in\#$ ($\mathcal{B}$ − $\{\#b1\#\}$). (*b1* $|\cap|_2$ *b2*))
using *n-inter-num-choose-design-inter assms* **by** (*simp add*: *in-diffD*)
then have *sum-fact*: ($\sum$ *b2* $\in\#$ ($\mathcal{B}$ − $\{\#b1\#\}$).((*b1* $|\cap|$ *b2*) *choose 2*))

```
    = (k choose 2) * (Λ -1)
   using assms sym-sum-mset-inter2-sets-size
   by (auto simp add: size-big-union-sum n-intersect-num-subset-def)
 have (k choose 2) * (Λ -1) = ((Λ * (v - 1) div 2)) * (Λ -1)
   using choose-two symmetric-condition-2 k-non-zero by auto
 also have ... = ((Λ * (Λ - 1) 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 choose 2) * (Λ -1) = ((Λ * (Λ - 1) div 2)) * (v -1) .
 then show ?thesis using sum-fact by simp
qed

lemma sym-sum-inter-num-sq:
 assumes b1 ∈# B
 shows (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)^2) = Λ^2 * ( v - 1)
proof -
 have dvd: 2 dvd (( v - 1) * (Λ * (Λ - 1))) by fastforce
 have inner-dvd: ∀ bl ∈# (remove1-mset b1 B). 2 dvd ((b1 |∩| bl) * ((b1 |∩| bl)
- 1))
   by force
 have diff-le: ⋀ bl . bl ∈# (remove1-mset b1 B) ⟹ (b1 |∩| bl) ≤ (b1 |∩| bl)^2
   by (simp add: power2-nat-le-imp-le)
 have a: (∑ b2 ∈#(B - {#b1#}). ((b1 |∩| b2) choose 2)) =
       (∑ bl ∈# (remove1-mset b1 B). ((b1 |∩| bl) * ((b1 |∩| bl) - 1)) div 2)
   using choose-two by (simp add: intersection-num-non-neg)
 have b: (∑ b2 ∈#(B - {#b1#}). ((b1 |∩| b2) choose 2)) =
           (∑ b2 ∈#(B - {#b1#}). ((b1 |∩| b2) choose 2)) by simp
 have gtsq: (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl)^2) ≥ (∑ bl ∈# (remove1-mset
b1 B). (b1 |∩| bl))
   by (simp add: diff-le sum-mset-mono)
 have (∑ b2 ∈#(B - {#b1#}). ((b1 |∩| b2) choose 2)) = ((Λ * (Λ - 1)) div
2) * ( v - 1)
   using sum-choose-two-inter-num assms by blast
 then have start: (∑ bl ∈# (remove1-mset b1 B). ((b1 |∩| bl) * ((b1 |∩| bl) -
1)) div 2)
                    = ((Λ * (Λ - 1)) div 2) * (v - 1)
   using a b by linarith
 have sum-dvd: 2 dvd (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl) * ((b1 |∩| 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 (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl) * ((b1 |∩| bl) - 1)) div
2
     = ((Λ * (Λ - 1)) div 2) * (v - 1)
   using start sum-mset-distrib-div-if-dvd inner-dvd
   by (metis (mono-tags, lifting) image-mset-cong2)
 then have (∑ bl ∈# (remove1-mset b1 B). (b1 |∩| bl) * ((b1 |∩| bl) - 1)) div
```

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*)^*2* − (*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*)^*2*)
    − (∑ *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*)^*2*]
    diff-le **by** *presburger*
 then have (∑ *bl* ∈# (*remove1-mset b1* ℬ). (*b1* |∩| *bl*)^*2*)
    = (∑ *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*)^*2*) = (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*)^*2*) = (Λ ∗ (v − *1*)) ∗ (Λ − *1*) + (Λ ∗ (v − *1*))
   by *fastforce*
 then have (∑ *bl* ∈# (*remove1-mset b1* ℬ). (*b1* |∩| *bl*)^*2*) = (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*)^*2*) = 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))^*2*) = *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*))^*2*)) =
   (∑ *bl* ∈# (*remove1-mset b1* ℬ). (((*b1* |∩| *bl*))^*2*)) **by** *simp*
 then have *ssi*: (∑ *bl* ∈# (*remove1-mset b1* ℬ). ((*int* (*b1* |∩| *bl*))^*2*)) = Λ^*2* ∗ (v − *1*)
   using *sym-sum-inter-num-sq assms* **by** *simp*

have $\bigwedge bl$. $bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}) \Longrightarrow (int\ (b1\ |\cap|\ bl) - (int\ \Lambda))\hat{}2 =$
$(((int\ (b1\ |\cap|\ bl))\hat{}2) - (2 * (int\ \Lambda) * (int\ (b1\ |\cap|\ bl))) + ((int\ \Lambda)\hat{}2))$
by (*simp add*: *power2-diff*)
then have $(\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (int\ (b1\ |\cap|\ bl) - (int\ \Lambda))\hat{}2) =$
$(\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (((int\ (b1\ |\cap|\ bl))\hat{}2) - (2 * (int\ \Lambda) * (int\ (b1\ |\cap|\ bl))) + ((int\ \Lambda)\hat{}2)))$
using *sum-over-fun-eq* **by** *auto*
also have $... = (\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (((int\ (b1\ |\cap|\ bl))\hat{}2) - (2 * (int\ \Lambda) * (int\ (b1\ |\cap|\ bl))))) + (\sum\ bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B})\ .\ ((int\ \Lambda)\hat{}2))$
by (*simp add*: *sum-mset.distrib*)
also have $... = (\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ ((int\ (b1\ |\cap|\ bl))\hat{}2)) - (\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (2 * (int\ \Lambda) * (int\ (b1\ |\cap|\ bl)))) + (\sum\ bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B})\ .\ ((int\ \Lambda)\hat{}2))$
using *sum-mset-add-diff-int*[*of* $(\lambda\ bl\ .\ ((int\ (b1\ |\cap|\ bl))\hat{}2))\ (\lambda\ bl\ .\ (2 * (int\ \Lambda) * (int\ (b1\ |\cap|\ bl))))\ (remove1\text{-}mset\ b1\ \mathcal{B})$]
by *simp*
also have $... = \Lambda\hat{}2 * (\mathrm{v} - 1) - 2 * (int\ \Lambda) *(\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ ((b1\ |\cap|\ bl))) + (\mathrm{v} - 1) * ((int\ \Lambda)\hat{}2)$ **using** *ssi rm1-size assms* **by** (*simp add*: *sum-mset-distrib-left*)
also have $... = 2 * \Lambda\hat{}2 * (\mathrm{v} - 1) - 2 * (int\ \Lambda) *(\mathrm{k} * (\mathrm{r} - 1))$
using *sym-sum-inter-num assms* **by** *simp*
also have $... = 2 * \Lambda\hat{}2 * (\mathrm{v} - 1) - 2 * (int\ \Lambda) * (\Lambda * (\mathrm{v} - 1))$
using *rep-value-sym symmetric-condition-2* **by** *simp*
finally have $(\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (int\ (b1\ |\cap|\ bl) - int\ \Lambda)\hat{}2) = 0$
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*: $l = int\ \Lambda$
then have *pos*: $\bigwedge bl\ .\ (int\ (b1\ |\cap|\ bl) - l)\hat{}2 \geq 0$ **by** *simp*
have $(\sum bl \in\# (remove1\text{-}mset\ b1\ \mathcal{B}).\ (int\ (b1\ |\cap|\ bl) - l)\hat{}2) = 0$
using *sym-sum-inter-num-to-zero assms l-def* **by** *simp*
then have $\bigwedge bl.\ \ bl \in set\text{-}mset\ (remove1\text{-}mset\ b1\ \mathcal{B}) \Longrightarrow (int\ (b1\ |\cap|\ bl) - l)\hat{}2 = 0$
using *sum-mset-0-iff-ge-0 pos* **by** (*metis* (*no-types*, *lifting*))
then have $\bigwedge bl.\ \ bl \in set\text{-}mset\ (remove1\text{-}mset\ b1\ \mathcal{B}) \Longrightarrow int\ (b1\ |\cap|\ bl) = l$
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 ¬ (*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 *blleft*: *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 *keql*: k = Λ **using** *sym-block-intersections-index blleft assms* **by** *simp*
then have v = k
using *keql index-lt-replication rep-value-sym block-size-lt-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* = Λ
proof −
obtain *bl2* **where** *set*: *bl2* $\in\#$ *remove1-mset bl* $\mathcal{B}$ **and** *inter*: *b* = *bl2* ∩ *bl*
using *derived-blocks-def assms* **by** (*meson derived-obtain-orig-block*)
then have *card b* = *bl2* $|\cap|$ *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* = Λ − *1*
proof −

have *b-in*: $\bigwedge b$. $b \in\# (remove1\text{-}mset\ bl\ \mathcal{B}) \Longrightarrow ps \subseteq b \Longrightarrow ps \subseteq b \cap bl$
using *assms* **by** *blast*
then have *orig*: $ps \subseteq \mathcal{V}$
using *valid-block assms wellformed* **by** *blast*
then have *lam*: $size\ \{\#\ b \in\#\ \mathcal{B}\ .\ ps \subseteq b\ \#\} = \Lambda$ **using** *balanced*
by (*simp add*: *assms*(*2*) *points-index-def*)
then have $size\ \{\#\ b \in\#\ remove1\text{-}mset\ bl\ \mathcal{B}\ .\ ps \subseteq b\ \#\} = size\ \{\#\ b \in\#\ \mathcal{B}\ .\ ps \subseteq b\ \#\} - 1$
using *assms valid-block* **by** (*simp add*: *size-Diff-submset*)
then have $size\ \{\#\ b \in\#\ remove1\text{-}mset\ bl\ \mathcal{B}\ .\ ps \subseteq b\ \#\} = \Lambda - 1$
using *lam index-not-zero* **by** *linarith*
then have $size\ \{\#\ bl \cap b\ |\ \ b \in\#\ (remove1\text{-}mset\ bl\ \mathcal{B})\ .\ ps \subseteq bl \cap b\ \#\} = \Lambda - 1$
using *b-in* **by** (*metis* (*no-types*, *lifting*) *Int-subset-iff filter-mset-cong size-image-mset*)
then have $size\ \{\#\ x \in\#\ \{\#\ bl \cap b\ .\ b \in\#\ (remove1\text{-}mset\ bl\ \mathcal{B})\ \#\}\ .\ ps \subseteq x\ \#\} = \Lambda - 1$
by (*metis image-mset-filter-swap*)
then have $size\ \{\#\ x \in\#\ \mathcal{B}^D\ .\ ps \subseteq x\ \#\} = \Lambda - 1$ **by** (*simp add*: *derived-blocks-def*)
thus *?thesis* **by** (*simp add*: *points-index-def*)
qed

lemma *sym-derive-design-bibd*:
assumes $\Lambda > 1$
shows *bibd 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 $\Lambda < \mathrm{k}$ **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 $card\ b = \mathrm{k} - \Lambda$
proof −
obtain *bl2* **where** *sub*: $b = bl2 - bl$ **and** *mem*: $bl2 \in\#\ remove1\text{-}mset\ bl\ \mathcal{B}$
using *assms residual-blocks-def* **by** *auto*
then have $card\ b = card\ bl2 - card\ (bl2 \cap bl)$
using *card-Diff-subset-Int valid-block finite-blocks*
by (*simp add*: *card-Diff-subset-Int*)
then have $card\ b = card\ bl2 - bl2\ |\cap|\ bl$
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*: $\bigwedge b$. $(b \in\# \textit{remove1-mset } bl\ \mathcal{B} \Longrightarrow ps \subseteq b \Longrightarrow ps \subseteq (b - bl))$ **using** *assms*
by (*meson DiffI block-complement-elem-iff block-complement-subset-points subsetD subsetI*)
have *b*: $\bigwedge b$. $(b \in\# \textit{remove1-mset } bl\ \mathcal{B} \Longrightarrow ps \subseteq (b - bl) \Longrightarrow ps \subseteq b)$
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* $\{\# x \in\# \{\# b - bl . b \in\# (\textit{remove1-mset } bl\ \mathcal{B}) \#\} . ps \subseteq x \#\}$
using *assms valid-block* **by** (*simp add*: *points-index-def residual-blocks-def*)
also have ... = *size* $\{\# b - bl \mid b \in\# (\textit{remove1-mset } bl\ \mathcal{B})$. $ps \subseteq b - bl \#\}$
by (*metis image-mset-filter-swap*)
finally have $\mathcal{B}^R$ *index ps = size* $\{\# b \in\# (\textit{remove1-mset } bl\ \mathcal{B})$. $ps \subseteq b \#\}$ **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 $\mathrm{k} \geq \Lambda + 2$
shows *bibd* (bl^c) $\mathcal{B}^R$ $(\mathrm{k} - \Lambda)$ Λ
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 *bibd* $\subseteq$ *k-*Λ*-PBD* $\mathcal{V}$ $\mathcal{B}$ Λ k
using *block-size-gt-t* **by** (*unfold-locales*) *simp-all*

lemma *incomplete-PBD-is-bibd*:
assumes $k <$ *card V* **and** *k-*Λ*-PBD V B* Λ *k*
shows *bibd V B k* Λ
proof −

interpret *inc*: *incomplete-design V B k* **using** *assms*
by (*auto simp add*: *block-design.incomplete-designI k-Λ-PBD.axioms*(*2*))
interpret *pairwise-balance*: *pairwise-balance V B Λ* **using** *assms*
by (*auto simp add*: *k-Λ-PBD.axioms*(*1*))
show *?thesis* **using** *assms k-Λ-PBD.block-size-t* **by** (*unfold-locales*) (*simp-all*)
qed

lemma (**in** *bibd*) *bibd-to-pbdI*[*intro*]:
assumes $\Lambda = 1$
shows *k-PBD* $\mathcal{V}$ $\mathcal{B}$ k
proof −
interpret *pbd*: *k-Λ-PBD* $\mathcal{V}$ $\mathcal{B}$ Λ k
by (*simp add*: *k-Λ-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-Λ-PBD*

sublocale *incomplete-PBD* ⊆ *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* ⇒ *bool* **where**
resolution-class S ⟷ *partition-on* $\mathcal{V}$ *S* ∧ (∀ *bl* ∈ *S* . *bl* ∈# $\mathcal{B}$)

lemma *resolution-classI* [*intro*]: *partition-on* $\mathcal{V}$ *S* ⟹ (⋀ *bl* . *bl* ∈ *S* ⟹ *bl* ∈# $\mathcal{B}$)
⟹ *resolution-class S*
by (*simp add*: *resolution-class-def*)

lemma *resolution-classD1*: *resolution-class* $S \Longrightarrow$ *partition-on* $\mathcal{V}$ S
by (*simp add*: *resolution-class-def*)

lemma *resolution-classD2*: *resolution-class* $S \Longrightarrow$ $bl \in S \Longrightarrow bl \in\# \mathcal{B}$
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*: $\mathcal{V} \neq \{\} \Longrightarrow \mathcal{V} \in\# \mathcal{B} \Longrightarrow$ *resolution-class* $\{\mathcal{V}\}$
by (*auto simp add*: *resolution-class-def partition-on-space*)

lemma *resolution-class-union*: *resolution-class* $S \Longrightarrow \bigcup S = \mathcal{V}$
by (*simp add*: *resolution-class-def partition-on-def*)

lemma (**in** *finite-incidence-system*) *resolution-class-finite*: *resolution-class* $S \Longrightarrow$ *finite* S
using *finite-elements finite-sets* **by** (*auto simp add*: *resolution-class-def*)

lemma (**in** *design*) *resolution-class-sum-card*: *resolution-class* $S \Longrightarrow (\sum bl \in S$. *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* $\Rightarrow$ *bool* **where**
resolution $P \longleftrightarrow$ *partition-on-mset* $\mathcal{B}$ $P \wedge (\forall$ $S \in\#$ P . *distinct-mset* $S \wedge$ *resolution-class* (*set-mset* S))

lemma *resolutionI* : *partition-on-mset* $\mathcal{B}$ $P \Longrightarrow (\bigwedge$ S . $S \in\# P \Longrightarrow$ *distinct-mset* $S) \Longrightarrow$
$(\bigwedge$ S . $S \in\#$ $P \Longrightarrow$ *resolution-class* (*set-mset* S)) $\Longrightarrow$ *resolution* P
by (*simp add*: *resolution-def*)

lemma (**in** *proper-design*) *resolution-blocks*: *distinct-mset* $\mathcal{B} \Longrightarrow$ *disjoint* (*set-mset* $\mathcal{B}$) $\Longrightarrow$
$\bigcup$(*set-mset* $\mathcal{B}$) $= \mathcal{V} \Longrightarrow$ *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* (‹$\mathcal{P}$›)
assumes *resolvable*: *resolution* $\mathcal{P}$
begin

lemma *resolutionD1*: *partition-on-mset* $\mathcal{B}$ $\mathcal{P}$
using *resolvable* **by** (*simp add*: *resolution-def*)

lemma *resolutionD2*: $S \in\#\mathcal{P} \Longrightarrow$ *distinct-mset S*
using *resolvable* **by** (*simp add*: *resolution-def*)

lemma *resolutionD3*: $S\in\# \mathcal{P} \Longrightarrow$ *resolution-class* (*set-mset S*)
using *resolvable* **by** (*simp add*: *resolution-def*)

lemma *resolution-class-blocks-disjoint*: $S \in\# \mathcal{P} \Longrightarrow$ *disjoint* (*set-mset S*)
using *resolutionD3* **by** (*simp add*: *partition-on-def resolution-class-def*)

lemma *resolution-not-empty*: $\mathcal{B} \neq \{\#\} \Longrightarrow \mathcal{P} \neq \{\#\}$
using *partition-on-mset-not-empty resolutionD1* **by** *auto*

lemma *resolution-blocks-subset*: $S \in\# \mathcal{P} \Longrightarrow S \subseteq\# \mathcal{B}$
using *partition-on-mset-subsets resolutionD1* **by** *auto*

end

lemma (**in** *incidence-system*) *resolvable-designI* [*intro*]: *resolution* $\mathcal{P} \Longrightarrow$ *design* $\mathcal{V}$ $\mathcal{B} \Longrightarrow$
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*: $S \in\# \mathcal{P} \Longrightarrow bl \in\# S \Longrightarrow$ *card bl* = k
by (*metis resolutionD3 resolution-classD2 uniform-alt-def-all*)

lemma *resolution-class-size1*:
assumes $S \in\# \mathcal{P}$
shows v = k $*$ *size S*
proof −
have $(\sum bl \in\# S$. *card bl*$) = (\sum bl \in$ (*set-mset S*) . *card bl*) **using** *resolutionD2 assms*
by (*simp add*: *sum-unfold-sum-mset*)
then have *eqv*: $(\sum bl \in\# S$. *card bl*$)$ = v **using** *resolutionD3 assms resolution-class-sum-card*
by *presburger*
have $(\sum bl \in\# S$. *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 *size* S = v *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 *dvd* v
proof −
obtain S **where** *s-in*: $S \in\#\mathcal{P}$ **using** *resolution-not-empty design-blocks-nempty*
by *blast*
then have k ∗ *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*: *size* $\mathcal{P}$ = r
proof −
have *k-ne0*: k ≠ *0* **using** *k-non-zero* **by** *auto*
have *f1*: b = ($\sum S \in\# \mathcal{P}$. *size* S)
by (*metis partition-on-msetD1 resolutionD1 size-big-union-sum*)
then have b = ($\sum S \in\# \mathcal{P}$. v *div* k) **using** *resolution-class-size2 f1* **by** *auto*
then have *f2*: b = (*size* $\mathcal{P}$) ∗ (v *div* k) **by** *simp*
then have *size* $\mathcal{P}$ = b *div* (v *div* k)
using *b-non-zero* **by** *auto*
then have *size* $\mathcal{P}$ = (b ∗ k) *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 *dvd* b
proof −
have *f1*: b = ($\sum S \in\# \mathcal{P}$. *size* S)
by (*metis partition-on-msetD1 resolutionD1 size-big-union-sum*)
then have b = ($\sum S \in\# \mathcal{P}$. v *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

Boses 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 ≥ v + r − *1* ⟷ r ≥ k + Λ
proof −
from *necessary-condition-two v-non-zero* **have** *r*: ‹r = b ∗ k *div* v›
by (*metis div-mult-self1-is-m*)
define *k b v l r* **where** *intdefs*: *k* ≡ (*int* k) *b* ≡ *int* b *v* = *int* v *l* ≡ *int* Λ *r* ≡ *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*) *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*: ⋀ *x*. *l dvd* (*x* ∗ (*r* ∗ (*k* − *1*) + *l*))
by (*metis necess1-alt diff-add-cancel dvd-mult dvd-triv-left*)
have (*b* ≥ *v* + *r* − *1*) ⟷ ((v ∗ *r*) *div k* ≥ *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 ... ⟷ (((*v* ∗ *r*) − (*v* ∗ *k*)) *div k* ≥ *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*: ... ⟷ ((*v* ∗ (*r* − *k*)) *div k* ≥ (*r* − *1*))
using *int-distrib*(*3*) **by** (*simp add*: *mult.commute*)
also have *f2*: ... ⟷ ((*v* ∗ (*r* − *k*)) ≥ *k* ∗ (*r* − *1*))
using *k-non-zero kdvd intdefs* **by** *auto*
also have ... ⟷ ((((*r* ∗ (*k* − *1*) + *l*) *div l*) ∗ (*r* − *k*)) ≥ *k* ∗ (*r* − *1*))
using *v-eq* **by** *presburger*
also have ... ⟷ ((*r* − *k*) ∗ ((*r* ∗ (*k* − *1*) + *l*) *div l*) ≥ (*k* ∗ (*r* − *1*)))
by (*simp add*: *mult.commute*)
also have ... ⟷ (((*r* − *k*) ∗ (*r* ∗ (*k* − *1*) + *l*)) *div l* ≥ (*k* ∗ (*r* − *1*)))
using *div-mult-swap necessary-condition-one intdefs*
by (*metis diff-add-cancel dvd-triv-left necess1-alt*)
also have ... ⟷ (((*r* − *k*) ∗ (*r* ∗ (*k* − *1*) + *l*)) ≥ *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 ≥ v + r − *1* ⟷ r ≥ k + Λ
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 paramater 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* (‹$\mathcal{G}$›)
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*: *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*: $G1 \in \mathcal{G} \Longrightarrow G2 \in \mathcal{G} \Longrightarrow G1 \neq G2 \Longrightarrow$ *disjnt G1 G2*
using *group-partitions partition-onD2 pairwiseD* **by** *fastforce*

lemma *point-in-one-group*: $x \in G1 \Longrightarrow G1 \in \mathcal{G} \Longrightarrow G2 \in \mathcal{G} \Longrightarrow G1 \neq G2 \Longrightarrow x \notin G2$
using *groups-disjoint-pairwise* **by** (*simp add*: *disjnt-iff*)

lemma *point-has-unique-group*: $x \in \mathcal{V} \Longrightarrow \exists!G.\ x \in G \wedge G \in \mathcal{G}$
using *partition-on-partition-on-unique group-partitions*
by *fastforce*

lemma *rep-number-point-group-one*:
assumes $x \in \mathcal{V}$
shows *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*: $G \in \mathcal{G} \Longrightarrow x \in G \Longrightarrow x \in \mathcal{V}$
using *group-partitions partition-onD1* **by** *auto*

lemma *point-subset-in-group*: $G \in \mathcal{G} \Longrightarrow ps \subseteq G \Longrightarrow ps \subseteq \mathcal{V}$
using *point-in-group* **by** *auto*

lemma *group-subset-point-subset*: $G \in \mathcal{G} \Longrightarrow G' \subseteq G \Longrightarrow ps \subseteq G' \Longrightarrow ps \subseteq \mathcal{V}$
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*: $G \in \mathcal{G} \Longrightarrow$ *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 = $(\sum G \in \mathcal{G}.\ card\ G)$
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*: *card* $\mathcal{G} \geq$ *2* **using** *groups-size* **by** *simp*
then have $\bigwedge$ *G* . *G* $\in \mathcal{G} \Longrightarrow$ *G* $\neq \{\}$ **using** *partition-onD3 group-partitions* **by** *auto*
then have $\bigwedge$ *G* . *G* $\in \mathcal{G} \Longrightarrow$ *card G* $\geq$ *1*
using *group-elements-finite card-0-eq* **by** *fastforce*
then have $(\sum G \in \mathcal{G}.\ card\ G) \geq$ *2* **using** *assm*
using *sum-mono* **by** *force*
thus *?thesis* **using** *v-equals-sum-group-sizes*
by *linarith*
qed

lemma *min-group-size*: *G* $\in \mathcal{G} \Longrightarrow$ *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 *card G* < v
proof −
have $(\sum G' \in \mathcal{G}.\ card\ G')$ = v **using** *gdd-min-v v-equals-sum-group-sizes*
by *linarith*
then have *split-sum*: *card G* + $(\sum G' \in (\mathcal{G} - \{G\}).\ card\ G')$ = v **using** *assms sum.remove*
by (*metis groups-finite v-equals-sum-group-sizes*)
have *card* $(\mathcal{G} - \{G\}) \geq$ *1* **using** *groups-size*
by (*simp add*: *assms groups-finite*)
then obtain *G′* **where** *gin*: *G′* $\in (\mathcal{G} - \{G\})$
by (*meson elem-exists-non-empty-set less-le-trans less-numeral-extra*(*1*))
then have *card G′* $\geq$ *1* **using** *min-group-size* **by** *auto*
then have $(\sum G' \in (\mathcal{G} - \{G\}).\ card\ G') \geq$ *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: $g_1^{a_1} g 2^{a_2} ... g_n^{a_n}$

definition *group-sizes* :: *nat set* **where**
group-sizes $\equiv \{card\ G \mid G\ .\ G \in \mathcal{G}\}$

definition *groups-of-size* :: *nat* ⇒ *nat* **where**
groups-of-size g ≡ *card* $\{\ G \in \mathcal{G}\ .\ card\ G = g\ \}$

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-existance*: *x* ∈ *group-sizes* ⟹ ∃ *G*. *G* ∈ 𝒢 ∧ *card G* = *x*
unfolding *group-sizes-def* **by** *auto*

lemma *groups-of-size-zero*: *groups-of-size 0* = *0*
proof −
have *empty*: {*G* ∈ 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* ∈ 𝒢 . *card G* = *g*} = {} **using** *group-size-lt-v assms* **by** *fastforce*
thus *?thesis* **unfolding** *groups-of-size-def*
by (*simp add*: ‹{*G* ∈ 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* ∈ 𝒢 . *card G* = *g*} = *a*
unfolding *group-type-def groups-of-size-def* **by** *simp*

lemma *group-card-in-sizes*: *g* ∈ 𝒢 ⟹ *card g* ∈ *group-sizes*
unfolding *group-sizes-def* **by** *auto*

lemma *group-card-non-zero-groups-of-size-min*:
assumes *g* ∈ 𝒢
assumes *card g* = *a*
shows *groups-of-size a* ≥ *1*
proof −
have *g* ∈ {*G* ∈ 𝒢 . *card G* = *a*} **using** *assms* **by** *simp*
then have {*G* ∈ 𝒢 . *card G* = *a*} ≠ {} **by** *auto*
then have *card* {*G* ∈ 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 $a \in$ *group-sizes*
shows *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 *groups-of-size* $a \leq$ v
proof −
have $\{G \in \mathcal{G}$. *card* $G = a\} \subseteq \mathcal{G}$ **by** *simp*
then have *card* $\{G \in \mathcal{G}$. *card* $G = a\} \leq$ *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*: $g \in \mathcal{G} \Longrightarrow \exists\ x$. (*card* g, x) $\in$ *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*: *partition-on* $\mathcal{G}$ $\{\{\ G \in \mathcal{G}$. *card* $G = g\ \} \mid g$. $g \in$ *group-sizes*$\}$
proof (*intro partition-onI*, *auto*)
fix g
assume *a1*: $g \in$ *group-sizes*
assume $\forall x.\ x \in \mathcal{G} \longrightarrow$ *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 $\exists\, xa.\ (\exists\, g.\ xa = \{G \in \mathcal{G}.$ *card* $G = g\} \wedge g \in$ *group-sizes*$) \wedge x \in xa$
using *group-card-in-sizes* **by** *auto*
qed

lemma *group-size-partition-covers-points*: $\bigcup(\bigcup\{\{\ G \in \mathcal{G}$. *card* $G = g\ \} \mid g$. $g \in$ *group-sizes*$\}) = \mathcal{V}$
by (*metis* (*no-types*, *lifting*) *group-partitions partition-groups-on-size partition-onD1*)

lemma *groups-of-size-alt-def-count*: *groups-of-size* $g =$ *count* $\{\#$ *card* G . $G \in\#$ *mset-set* $\mathcal{G}$ $\#\}$ g
proof −
have *a*: *groups-of-size* $g =$ *card* $\{\ G \in \mathcal{G}$. *card* $G = g\ \}$ **unfolding** *groups-of-size-def*
by *simp*
then have *groups-of-size* $g =$ *size* $\{\#\ G \in\#$ (*mset-set* $\mathcal{G}$) . *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*:$\forall\ x \in X\ .\ P\ x \Longrightarrow$ *Set.filter P X* = *X*
by (*simp add*: *Collect-conj-eq Int-absorb2 Set.filter-def subsetI*)

lemma *uniform-groups-groups-of-size-m*: *groups-of-size* m = *card* $\mathcal{G}$
proof(*simp add*: *groups-of-size-def*)
have {$G \in \mathcal{G}$. *card G* = m} = $\mathcal{G}$ **using** *uniform-groups-alt set-filter-eq-P-forall*
by *auto*
thus *card* {$G \in \mathcal{G}$. *card G* = m} = *card* $\mathcal{G}$ **by** *simp*
qed

lemma *uniform-groups-of-size-not-m*: *x* ≠ m ⟹ *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 Λtimes if in diff groups, or 0 times if in the same group

locale *GDD* = *group-design* +
fixes *index* :: *int* (‹Λ›)
assumes *index-ge-1*: Λ ≥ *1*
assumes *index-together*: *G* ∈ $\mathcal{G}$ ⟹ *x* ∈ *G* ⟹ *y* ∈ *G* ⟹ *x* ≠ *y* ⟹ $\mathcal{B}$ *index* {*x*, *y*} = *0*
assumes *index-distinct*: *G1* ∈ $\mathcal{G}$ ⟹ *G2* ∈ $\mathcal{G}$ ⟹ *G1* ≠ *G2* ⟹ *x* ∈ *G1* ⟹ *y* ∈ *G2* ⟹
$\mathcal{B}$ *index* {*x*, *y*} = Λ
begin

lemma *points-sep-groups-ne*: *G1* ∈ $\mathcal{G}$ ⟹ *G2* ∈ $\mathcal{G}$ ⟹ *G1* ≠ *G2* ⟹ *x* ∈ *G1* ⟹ *y* ∈ *G2* ⟹ *x* ≠ *y*
by (*meson point-in-one-group*)

lemma *index-together-alt-ss*: *ps* ⊆ *G* ⟹ *G* ∈ $\mathcal{G}$ ⟹ *card ps* = *2* ⟹ $\mathcal{B}$ *index ps* = *0*
using *index-together* **by** (*metis card-2-iff insert-subset*)

lemma *index-distinct-alt-ss*: *ps* ⊆ $\mathcal{V}$ ⟹ *card ps* = *2* ⟹ (⋀ *G* . *G* ∈ $\mathcal{G}$ ⟹ ¬ *ps* ⊆ *G*) ⟹
$\mathcal{B}$ *index ps* = Λ
using *index-distinct* **by** (*metis card-2-iff empty-subsetI insert-subset point-has-unique-group*)

lemma *gdd-index-options*: *ps* ⊆ $\mathcal{V}$ ⟹ *card ps* = *2* ⟹ $\mathcal{B}$ *index ps* = *0* ∨ $\mathcal{B}$ *index ps* = Λ
using *index-distinct-alt-ss index-together-alt-ss* **by** *blast*

lemma *index-zero-implies-same-group*: $ps \subseteq \mathcal{V} \Longrightarrow$ *card* $ps = 2 \Longrightarrow \mathcal{B}$ *index* $ps = 0 \Longrightarrow$
$\exists\ G \in \mathcal{G}\ .\ ps \subseteq G$ **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*: $ps \subseteq \mathcal{V} \Longrightarrow$ *card* $ps = 2 \Longrightarrow \mathcal{B}$ *index* $ps = 0 \Longrightarrow$
$\exists!\ G \in \mathcal{G}\ .\ ps \subseteq G$
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*: $ps \subseteq \mathcal{V} \Longrightarrow$ *card* $ps = 2 \Longrightarrow \mathcal{B}$ *index* $ps = \Lambda \Longrightarrow$
$(\bigwedge G\ .\ G \in \mathcal{G} \Longrightarrow \neg\ ps \subseteq G)$
using *index-ge-1 index-together-alt-ss* **by** *auto*

lemma *index-zero-implies-one-group*:
assumes $ps \subseteq \mathcal{V}$
and *card* $ps = 2$
and $\mathcal{B}$ *index* $ps = 0$
shows *size* $\{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = 1$
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*: $\bigwedge G2\ .\ G2 \in \mathcal{G} \Longrightarrow G \neq G2 \Longrightarrow \neg\ ps \subseteq G2$
using *index-zero-implies-same-group-unique* **by** (*metis assms*)
have $\bigwedge G'.\ G' \in \mathcal{G} \longleftrightarrow G' \in\#\ mset\text{-}set\ \mathcal{G}$
by (*simp add*: *groups-finite*)
then have *eq-mset*: $\{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = mset\text{-}set\ \{b \in \mathcal{G}\ .\ ps \subseteq b\}$
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*: $ps \subseteq \mathcal{V} \Longrightarrow$ *card* $ps = 2 \Longrightarrow$
size $\{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = 0 \Longrightarrow \mathcal{B}$ *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 *card* $ps = 2$
and $\mathcal{B}$ *index* $ps = \Lambda$
shows *size* $\{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = 0$
proof −
have $\bigwedge G\ .\ G \in \mathcal{G} \Longrightarrow\ \neg\ ps \subseteq G$ **using** *index-not-zero-impl-diff-group assms*
by *simp*

then have $\{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = \{\#\}$
using *filter-mset-empty-if-finite-and-filter-set-empty* **by** *force*
thus *?thesis* **by** *simp*
qed

lemma *gdd-index-non-zero-iff*: $ps \subseteq \mathcal{V} \Longrightarrow card\ ps = 2 \Longrightarrow$
$\mathcal{B}\ index\ ps = \Lambda \longleftrightarrow size\ \{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = 0$
using *index-non-zero-implies-no-group index-distinct-group-num-alt-def* **by** *auto*

lemma *gdd-index-zero-iff*: $ps \subseteq \mathcal{V} \Longrightarrow card\ ps = 2 \Longrightarrow$
$\mathcal{B}\ index\ ps = 0 \longleftrightarrow size\ \{\#b \in\#\ \ mset\text{-}set\ \mathcal{G}\ .\ ps \subseteq b\#\} = 1$
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*: $ps \subseteq \mathcal{V} \Longrightarrow card\ ps = 2 \Longrightarrow \mathcal{B}\ index\ ps \leq \Lambda$
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 $card\ bl \geq 2$
shows $multiplicity\ bl = 1$
proof (*rule ccontr*)
assume $\neg$ ($multiplicity\ bl = 1$)
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*: $ps \subseteq bl \wedge card\ ps = 2$
using *nat-int-comparison*(*3*) *obtain-subset-with-card-n* **by** (*metis assms*(*3*))
then have $\mathcal{B}\ index\ ps \geq 2$
using *m points-index-count-min ps* **by** *blast*
then show *False* **using** *assms index-distinct ps antisym-conv2 not-numeral-less-zero*

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 $\bigwedge bl\ .\ card\ bl \geq 2$
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-Λ-GDD = K-block-design + GDD*

locale *k-Λ-GDD = block-design + GDD*

sublocale *k-Λ-GDD* ⊆ *K-Λ-GDD* $\mathcal{V}$ $\mathcal{B}$ {k} $\mathcal{G}$ Λ
by (*unfold-locales*)

locale *K-GDD = K-Λ-GDD* $\mathcal{V}$ $\mathcal{B}$ $\mathcal{K}$ $\mathcal{G}$ *1*
for *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *sizes* (‹$\mathcal{K}$›) **and** *groups* (‹$\mathcal{G}$›)

locale *k-GDD = k-Λ-GDD* $\mathcal{V}$ $\mathcal{B}$ k $\mathcal{G}$ *1*
for *point-set* (‹$\mathcal{V}$›) **and** *block-collection* (‹$\mathcal{B}$›) **and** *u-block-size* (‹k›) **and** *groups* (‹$\mathcal{G}$›)

sublocale *k-GDD* ⊆ *K-GDD* $\mathcal{V}$ $\mathcal{B}$ {k} $\mathcal{G}$
by (*unfold-locales*)

lemma (**in** *K-GDD*) *multiplicity-1*: *bl* ∈# $\mathcal{B}$ ⟹ *card bl* ≥ *2* ⟹ *multiplicity bl* = *1*
using *index-1-imp-mult-1* **by** *simp*

locale *RGDD = GDD + 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 *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 $\{y, z\} \subseteq$ (*del-point x*)
using ‹$y \neq x$› ‹$z \neq x$› *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 $\{y, z\} \subseteq$ *del-point x*
using ‹$y \neq x$› ‹$z \neq x$› *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 $\{x\} \in \mathcal{G} \Longrightarrow$ *card* $\mathcal{G} > 2$
shows $1 <$ *card* $\{g - \{x\} \mid g.\ g \in \mathcal{G} \wedge g \neq \{x\}\}$
proof (*cases* $\{x\} \in \mathcal{G}$)
case *True*
then have $\bigwedge g\ .\ g \in (\mathcal{G} - \{\{x\}\}) \Longrightarrow x \notin g$
by (*meson disjnt-insert1 groups-disjoint pairwise-alt*)
then have *simpg*: $\bigwedge g\ .\ g \in (\mathcal{G} - \{\{x\}\}) \Longrightarrow g - \{x\} = g$
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** *simpg*
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 *card* $(\mathcal{G} - \{\{x\}\}) =$ *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 $\bigwedge g'\ y.\ \{x\} \notin \mathcal{G} \Longrightarrow g' \in \mathcal{G} \Longrightarrow y \in \mathcal{G} \Longrightarrow g' - \{x\} = y - \{x\} \Longrightarrow g' = y$
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 *card* $\{g - \{x\} \mid g.\ g \in \mathcal{G} \wedge g \neq \{x\}\} =$ *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 $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow x \in bl \Longrightarrow 2 \leq$ *card bl*
assumes $\{x\} \in \mathcal{G} \Longrightarrow$ *card* $\mathcal{G} > 2$
shows *GDD* (*del-point x*) (*del-point-blocks x*) $\{g - \{x\} \mid g\ .\ g \in \mathcal{G} \wedge g \neq \{x\}\}\ \Lambda$
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 *card ps* = *2*
shows $(\mathcal{B} +$ *mset-set* $\mathcal{G})$ *index ps* = *1*
proof −
have *index1*: $(\bigwedge\ G\ .\ G \in \mathcal{G} \Longrightarrow \neg\ ps \subseteq G) \Longrightarrow \mathcal{B}$ *index ps* = *1*
using *index-distinct-alt-ss assms* **by** *fastforce*

have *groups1*: $\mathcal{B}$ *index ps* = *0* $\Longrightarrow$ *size* {#*b* $\in$# *mset-set* $\mathcal{G}$. *ps* $\subseteq$ *b*#} = *1*
using *index-zero-implies-one-group assms* **by** *simp*
then have ($\mathcal{B}$ + *mset-set* $\mathcal{G}$) *index ps* = *size* (*filter-mset* ((⊆) *ps*) ($\mathcal{B}$ + *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* $\mathcal{V}$ ($\mathcal{B}$ + *mset-set* $\mathcal{G}$) *1*
proof −
let *?B* = $\mathcal{B}$ + *mset-set* $\mathcal{G}$
have *ss*: $\bigwedge$ *G*. *G* $\in$ $\mathcal{G}$ $\Longrightarrow$ *G* $\subseteq$ $\mathcal{V}$
by (*simp add*: *point-in-group subsetI*)
have $\bigwedge$ *G*. *G* $\in$ $\mathcal{G}$ $\Longrightarrow$ *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* $\in$ $\mathcal{G}$ $\Longrightarrow$ *G* $\neq$ {}) $\Longrightarrow$ *b* $\in$# $\mathcal{B}$ + *mset-set* $\mathcal{G}$ $\Longrightarrow$ *b* $\subseteq$ $\mathcal{V}$
by (*metis finite-set-mset-mset-set groups-finite ss union-iff wellformed*)
show ($\bigwedge$*G*. *G* $\in$ $\mathcal{G}$ $\Longrightarrow$ *G* $\neq$ {}) $\Longrightarrow$ *finite* $\mathcal{V}$ **by** (*simp add*: *finite-sets*)
show $\bigwedge$*bl*. ($\bigwedge$*G*. *G* $\in$ $\mathcal{G}$ $\Longrightarrow$ *G* $\neq$ {}) $\Longrightarrow$ *bl* $\in$# $\mathcal{B}$ + *mset-set* $\mathcal{G}$ $\Longrightarrow$ *bl* $\neq$ {}
using *blocks-nempty groups-finite* **by** *auto*
qed
show *?thesis* **proof** (*unfold-locales*)
show *inc*.b $\neq$ *0* **using** *b-positive* **by** *auto*
show (*1* ::*nat*) $\leq$ *2* **by** *simp*
show *2* $\leq$ *inc*.v **by** (*simp add*: *gdd-min-v*)
then show $\bigwedge$*ps*. *ps* $\subseteq$ $\mathcal{V}$ $\Longrightarrow$ *card ps* = *2* $\Longrightarrow$ ($\mathcal{B}$ + *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* $\in$ $\mathcal{G}$ $\Longrightarrow$ *card G* $\in$ $\mathcal{K}$
assumes $\bigwedge$ *k* . *k* $\in$ $\mathcal{K}$ $\Longrightarrow$ *k* $\geq$ *2*
shows *PBD* $\mathcal{V}$ ($\mathcal{B}$ + *mset-set* $\mathcal{G}$) $\mathcal{K}$
proof −
let *?B* = $\mathcal{B}$ + *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 $ps \subseteq insert\ x\ \mathcal{V}$
assumes $card\ ps = 2$
shows $(\mathcal{B} + mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ index\ ps = 1$
proof −
have *inj-on* $((insert)\ x)\ \mathcal{G}$
by (*meson assms*(*1*) *inj-onI insert-ident point-in-group*)
then have *eq*: $mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\} = \{\#\ insert\ x\ g\ .\ g \in\#\ mset\text{-}set\ \mathcal{G}\#\}$
by (*simp add*: *image-mset-mset-set setcompr-eq-image*)
thus *?thesis*
proof (*cases* $x \in ps$)
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*: $\bigwedge g.\ g \in\#\ (mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\}) \Longrightarrow x \in g$
using *eq* **by** *force*
have *iff*: $\bigwedge g\ .\ g \in \mathcal{G} \Longrightarrow y \in (insert\ x\ g) \longleftrightarrow y \in g$ **using** *ynex* **by** *simp*
have *b*: $\mathcal{B}\ index\ ps = 0$
using *True assms*(*1*) *points-index-ps-nin* **by** *fastforce*
then have $(\mathcal{B} + mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ index\ ps =$
$(mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ index\ ps$
using *eq* **by** (*simp add*: *point-index-distrib*)
also have $... = (mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ rep\ y$ **using** *points-index-pair-rep-num*
by (*metis* (*no-types*, *lifting*) *all-g y-ps*)
also have *0*: $... = card\ \{b \in \{insert\ x\ g\ |g.\ g \in \mathcal{G}\}\ .\ y \in b\}$
by (*simp add*: *groups-finite rep-number-on-set-def*)
also have *1*: $... = card\ \{insert\ x\ g\ |g.\ g \in \mathcal{G} \wedge y \in insert\ x\ g\}$
by (*smt* (*verit*) *Collect-cong mem-Collect-eq*)
also have *2*: $... = card\ \{insert\ x\ g\ |g.\ g \in \mathcal{G} \wedge y \in g\}$
using *iff* **by** *metis*
also have $... = card\ \{g \in \mathcal{G}\ .\ y \in g\}$ **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 $(\mathcal{B} + mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ index\ ps = 1$
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 $(\mathcal{B} + mset\text{-}set\ \{insert\ x\ g\ |g.\ g \in \mathcal{G}\})\ index\ ps = (\mathcal{B} + mset\text{-}set\ \mathcal{G})\ index\ ps$
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*) ($\mathcal{B}$ + *mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}) *1*
proof −
let *?B* = $\mathcal{B}$ + *mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}
let *?V* = *add-point x*
have *vdef*: *?V* = $\mathcal{V}$ ∪ {*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 ⋀ *G*. *G* ∈ $\mathcal{G}$ ⟹ *insert x G* ⊆ *?V*
by (*simp add*: *point-in-group subsetI vdef*)
then have ⋀ *G*. *G* ∈# (*mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}) ⟹ *G* ⊆ *?V*
by (*smt* (*verit*, *del-insts*) *elem-mset-set empty-iff infinite-set-mset-mset-set mem-Collect-eq*)
then show ⋀*b*. *b* ∈# $\mathcal{B}$ ∨ *b* ∈# *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$} ⟹ *b* ⊆ *insert x* $\mathcal{V}$
using *wellformed add-point-def* **by** *fastforce*
next
have ⋀ *G*. *G* ∈ $\mathcal{G}$ ⟹ *insert x G* ≠ {} **using** *group-partitions*
using *partition-onD3* **by** *auto*
then have *gnempty*: ⋀ *G*. *G* ∈# (*mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}) ⟹ *G* ≠ {}
by (*smt* (*verit*, *del-insts*) *elem-mset-set empty-iff infinite-set-mset-mset-set mem-Collect-eq*)
then show ⋀*bl*. *bl* ∈# $\mathcal{B}$ ∨ *bl* ∈# *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$} ⟹ *bl* ≠ {}
using *blocks-nempty* **by** *auto*
next
have *card* $\mathcal{V}$ ≥ *2* **using** *gdd-min-v* **by** *simp*
then have *card* (*insert x* $\mathcal{V}$) ≥ *2*
by (*meson card-insert-le dual-order.trans finite-sets*)
then show *2* ≤ *card* (*insert x* $\mathcal{V}$) **by** *auto*
next
show ⋀*ps*. *ps* ⊆ *insert x* $\mathcal{V}$ ⟹
card ps = *2* ⟹ ($\mathcal{B}$ + *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$}) *index ps* = *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 ⋀ *k* . *k* ∈ $\mathcal{K}$ ⟹ *k* ≥ *2*
shows *PBD* (*add-point x*) ($\mathcal{B}$ + *mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}) ($\mathcal{K}$ ∪ {(*card g*) + *1* | *g* . *g* ∈ $\mathcal{G}$})
proof −

let *?B* = $\mathcal{B}$ + *mset-set* { *insert x g* | *g*. *g* ∈ $\mathcal{G}$}
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*: ⋀ *g*. *g* ∈ $\mathcal{G}$ ⟹ *x* ∉ *g*
using *assms point-in-group* **by** *auto*
have ⋀ *bl* . *bl* ∈# $\mathcal{B}$ ⟹ *card bl* ∈ $\mathcal{K}$ **by** (*simp add*: *block-sizes*)
have ⋀ *bl* . *bl* ∈# *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$} ⟹ *bl* ∈ {*insert x g* | *g* . *g*
∈ $\mathcal{G}$}
by (*simp add*: *groups-finite*)
then have ⋀ *bl* . *bl* ∈# *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$} ⟹
card bl ∈ {*card g* + *1* |*g*. *g* ∈ $\mathcal{G}$}
proof −
fix *bl*
assume *bl* ∈# *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$}
then have *bl* ∈ {*insert x g* | *g* . *g* ∈ $\mathcal{G}$} **by** (*simp add*: *groups-finite*)
then obtain *g* **where** *gin*: *g* ∈ $\mathcal{G}$ **and** *i*: *bl* = *insert x g* **by** *auto*
thus *card bl* ∈ {(*card g* + *1*) |*g*. *g* ∈ $\mathcal{G}$}
using *gin group-elements-finite i xg* **by** *auto*
qed
then show ⋀*bl*. *bl* ∈# $\mathcal{B}$ + *mset-set* {*insert x g* |*g*. *g* ∈ $\mathcal{G}$} ⟹
(*card bl*) ∈ $\mathcal{K}$ ∪ {(*card g* + *1*) |*g*. *g* ∈ $\mathcal{G}$}
using *UnI1 UnI2 block-sizes union-iff* **by** *auto*
show ⋀*x*. *x* ∈ $\mathcal{K}$ ∪ {*card g* + *1* |*g*. *g* ∈ $\mathcal{G}$} ⟹ *0* < *x*
using *min-group-size positive-ints* **by** *auto*
show ⋀*k*. *k* ∈ $\mathcal{K}$ ∪ {*card g* + *1* |*g*. *g* ∈ $\mathcal{G}$} ⟹ *2* ≤ *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 ⋀ *n*. *n* ∈ $\mathcal{K}'$ ⟹ *n* ≥ *2*
assumes ⋀ *B* . *B* ∈# $\mathcal{B}$ ⟹ *K-GDD* (*B* × *I*) (*f B*) $\mathcal{K}'$ {{*x*} × *I* |*x* . *x* ∈ *B* }
shows *proper-design* ($\mathcal{V}$ × *I*) (∑ *B* ∈# $\mathcal{B}$. (*f B*)) (**is** *proper-design ?Y ?B*)
proof (*unfold-locales*, *simp-all*)
show ⋀*b*. ∃ *x*∈#$\mathcal{B}$. *b* ∈# *f x* ⟹ *b* ⊆ $\mathcal{V}$ × *I*
proof −
fix *b*
assume ∃ *x*∈#$\mathcal{B}$. *b* ∈# *f x*
then obtain *B* **where** *B* ∈# $\mathcal{B}$ **and** *b* ∈# (*f B*) **by** *auto*
then interpret *kgdd*: *K-GDD* (*B* × *I*) (*f B*) $\mathcal{K}'$ {{*x*} × *I* |*x* . *x* ∈ *B* } **using**
assms **by** *auto*

```
    show b ⊆ 𝒱 × I using kgdd.wellformed
      using ‹B ∈# ℬ› ‹b ∈# f B› wellformed by fastforce
  qed
 show finite (𝒱 × I) using finite-sets assms bot-nat-0.not-eq-extremum card.infinite
by blast
  show ⋀bl. ∃x∈#ℬ. bl ∈# f x ⟹ bl ≠ {}
  proof −
    fix bl
    assume ∃x∈#ℬ. bl ∈# f x
    then obtain B where B ∈# ℬ and bl ∈# (f B) by auto
    then interpret kgdd: K-GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B } using
assms by auto
    show bl ≠ {} using kgdd.blocks-nempty by (simp add: ‹bl ∈# f B›)
  qed
  show ∃i∈#ℬ. f i ≠ {#}
  proof −
    obtain B where B ∈# ℬ
      using design-blocks-nempty by auto
    then interpret kgdd: K-GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B } using
assms by auto
    have f B ≠ {#} using kgdd.design-blocks-nempty by simp
    then show ∃i∈#ℬ. f i ≠ {#} using ‹B ∈# ℬ› by auto
  qed
qed

lemma pair-construction-block-sizes:
  assumes K-GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B }
  assumes B ∈# ℬ
  assumes b ∈# (f B)
  shows card b ∈ 𝒦'
proof −
  interpret bkgdd: K-GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B }
    using assms by simp
  show card b ∈ 𝒦' using bkgdd.block-sizes by (simp add:assms)
qed

lemma wilsons-construction-index-0:
  assumes ⋀ B . B ∈# ℬ ⟹ K-GDD (B × I) (f B) 𝒦' {{x} × I |x . x ∈ B }
  assumes G ∈ {GG × I |GG. GG ∈ 𝒢}
  assumes X ∈ G
  assumes Y ∈ G
  assumes X ≠ Y
  shows (∑# (image-mset f ℬ)) index {X, Y} = 0
proof −
  obtain G' where gi: G = G' × I and ging: G' ∈ 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 ∈ I and xing: x ∈ G' using assms gi by auto
  have iyin: iy ∈ I and ying: y ∈ G' using assms ypair gi by auto
```

have *ne-index-0*: $x \neq y \Longrightarrow \mathcal{B}$ *index* $\{x, y\} = 0$
using *ying xing index-together ging* **by** *simp*
have $\bigwedge B.\ B \in\# \mathcal{B} \Longrightarrow (f\ B)$ *index* $\{(x, ix), (y, iy)\} = 0$
proof −
fix B
assume *assm*: $B \in\# \mathcal{B}$
then interpret *kgdd*: *K-GDD* $(B \times I)$ $(f\ B)$ $\mathcal{K}'$ $\{\{x\} \times I \mid x\ .\ x \in B\}$ **using** *assms* **by** *simp*
have *not-ss-0*: $\neg\ (\{(x, ix), (y, iy)\} \subseteq (B \times I)) \Longrightarrow (f\ B)$ *index* $\{(x, ix), (y, iy)\} = 0$
by (*metis kgdd.points-index-ps-nin*)
have $x \neq y \Longrightarrow \neg\ \{x, y\} \subseteq B$ **using** *ne-index-0 assm points-index-0-left-imp* **by** *auto*
then have $x \neq y \Longrightarrow \neg\ (\{(x, ix), (y, iy)\} \subseteq (B \times I))$ **using** *assms*
by (*meson empty-subsetI insert-subset mem-Sigma-iff*)
then have *nexy*: $x \neq y \Longrightarrow (f\ B)$ *index* $\{(x, ix), (y, iy)\} = 0$ **using** *not-ss-0* **by** *simp*
have $x = y \Longrightarrow (\{(x, ix), (y, iy)\} \subseteq (B \times I)) \Longrightarrow (f\ B)$ *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** $(y, ix) \in g$
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 $(f\ B)$ *index* $\{(x, ix), (y, iy)\} = 0$ **using** *not-ss-0 nexy* **by** *auto*
qed
then have $\bigwedge B.\ B \in\# (\mathit{image\text{-}mset}\ f\ \mathcal{B}) \Longrightarrow B$ *index* $\{(x, ix), (y, iy)\} = 0$ **by** *auto*
then show $(\sum_{\#}\ (\mathit{image\text{-}mset}\ f\ \mathcal{B}))$ *index* $\{X, Y\} = 0$
by (*simp add*: *points-index-sum xpair ypair*)
qed

lemma *wilsons-construction-index-1*:
assumes $\bigwedge B\ .\ B \in\# \mathcal{B} \Longrightarrow$ *K-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 $(\sum_{\#}\ (\mathit{image\text{-}mset}\ f\ \mathcal{B}))$ *index* $\{(x, ix), (y, iy)\} = (1 :: int)$
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 $\bigwedge\ b\ .\ b \in\# \mathcal{B} - \{\#bl\#\} \Longrightarrow \neg\ \{x, y\} \subseteq b$ **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*: $\bigwedge\ b\ .\ b \in\# \mathcal{B} - \{\#bl\#\} \Longrightarrow \neg\ (\{(x, ix), (y, iy)\} \subseteq (b \times I))$ **using** *assms*
by (*meson SigmaD1 empty-subsetI insert-subset*)
then have *pi0*: $\bigwedge\ b\ .\ b \in\# \mathcal{B} - \{\#bl\#\} \Longrightarrow (f\ b)\ index\ \{(x, ix), (y, iy)\}\ = 0$
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* $(b \times I)\ (f\ b)\ \mathcal{K}'\ \{\{x\} \times I\ |x\ .\ x \in b\ \}$ **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* $= \{\{x\} \times I\ |x\ .\ x \in bl\ \}$
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 $(\sum_{\#}\ (image\text{-}mset\ f\ \mathcal{B}))\ index\ \{(x, ix), (y, iy)\} =$
$(\sum B \in\# \mathcal{B}.\ (f\ B)\ index\ \{(x, ix), (y, iy)\}\ )$
by (*simp add*: *points-index-sum*)
then have $(\sum_{\#}\ (image\text{-}mset\ f\ \mathcal{B}))\ index\ \{(x, ix), (y, iy)\} =$
$(\sum B \in\# (\mathcal{B} - \{\#bl\#\}).\ (f\ B)\ index\ \{(x, ix), (y, iy)\}) + (f\ bl)\ index\ \{(x, ix), (y, iy)\}$
by (*metis* (*no-types*, *lifting*) *add.commute blinb insert-DiffM sum-mset.insert*)
thus *?thesis* **using** *pi0 pi1* **by** *simp*
qed

theorem *Wilsons-Construction*:
assumes $card\ I = w$
assumes $w > 0$
assumes $\bigwedge\ n.\ n \in \mathcal{K}' \Longrightarrow n \geq 2$
assumes $\bigwedge\ B\ .\ B \in\# \mathcal{B} \Longrightarrow$ *K-GDD* $(B \times I)\ (f\ B)\ \mathcal{K}'\ \{\{x\} \times I\ |x\ .\ x \in B\ \}$
shows *K-GDD* $(\mathcal{V} \times I)\ (\sum B \in\# \mathcal{B}.\ (f\ B))\ \mathcal{K}'\ \{G \times I\ |\ G\ .\ G \in \mathcal{G}\}$

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* $\in\#$ ($\sum B \in\# \mathcal{B}.\ (f\ B)$) $\Longrightarrow$ *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* (λ *G*. *G* $\times$ *I*) $\mathcal{G}$
using *inj-on-def pd.design-points-nempty* **by** *auto*
then have *card* $\{G \times I \mid G.\ G \in \mathcal{G}\}$ = *card* $\mathcal{G}$ **using** *card-image* **by** (*simp add*: *Setcompr-eq-image*)
then show *1* < *card* $\{G \times I \mid G.\ G \in \mathcal{G}\}$ **using** *groups-size* **by** *linarith*
show (*1*::*int*) ≤ *1* **by** *simp*
have *gdd-fact*: $\bigwedge$ *B* . *B* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *K-GDD* (*B* × *I*) (*f B*) $\mathcal{K}'$ $\{\{x\} \times I \mid x\ .\ x \in B\ \}$
using *assms* **by** *simp*
show $\bigwedge G\ X\ Y.\ G \in \{GG \times I \mid GG.\ GG \in \mathcal{G}\} \Longrightarrow X \in G \Longrightarrow Y \in G \Longrightarrow X \neq Y$
$\Longrightarrow$ ($\sum_\#$ (*image-mset f* $\mathcal{B}$)) *index* $\{X,\ Y\}$ = *0*
using *wilsons-construction-index-0*[*OF assms*(*4*)] **by** *auto*
show $\bigwedge$*G1 G2 X Y*. *G1* $\in \{G \times I \mid G.\ G \in \mathcal{G}\} \Longrightarrow$ *G2* $\in \{G \times I \mid G.\ G \in \mathcal{G}\}$
$\Longrightarrow$ *G1* ≠ *G2* $\Longrightarrow$ *X* $\in$ *G1* $\Longrightarrow$ *Y* $\in$ *G2* $\Longrightarrow$ (($\sum_\#$ (*image-mset f* $\mathcal{B}$)) *index* $\{X,\ Y\}$) = (*1* ::*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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *card bl* ≥ *2*
shows *pairwise-balance* (*del-point x*) (*del-point-blocks x*) Λ
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 $2 < \mathrm{v} \Longrightarrow (\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow 2 \leq card\ bl) \Longrightarrow 2 \leq (card\ (\mathcal{V} - \{x\}))$
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 $\bigwedge ps\ .\ ps \subseteq \mathcal{V} - \{x\} \Longrightarrow ps \subseteq \mathcal{V}$ **by** *auto*
then show $\bigwedge ps.\ ps \subseteq \mathcal{V} - \{x\} \Longrightarrow card\ ps = 2 \Longrightarrow \{\#bl - \{x\}.\ bl \in\# \mathcal{B}\#\}$ *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 $\mathrm{k} > 1$
assumes $\bigwedge g.\ g \in \mathcal{G} \Longrightarrow card\ g = \mathrm{k} - 1$
assumes $x \notin \mathcal{V}$
shows *bibd* (*add-point x*) ($\mathcal{B}$ + *mset-set* { *insert x g* | *g*. $g \in \mathcal{G}$}) ({k}) *1*
proof −
have $\bigwedge k\ .\ k \in \{\mathrm{k}\} \Longrightarrow k = \mathrm{k}$ **by** *blast*
then have *kge*: $\bigwedge k\ .\ k \in \{\mathrm{k}\} \Longrightarrow k \geq 2$ **using** *assms*(*1*) **by** *simp*
have $\bigwedge g\ .\ g \in \mathcal{G} \Longrightarrow card\ g + 1 = \mathrm{k}$ **using** *assms k-non-zero* **by** *auto*
then have *s*: $(\{\mathrm{k}\} \cup \{(card\ g) + 1 \mid g\ .\ g \in \mathcal{G}\}) = \{\mathrm{k}\}$ **by** *auto*
then interpret *pbd*: *PBD* (*add-point x*) $\mathcal{B}$ + *mset-set* { *insert x g* | *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*: $ps \subseteq \mathcal{V} \Longrightarrow card\ ps = 2 \Longrightarrow \mathcal{B}\ index\ ps = 1$
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 $\neg\ y \notin b2$
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 $\mathcal{B}$ *index* $\{x,\ y\} = 1$ **using** *pbd-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 $G \in \{b - \{x\}\ |b.\ b \in\# \mathcal{B} \wedge x \in b\}$
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 pbd-index1-points-imply-unique-block*
by *fastforce*
qed

lemma *strong-delete-point-groups-index-one*:
assumes $G1 \in \{b - \{x\}\ |b.\ b \in\# \mathcal{B} \wedge x \in b\}$
assumes $G2 \in \{b - \{x\}\ |b.\ b \in\# \mathcal{B} \wedge x \in b\}$
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** *gb1*: $G1 = b1 - \{x\}$ **and** *b1in*: $b1 \in\# \mathcal{B}$ **and** *xin1*: $x \in b1$
using *assms* **by** *blast*
obtain *b2* **where** *gb2*: $G2 = b2 - \{x\}$ **and** *b2in*: $b2 \in\# \mathcal{B}$ **and** *xin2*:$x \in b2$
using *assms* **by** *blast*
have *bneq*: $b1 \neq b2$ **using** *assms*(3) *gb1 gb2* **by** *auto*
have $xa \neq y$ **using** *gb1 b1in xin1 gb2 b2in xin2 assms*(3) *assms*(4) *assms*(5) *insert-subset*
by (*smt* (*verit*, *best*) *Diff-eq-empty-iff Diff-iff empty-Diff insertCI pbd-index1-points-imply-unique-block*)

then have *pair*: *card* $\{xa,\ y\} = 2$ **by** *simp*
have *inv*: $\{xa,\ y\} \subseteq \mathcal{V}$ **using** *gb1 b1in gb2 b2in assms*(4) *assms*(5)
by (*metis Diff-cancel Diff-subset insert-Diff insert-subset wellformed*)
have $\{\#\ bl \in\# \mathcal{B}\ .\ x \in bl\#\}$ *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* $(\mathcal{V} - \{x\})$ $\{b - \{x\} \,|b.\ b \in\# \mathcal{B} \wedge x \in b\}$
proof (*intro partition-onI*)
have *gtt*: $\bigwedge$ *bl*. *bl* $\in\# \mathcal{B} \Longrightarrow$ *card bl* $\geq$ *2* **using** *block-size-gt-t*
using *block-sizes* **by** *blast*
show $\bigwedge p.\ p \in \{b - \{x\} \,|b.\ b \in\# \mathcal{B} \wedge x \in b\} \Longrightarrow p \neq \{\}$
proof −
fix p **assume** $p \in \{b - \{x\} \,|b.\ b \in\# \mathcal{B} \wedge x \in b\}$
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 $\bigcup \{b - \{x\} \,|b.\ b \in\# \mathcal{B} \wedge x \in b\} = \mathcal{V} - \{x\}$
proof (*intro subset-antisym subsetI*)
fix *xa*
assume $xa \in \bigcup \{b - \{x\} \,|b.\ b \in\# \mathcal{B} \wedge x \in b\}$
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 $\bigwedge p\ p'.\ p \in \{b - \{x\} \mid b.\ b \in\# \mathcal{B} \wedge x \in b\} \Longrightarrow p' \in \{b - \{x\} \mid b.\ b \in\# \mathcal{B} \wedge x \in b\} \Longrightarrow$
$p \neq p' \Longrightarrow p \cap p' = \{\}$
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 $\bigwedge y.\ y \in b \Longrightarrow y \neq x \Longrightarrow y \notin b'$
using *b1in b2in xinb1 xinb2 pbd-index1-points-imply-unique-block*
by (*meson empty-subsetI insert-subset*)
then have $\bigwedge y.\ y \in p \Longrightarrow y \notin p'$
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-GDD* (*del-point x*) (*str-del-point-blocks x*) $\mathcal{K}$ $\{\ b - \{x\} \mid b\ .\ b \in\# \mathcal{B} \wedge x \in b\}$
proof −
have $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow$ *card bl* ≥ 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*: *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 *card* $\{b - \{x\} \mid b.\ b \in\# \mathcal{B} \wedge x \in b\}$ = *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 *Suc 0* $<$ *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} | a . a ∈ A} = card A
  by (simp add: card-image Setcompr-eq-image)

lemma KGDD-from-PBD: K-GDD 𝒱 ℬ 𝒦 {{x} | x . x ∈ 𝒱}
proof (unfold-locales,auto simp add: Setcompr-eq-image partition-on-singletons)
  have card ((λx. {x}) ` 𝒱) ≥ 2 using t-lt-order card-singletons-eq
    by (metis Collect-mem-eq setcompr-eq-image)
  then show Suc 0 < card ((λx. {x}) ` 𝒱) by linarith
  show ⋀xa xb. xa ∈ 𝒱 ⟹ xb ∈ 𝒱 ⟹ ℬ index {xa, xb} ≠ Suc 0 ⟹ xa = xb
  proof (rule ccontr)
    fix xa xb
    assume ain: xa ∈ 𝒱 and bin: xb ∈ 𝒱 and ne1: ℬ index {xa, xb} ≠ Suc 0
    assume xa ≠ xb
    then have card {xa, xb} = 2 by auto
    then have ℬ 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 Λ = 1
  assumes x ∈ 𝒱
  shows k-GDD (del-point x) (str-del-point-blocks x) k { b − {x} | b . b ∈# ℬ ∧
x ∈ b}
proof −
  interpret pbd: PBD 𝒱 ℬ {k} using assms
    using PBD.intro Λ-PBD-axioms by auto
  have lt: ℬ rep x < b using block-num-gt-rep
    by (simp add: assms(2))
  have ℬ rep x > 1 using r-ge-two assms by simp
  then interpret kgdd: K-GDD (del-point x) str-del-point-blocks x
    {k} { b − {x} | b . b ∈# ℬ ∧ x ∈ 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-Componer*
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* ≠ {}

begin

lemma *verts-not-empty*: *verts G* ≠ {}
  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* ⇒ *'a set* **where**
  arc-to-block e ≡ {*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*} ⟹
  arc-to-ends G e = (*x*, *y*) ∨ *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*) ⟹ *arc-to-ends G e2* = (*v*, *u*) ⟹

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 ≡ *mset-set* (*arc-to-block* ‘ (*arcs G*))

lemma *arcs-blocks-ends*: (*x*, *y*) ∈ *arcs-ends G* ⟹ {*x*, *y*} ∈# *arcs-blocks*
proof (*auto simp add*: *arcs-ends-def arcs-blocks-def*)
fix *xa*
assume *assm1*: (*x*, *y*) = *arc-to-ends G xa* **and** *assm2*: *xa* ∈ *arcs G*
obtain *z* **where** *zin*: *z* ∈ (*arc-to-block* ‘ (*arcs G*)) **and** *z* = *arc-to-block xa*
using *assm2* **by** *blast*
thus {*x*, *y*} ∈ *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* ⊆ *arcs G* ⟹ (*x*, *y*) ∈ ((*arc-to-ends G*) ‘ *E*) ⟹

{*x*, *y*} ∈ (*arc-to-block* ‘ *E*)
by (*auto simp add*: *arc-to-ends-def arc-to-block-def*)

lemma *arc-blocks-end-subset*: **assumes** *E* ⊆ *arcs G* **and** {*x*, *y*} ∈ (*arc-to-block* ‘ *E*)
shows (*x*, *y*) ∈ ((*arc-to-ends G*) ‘ *E*) ∨ (*y*, *x*) ∈ ((*arc-to-ends G*) ‘ *E*)
proof −
obtain *e* **where** *e* ∈ *E* **and** *arc-to-block e* = {*x*,*y*} **using** *assms*
by *fastforce*
then have *arc-to-ends G e* = (*x*, *y*) ∨ *arc-to-ends G 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*} ∈# *arcs-blocks* ⟹ (*x*, *y*) ∈ *arcs-ends G* ∧ (*y*, *x*) ∈ *arcs-ends G*
proof (*auto simp add*: *arcs-ends-def arcs-blocks-def*)
fix *xa*
assume *assm1*: {*x*, *y*} = *arc-to-block xa* **and** *assm2*: *xa* ∈ (*arcs G*)
obtain *z* **where** *zin*: *z* ∈ (*arc-to-ends G* ‘ (*arcs G*)) **and** *z* = *arc-to-ends G xa*
using *assm2* **by** *blast*
then have *z* = (*x*, *y*) ∨ *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*) ∈ *arc-to-ends G* ‘ (*arcs G*) **using** *assm2*
by (*metis arcs-ends-def arcs-ends-symmetric sym-arcs zin*)
next
fix *xa*
assume *assm1*: {*x*, *y*} = *arc-to-block xa* **and** *assm2*: *xa* ∈ (*arcs G*)
thus (*y*, *x*) ∈ *arc-to-ends G* ‘ *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*: $\{x, y\} \in\# \textit{arcs-blocks} \longleftrightarrow (x, y) \in \textit{arcs-ends}\ G \wedge (y, x) \in \textit{arcs-ends}\ G$
using *arcs-ends-blocks arcs-blocks-ends* **by** *blast*

lemma *arcs-ends-wf*: $(x, y) \in \textit{arcs-ends}\ G \Longrightarrow x \in \textit{verts}\ G \wedge y \in \textit{verts}\ G$
by *auto*

lemma *arcs-blocks-elem*: $bl \in\# \textit{arcs-blocks} \Longrightarrow \exists\ x\ y\ .\ bl = \{x, y\}$
apply (*auto simp add*: *arcs-blocks-def*)
by (*meson arc-to-block-def*)

lemma *arcs-ends-blocks-wf*:
assumes $bl \in\# \textit{arcs-blocks}$
shows $bl \subseteq \textit{verts}\ G$
proof −
obtain *x y* **where** *blpair*: $bl = \{x, y\}$ **using** *arcs-blocks-elem assms*
by *fastforce*
then have $(x, y) \in \textit{arcs-ends}\ G$ **using** *arcs-ends-blocks assms* **by** *simp*
thus *?thesis* **using** *arcs-ends-wf blpair* **by** *auto*
qed

lemma *arcs-blocks-simple*: $bl \in\# \textit{arcs-blocks} \Longrightarrow \textit{count}\ (\textit{arcs-blocks})\ bl = 1$
by (*simp add*: *arcs-blocks-def*)

lemma *arcs-blocks-ne*: $\textit{arcs}\ G \neq \{\} \Longrightarrow \textit{arcs-blocks} \neq \{\#\}$
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** $bl \in\# \textit{arcs-blocks}$ **shows** $\textit{card}\ bl = 2$

proof −
obtain *x y* **where** *blrep*: *bl* = {*x*, *y*} **using** *assms arcs-blocks-elem*
by *fastforce*
then have (*x*, *y*) ∈ *arcs-ends G* **using** *arcs-ends-blocks assms* **by** *simp*
then have *x* ≠ *y* **using** *no-loops* **using** *adj-not-same* **by** *blast*
thus *?thesis* **using** *blrep* **by** *simp*
qed

sublocale *non-empty-graph* ⊆ *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* ∈ *verts G* ⟹ (*e* ∈ (*in-arcs G v*) ⟹
∃ *e′* . (*e′* ∈ (*out-arcs G v*) ∧ *head G e′* = *tail G e*))
using *symmetric-conv sym-arcs* **by** *fastforce*

lemma *out-in-arcs-reflexive*: *v* ∈ *verts G* ⟹ (*e* ∈ (*out-arcs G v*) ⟹
∃ *e′* . (*e′* ∈ (*in-arcs G v*) ∧ *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* ∈ *verts G* **and** *u* ∈ *verts G*
assumes *e1* ∈ *in-arcs G v* **and** *e2* ∈ *in-arcs G v*
assumes *tail G e1* = *u* **and** *tail G e2* = *u*
shows *e1* = *e2*
proof −
have *in-arcs1*: *e1* ∈ *arcs G* **and** *in-arcs2*: *e2* ∈ *arcs G* **using** *assms* **by** *auto*
have *arc-to-ends G e1* = *arc-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* ∈ *verts G* **and** *u* ∈ *verts G*
assumes *e1* ∈ *out-arcs G v* **and** *e2* ∈ *out-arcs G v*
assumes *head G e1* = *u* **and** *head G e2* = *u*
shows *e1* = *e2*
proof −

have *in-arcs1*: *e1* ∈ *arcs G* **and** *in-arcs2*: *e2* ∈ *arcs G* **using** *assms* **by** *auto*
have *arc-to-ends G e1* = *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* ∈ *verts G* ⟹ (*e* ∈ (*in-arcs G v*) ⟹
∃! *e′* . (*e′* ∈ (*out-arcs G v*) ∧ *head G e′* = *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* ∈ *verts G* ⟹ *e* ∈ (*out-arcs G v*) ⟹
∃! *e′* . (*e′* ∈ (*in-arcs G v*) ∧ *tail G e′* = *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*) ∈ ((*arc-to-ends G*) ‘ (*in-arcs G v*)) ⟷
(*v*, *u*) ∈ ((*arc-to-ends G*) ‘ (*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*: *in-degree G v* = *card* ((*arc-to-ends G*) ‘ (*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*: *in-degree G v* = *card* (*arc-to-block* ‘ (*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*: *out-degree G v* = *card* (*arc-to-block* ‘ (*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* = *card* ((*arc-to-ends G*) ‘ (*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* (λ (*u*, *v*) . (*v*, *u*)) ((*arc-to-ends G*) ‘ (*in-arcs G v*)) ((*arc-to-ends G*) ‘ (*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* = *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* ‘ (*in-arcs G v*) = *arc-to-block* ‘ (*out-arcs G v*)
proof (*auto*)
fix *xa*
assume *a1*: *xa* ∈ *arcs G* **and** *a2*: *v* = *head G xa*
then have *xa* ∈ *in-arcs G v* **by** *simp*
then obtain *e* **where** *out*: *e* ∈ *out-arcs G v* **and** *head G e* = *tail G xa*
using *out-in-arcs-reflexive-uniq* **by** *force*
then have *arc-to-ends G e* = (*v*, *tail G xa*)
by (*simp add*: *arc-to-ends-def*)
then have *arc-to-block xa* = *arc-to-block e*
using *arc-to-block-sym* **by** (*metis a2 arc-to-ends-def*)
then show *arc-to-block xa* ∈ *arc-to-block* ‘ *out-arcs G* (*head G xa*)
using *out a2* **by** *blast*
next
fix *xa*
assume *a1*: *xa* ∈ *arcs G* **and** *a2*: *v* = *tail G xa*
then have *xa* ∈ *out-arcs G v* **by** *simp*
then obtain *e* **where** *ina*: *e* ∈ *in-arcs G v* **and** *tail G e* = *head G xa*
using *out-in-arcs-reflexive-uniq* **by** *force*
then have *arc-to-ends G e* = (*head G xa*, *v*)
by (*simp add*: *arc-to-ends-def*)
then have *arc-to-block xa* = *arc-to-block e*
using *arc-to-block-sym* **by** (*metis a2 arc-to-ends-def*)
then show *arc-to-block xa* ∈ *arc-to-block* ‘ *in-arcs G* (*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 :: *nat*
  assumes *in-deg-r*: $v \in$ *verts* $G \Longrightarrow$ *in-degree* G v = r
  assumes *out-deg-r*: $v \in$ *verts* $G \Longrightarrow$ *out-degree* G v = r

locale *regular-graph* = *graph* + *regular-digraph*
begin

lemma *rep-vertices-in-blocks* [*simp*]:
  assumes $x \in$ *verts* G
  shows *size* {# $e \in\#$ *arcs-blocks* . $x \in e$ #} = r
proof −
  have $\bigwedge e$. $e \in$ (*arc-to-block* ‘ (*arcs* G)) $\Longrightarrow$ $x \in e \Longrightarrow e \in$ (*arc-to-block* ‘ *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 { $e \in$ (*arc-to-block* ‘ (*arcs* G)) . $x \in e$} = (*arc-to-block* ‘ (*in-arcs* G x))
    using *arc-to-block-def* **by** *auto*
  then have *card* { $e \in$ (*arc-to-block* ‘ (*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 . $v \in$ (*verts* G) $\Longrightarrow$ *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 . $v \in$ (*verts* G) $\Longrightarrow$ *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* ⊆ *non-empty-graph*
by *unfold-locales*

sublocale *non-empty-regular-graph* ⊆ *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*: $x \in \mathcal{U} \Longrightarrow x \in \mathcal{V}$
using *points-subset* **by** *auto*

lemma *sub-blocks*: $bl \in\# \mathcal{A} \Longrightarrow bl \in\# \mathcal{B}$
using *blocks-subset* **by** *auto*

lemma *sub-blocks-count*: *count* $\mathcal{A}$ $b \leq$ *count* $\mathcal{B}$ 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*: $A \in\# \mathcal{A} \Longrightarrow A \neq \{\}$
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*: $\mathcal{U} \subset \mathcal{V}$
  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*: $\mathcal{U} \neq \mathcal{V} \Longrightarrow$ *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*: $\mathcal{U} \neq \mathcal{V} \Longrightarrow$ *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*: $\mathcal{U} \neq \mathcal{V} \Longrightarrow$ *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*]: *proper-sub-set-system* $\mathcal{U}$ $\mathcal{A}$ $\mathcal{V}$ $\mathcal{B} \Longrightarrow$ *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*!]: *incidence-system* $\mathcal{V}$ $\mathcal{B} \Longrightarrow \mathcal{U} \subseteq \mathcal{V} \Longrightarrow \mathcal{A} \subseteq\# \mathcal{B} \Longrightarrow$ *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*: *ins.sys-block-sizes* $\subseteq$ *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*: $x \in \mathcal{U} \Longrightarrow \mathcal{A}$ *rep* $x \leq \mathcal{B}$ *rep* x
by (*simp add*: *point-replication-number-def blocks-subset multiset-filter-mono size-mset-mono*)

lemma *sub-point-index-le*: $ps \subseteq \mathcal{U} \Longrightarrow \mathcal{A}$ *index* $ps \leq \mathcal{B}$ *index* ps
by (*simp add*: *points-index-def blocks-subset multiset-filter-mono size-mset-mono*)

lemma *sub-sys-intersection-numbers*: *ins.intersection-numbers* $\subseteq$ *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*]: $\mathcal{U} \subseteq \mathcal{V} \Longrightarrow \mathcal{A} \subseteq\# \mathcal{B} \Longrightarrow$ *sub-set-system* $\mathcal{U}\ \mathcal{A}\ \mathcal{V}\ \mathcal{B}$
by (*simp add*: *sub-ssI incidence-system-axioms*)

lemma *sub-inc-sysI*[*intro*]: *incidence-system* $\mathcal{U}\ \mathcal{A} \Longrightarrow \mathcal{U} \subseteq \mathcal{V} \Longrightarrow \mathcal{A} \subseteq\# \mathcal{B} \Longrightarrow$
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 *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*: *sub-incidence-system* $\mathcal{V}\ \mathcal{B}$ (*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*: *sub-incidence-system* (*del-point p*) (*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*: *sub-incidence-system* $\mathcal{V}\ \mathcal{B}$ ($\mathcal{V} \cup b$) (*add-block b*)
using *add-block-wf wf-invalid-point add-block-def* **by** (*unfold-locales*) (*auto*)

lemma *delete-block-sub-sys*: *sub-incidence-system* $\mathcal{V}$ (*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*: *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*: *sub-incidence-system* $\mathcal{V}'\ \mathcal{B}'\ \mathcal{V}^+$ $\mathcal{B}^+$
by (*unfold-locales*) (*simp-all*)

context *design*
begin

lemma *sub-designI* [*intro*]: *design* $\mathcal{U}\ \mathcal{A} \Longrightarrow$ *sub-incidence-system* $\mathcal{U}\ \mathcal{A}\ \mathcal{V}\ \mathcal{B} \Longrightarrow$ *sub-design* $\mathcal{U}\ \mathcal{A}\ \mathcal{V}\ \mathcal{B}$
by (*simp add*: *sub-design.intro wf-design*)

lemma *sub-designII* [*intro*]: *design* $\mathcal{U}$ $\mathcal{A}$ $\Longrightarrow$ *sub-incidence-system* $\mathcal{V}$ $\mathcal{B}$ $\mathcal{U}$ $\mathcal{A}$ $\Longrightarrow$ *sub-design* $\mathcal{V}$ $\mathcal{B}$ $\mathcal{U}$ $\mathcal{A}$
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* $\Longrightarrow$ $b \neq \{\}$ $\Longrightarrow$ *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* ((‘) *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* ‘ $\mathcal{V}$) (*blocks-image* $\mathcal{B}$ *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* ‘ $\mathcal{V}$) (*blocks-image* $\mathcal{B}$ *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* $\mathcal{V}$ $\mathcal{B}$ + *target*: *incidence-system* $\mathcal{V}'$ $\mathcal{B}'$
for $\mathcal{V}$ **and** $\mathcal{B}$ **and** $\mathcal{V}'$ **and** $\mathcal{B}'$ + **fixes** *bij-map* (‹π›)
assumes *bij*: *bij-betw* π $\mathcal{V}$ $\mathcal{V}'$
assumes *block-img*: *image-mset* ((‘) π) $\mathcal{B}$ = $\mathcal{B}'$
begin

lemma *iso-eq-order*: *card* $\mathcal{V}$ = *card* $\mathcal{V}'$
using *bij bij-betw-same-card* **by** *auto*

lemma *iso-eq-block-num*: *size* $\mathcal{B}$ = *size* $\mathcal{B}'$
using *block-img* **by** (*metis size-image-mset*)

lemma *iso-block-img-alt-rep*: {# π ‘ *bl* . *bl* ∈# $\mathcal{B}$#} = $\mathcal{B}'$
using *block-img* **by** *simp*

lemma *inv-iso-block-img*: *image-mset* ((‘) (*inv-into* $\mathcal{V}$ π)) $\mathcal{B}'$ = $\mathcal{B}$
proof −
have ⋀ *x*. *x* ∈ $\mathcal{V}$ ⟹ ((*inv-into* $\mathcal{V}$ π) ∘ π) *x* = *x*
using *bij bij-betw-inv-into-left comp-apply* **by** *fastforce*
then have ⋀ *bl x* . *bl* ∈# $\mathcal{B}$ ⟹ *x* ∈ *bl* ⟹ ((*inv-into* $\mathcal{V}$ π) ∘ π) *x* = *x*
using *source.wellformed* **by** *blast*
then have *img*: ⋀ *bl* . *bl* ∈# $\mathcal{B}$ ⟹ *image* ((*inv-into* $\mathcal{V}$ π) ∘ π) *bl* = *bl*
by *simp*
have *image-mset* ((‘) (*inv-into* $\mathcal{V}$ π)) $\mathcal{B}'$ = *image-mset* ((‘) (*inv-into* $\mathcal{V}$ π)) (*image-mset* ((‘) π) $\mathcal{B}$)
using *block-img* **by** *simp*
then have *image-mset* ((‘) (*inv-into* $\mathcal{V}$ π)) $\mathcal{B}'$ = *image-mset* ((‘) ((*inv-into* $\mathcal{V}$ π) ∘ π)) $\mathcal{B}$
by (*metis* (*no-types*, *opaque-lifting*) *block-img comp-apply image-comp multiset.map-comp multiset.map-cong0*)
thus *?thesis* **using** *img* **by** *simp*
qed

lemma *inverse-incidence-sys-iso*: *incidence-system-isomorphism* $\mathcal{V}'$ $\mathcal{B}'$ $\mathcal{V}$ $\mathcal{B}$ (*inv-into*

$\mathcal{V}$ π)
using *bij bij-betw-inv-into inv-iso-block-img* **by** (*unfold-locales*) *simp*

lemma *iso-points-map*: π ‘ $\mathcal{V} = \mathcal{V}'$
using *bij* **by** (*simp add*: *bij-betw-imp-surj-on*)

lemma *iso-points-inv-map*: (*inv-into* $\mathcal{V}$ π) ‘ $\mathcal{V}' = \mathcal{V}$
using *incidence-system-isomorphism.iso-points-map inverse-incidence-sys-iso* **by** *blast*

lemma *iso-points-ss-card*:
assumes *ps* $\subseteq$ $\mathcal{V}$
shows *card ps* = *card* (π ‘ *ps*)
using *assms bij bij-betw-same-card bij-betw-subset* **by** *blast*

lemma *iso-block-in*: *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ (π ‘ *bl*) $\in\#$ $\mathcal{B}'$
using *iso-block-img-alt-rep*
by (*metis image-eqI in-image-mset*)

lemma *iso-inv-block-in*: *x* $\in\#$ $\mathcal{B}'$ $\Longrightarrow$ *x* $\in$ (‘) π ‘ *set-mset* $\mathcal{B}$
by (*metis block-img in-image-mset*)

lemma *iso-img-block-orig-exists*: *x* $\in\#$ $\mathcal{B}'$ $\Longrightarrow$ $\exists$ *bl* . *bl* $\in\#$ $\mathcal{B}$ $\wedge$ *x* = π ‘ *bl*
using *iso-inv-block-in* **by** *blast*

lemma *iso-blocks-map-inj*: *x* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *y* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ π ‘ *x* = π ‘ *y* $\Longrightarrow$ *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* ((‘) π) (*set-mset* $\mathcal{B}$) (*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* ((‘) (*inv-into* $\mathcal{V}$ π)) (*set-mset* $\mathcal{B}'$) (*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* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *bij-betw* π *bl* (π ‘ *bl*)
using *bij bij-betw-subset source.wellformed* **by** *blast*

lemma *iso-bij-betw-individual-blocks-inv*: *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ *bij-betw* (*inv-into* $\mathcal{V}$ π) (π ‘ *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 $\in\#$ $\mathcal{B}'$ $\Longrightarrow$ *bij-betw* (*inv-into* $\mathcal{V}$ π) *bl* ((*inv-into* $\mathcal{V}$ π) ‘ *bl*)
using *incidence-system-isomorphism.iso-bij-betw-individual-blocks inverse-incidence-sys-iso*
by *blast*

lemma *iso-inv-block-in-alt*: $(\pi \mathbin{`} bl) \in\# \mathcal{B}' \Longrightarrow bl \subseteq \mathcal{V} \Longrightarrow bl \in\# \mathcal{B}$
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 $(\pi \mathbin{`} x) \notin\# \mathcal{B}'$
proof (*rule ccontr*)
assume *a*: $\neg\ \pi \mathbin{`} x \notin\# \mathcal{B}'$
then have *a*: $\pi \mathbin{`} x \in\# \mathcal{B}'$ **by** *simp*
then have $\bigwedge y\ .\ y \in (\pi \mathbin{`} x) \Longrightarrow (\mathit{inv\text{-}into}\ \mathcal{V}\ \pi)\ y \in \mathcal{V}$
using *target.wf-invalid-point iso-points-inv-map* **by** *auto*
then have $((`)\ (\mathit{inv\text{-}into}\ \mathcal{V}\ \pi))\ (\pi \mathbin{`} x) \in\# \mathcal{B}$
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 *source.multiplicity bl* = *target.multiplicity* $(\pi \mathbin{`} bl)$
proof (*cases* $bl \in\# \mathcal{B}$)
case *True*
have *inj-on* $((`)\ \pi)$ (*set-mset* $\mathcal{B}$)
using *bij-betw-imp-inj-on iso-bij-betwn-block-sets* **by** *auto*
then have *count* $\mathcal{B}$ *bl* = *count* $\mathcal{B}'$ $(\pi \mathbin{`} bl)$
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-mult*: *source.multiplicity bl* = *0*
by (*simp add*: *False count-eq-zero-iff*)
then have *target.multiplicity* $(\pi \mathbin{`} bl)$ = *0*
using *False count-inI iso-inv-block-in-alt*
by (*metis assms*)
thus *?thesis*
using *s-mult* **by** *simp*
qed

lemma *iso-point-in-block-img-iff*: $p \in \mathcal{V} \Longrightarrow bl \in\# \mathcal{B} \Longrightarrow p \in bl \longleftrightarrow (\pi\ p) \in (\pi \mathbin{`} bl)$
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*: $p \subseteq \mathcal{V} \Longrightarrow bl \in\# \mathcal{B} \Longrightarrow p \subseteq bl \longleftrightarrow (\pi \mathbin{`} p) \subseteq$

$(\pi$ ‘ $bl)$
apply *auto*
using *image-subset-iff iso-point-in-block-img-iff subset-iff* **by** *metis*

lemma *iso-is-image-block*: $\mathcal{B}' =$ *blocks-image* $\mathcal{B}$ π
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* $\mathcal{V}$ $\mathcal{B}$ $\mathcal{V}'$ $\mathcal{B}'$ π + *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* (‹π›)

context *design-isomorphism*
begin

lemma *inverse-design-isomorphism*: *design-isomorphism* $\mathcal{V}'$ $\mathcal{B}'$ $\mathcal{V}$ $\mathcal{B}$ (*inv-into* $\mathcal{V}$ π)
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** ‹$\cong_D$› *50*) **where**
$\mathcal{D} \cong_D \mathcal{D}' \longleftrightarrow$ ($\exists$ π . *design-isomorphism* (*fst* $\mathcal{D}$) (*snd* $\mathcal{D}$) (*fst* $\mathcal{D}'$) (*snd* $\mathcal{D}'$) π)

lemma *isomorphic-designs-symmetric*: $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \Longrightarrow (\mathcal{V}', \mathcal{B}') \cong_D (\mathcal{V}, \mathcal{B})$
using *isomorphic-designs-def design-isomorphism.inverse-design-isomorphism*
by *metis*

lemma *isomorphic-designs-implies-bij*: $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \Longrightarrow \exists$ π . *bij-betw* π $\mathcal{V}$ $\mathcal{V}'$
using *incidence-system-isomorphism.bij isomorphic-designs-def*
by (*metis design-isomorphism.axioms*(*1*) *fst-conv*)

lemma *isomorphic-designs-implies-block-map*: $(\mathcal{V}, \mathcal{B}) \cong_D (\mathcal{V}', \mathcal{B}') \Longrightarrow \exists$ π . *image-mset* ((‘) π) $\mathcal{B} = \mathcal{B}'$
using *incidence-system-isomorphism.block-img isomorphic-designs-def*
using *design-isomorphism.axioms*(*1*) **by** *fastforce*

context *design*
begin

lemma *isomorphic-designsI* [*intro*]: *design* $\mathcal{V}'$ $\mathcal{B}'$ $\Longrightarrow$ *bij-betw* π $\mathcal{V}$ $\mathcal{V}'$ $\Longrightarrow$ *image-mset* ((`) π) $\mathcal{B}$ = $\mathcal{B}'$
$\Longrightarrow$ ($\mathcal{V}$, $\mathcal{B}$) $\cong_D$ ($\mathcal{V}'$, $\mathcal{B}'$)
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 $\mathcal{B}$ *rep* p = $\mathcal{B}'$ *rep* (π p)
proof −
have {#b ∈# $\mathcal{B}$. $p \in b$#} = {#b ∈# $\mathcal{B}$. π $p \in \pi$ ` b#}
using *assms filter-mset-cong iso-point-in-block-img-iff assms* **by** *force*
then have {#b ∈# $\mathcal{B}'$. π $p \in b$#} = *image-mset* ((`) π) {#b ∈# $\mathcal{B}$. $p \in b$#}
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* ∈# $\mathcal{B}$ $\Longrightarrow$ *card bl* = *card* (π ` *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 $\mathcal{B}$ *index ps* = $\mathcal{B}'$ *index* (π ‘ *ps*)
proof −
have $\bigwedge b$. $b \in\# \mathcal{B} \Longrightarrow ((ps \subseteq b) = ((\pi$ ‘ *ps*) $\subseteq \pi$ ‘ *b*))
using *iso-point-subset-block-iff assms* **by** *blast*
then have {#*b* ∈# $\mathcal{B}$. *ps* ⊆ *b*#} = {#*b* ∈# $\mathcal{B}$. (π ‘ *ps*) ⊆ (π ‘ *b*)#}
using *assms filter-mset-cong* **by** *force*
then have {#*b* ∈# $\mathcal{B}'$. π ‘ *ps* ⊆ *b*#} = *image-mset* ((‘) π) {#*b* ∈# $\mathcal{B}$. *ps* ⊆ *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 $x \in$ *source.point-indices t*
shows $x \in$ *target.point-indices t*
proof −
obtain *ps* **where** *t*: *card ps* = *t* **and** *ss*: *ps* ⊆ $\mathcal{V}$ **and** *x*: $\mathcal{B}$ *index ps* = *x* **using** *assms*
by (*auto simp add*: *source.point-indices-def*)
then have *x-val*: *x* = $\mathcal{B}'$ *index* (π ‘ *ps*) **using** *design-iso-points-index-eq* **by** *auto*
have *x-img*: (π ‘ *ps*) ⊆ $\mathcal{V}'$
using *ss bij iso-points-map* **by** *fastforce*
then have *card* (π ‘ *ps*) = *t* **using** *t ss iso-points-ss-card* **by** *auto*
then show *?thesis* **using** *target.point-indices-elem-in x-img x-val* **by** *blast*
qed

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* ∈# $\mathcal{B}$
assumes *b2* ∈# $\mathcal{B}$
shows *b1* |∩| *b2* = (π ‘ *b1*) |∩| (π ‘ *b2*)
proof −
have *split*: π ‘ (*b1* ∩ *b2*) = (π ‘ *b1*) ∩ (π ‘ *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 $x \in$ *source.intersection-numbers*
shows $x \in$ *target.intersection-numbers*
proof −
obtain *b1 b2* **where** *1*: $b1 \in\# \mathcal{B}$ **and** *2*: $b2 \in\#$ (*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*: π ‘ $b1 \in\# \mathcal{B}'$ **by** (*simp add*: *iso-block-in*)
have *pi2*: π ‘ $b2 \in\#$ (*remove1-mset* (π ‘ *b1*) $\mathcal{B}'$) **using** *iso-block-in 2*
by (*metis* (*no-types*, *lifting*) *1 block-img image-mset-remove1-mset-if in-remove1-mset-neq*

iso-blocks-map-inj more-than-one-mset-mset-diff multiset.set-map)
have $x = (\pi$ ‘ *b1*) $\mid\cap\mid$ (π ‘ *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 $b1 \mid\cap\mid_n b2 = ((\pi$ ‘ *b1*) $\mid\cap\mid_n$ (π ‘ *b2*))
proof −
let *?A* = $\{x\ .\ x \subseteq b1 \wedge x \subseteq b2 \wedge$ *card* $x = n\}$
let *?B* = $\{y\ .\ y \subseteq (\pi$ ‘ *b1*) $\wedge\ y \subseteq (\pi$ ‘ *b2*) $\wedge$ *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 $\bigwedge x\ y\ .\ x \subseteq b1 \Longrightarrow x \subseteq b2 \Longrightarrow y \subseteq b1 \Longrightarrow y \subseteq b2 \Longrightarrow\ \pi$ ‘ $x = \pi$ ‘ $y \Longrightarrow x = y$
using *b1v bij* **by** (*metis bij-betw-imp-surj-on bij-betw-inv-into-subset dual-order.trans*)
then have *inj*: *inj-on* (($\cdot$) π) *?A* **by** (*simp add*: *inj-on-def*)
have *eqcard*: $\bigwedge xa.\ xa \subseteq b1 \Longrightarrow xa \subseteq b2 \Longrightarrow$ *card* (π ‘ *xa*) = *card xa* **using** *b1v b2v bij*
using *iso-points-ss-card* **by** *auto*
have *surj*: $\bigwedge x.\ x \subseteq \pi$ ‘ $b1 \Longrightarrow x \subseteq \pi$ ‘ $b2 \Longrightarrow$
$x \in \{(\pi$ ‘ *xa*) $\mid$ *xa* . *xa* $\subseteq b1 \wedge xa \subseteq b2 \wedge$ *card xa* = *card x*$\}$
proof −
fix *x*
assume *x1*: $x \subseteq \pi$ ‘ *b1* **and** *x2*: $x \subseteq \pi$ ‘ *b2*
then obtain *xa* **where** *eq-x*: π ‘ *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*: *card xa* = *card x* **using** *bij* **by** (*simp add*: *eq-x ss iso-points-ss-card*)
then show $x \in \{(\pi \mathbin{`} xa) \mid xa\ .\ xa \subseteq b1 \wedge xa \subseteq b2 \wedge card\ xa = card\ x\}$
using *f1 f2 f3* ‹$\pi \mathbin{`} xa = x$› **by** *auto*
qed
have *bij-betw* (($\mathbin{`}$) π) *?A ?B*
proof (*auto simp add*: *bij-betw-def*)
show *inj-on* (($\mathbin{`}$) π) $\{x.\ x \subseteq b1 \wedge x \subseteq b2 \wedge card\ x = n\}$ **using** *inj* **by** *simp*
show $\bigwedge xa.\ xa \subseteq b1 \Longrightarrow xa \subseteq b2 \Longrightarrow n = card\ xa \Longrightarrow card\ (\pi \mathbin{`} xa) = card\ xa$
using *eqcard* **by** *simp*
show $\bigwedge x.\ x \subseteq \pi \mathbin{`} b1 \Longrightarrow x \subseteq \pi \mathbin{`} b2 \Longrightarrow n = card\ x \Longrightarrow$
$x \in (\mathbin{`})\ \pi \mathbin{`} \{xa.\ xa \subseteq b1 \wedge xa \subseteq b2 \wedge card\ xa = card\ x\}$
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* ($\pi \mathbin{`} V$) (*blocks-image B* π) $\mathcal{V}'$ $\mathcal{B}'$
proof (*unfold-locales*)
show $\pi \mathbin{`} V \subseteq \mathcal{V}'$
by (*metis assms image-mono iso-points-map sub-set-system.points-subset*)
show *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* ($\pi \mathbin{`} V$) (*blocks-image B* π)
proof −
interpret *fin*: *finite-incidence-system* ($\pi \mathbin{`} V$) (*blocks-image B* π) **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*: *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 $\{\} \in\#$ *blocks-image B* $\pi \Longrightarrow$ *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* (π ‘ *V*) (*blocks-image B* π) π
proof −
interpret *design* (π ‘ *V*) (*blocks-image B* π)
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* π *V* (π ‘ *V*) **using** *bij*
by (*metis assms*(*1*) *bij-betw-subset sub-set-system.points-subset*)
show *image-mset* ((‘) π) *B* = *blocks-image B* π **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.