

Fisher's Inequality: Linear Algebraic Proof Techniques for Combinatorics

Chelsea Edmonds and Lawrence C. Paulson

September 13, 2023

Abstract

Linear algebraic techniques are powerful, yet often underrated tools in combinatorial proofs. This formalisation provides a library including matrix representations of incidence set systems, general formal proof techniques for the rank argument and linear bound argument, and finally a formalisation of a number of variations of the well-known Fisher's inequality. We build on our prior work formalising combinatorial design theory using a locale-centric approach, including extensions such as constant intersect designs and dual incidence systems. In addition to Fisher's inequality, we also formalise proofs on other incidence system properties using the incidence matrix representation, such as design existence, dual system relationships and incidence system isomorphisms. This formalisation is presented in the paper "Formalising Fisher's Inequality: Formal Linear Algebraic Techniques in Combinatorics", accepted to ITP 2022.

Contents

1	Micellaneous Multiset/Set Extras	3
1.1	Set extras	3
1.2	Multiset Extras	3
1.3	Permutation on Sets and Multisets	5
1.4	Lists	5
1.5	Summation Rules	7
2	Matrix and Vector Additions	9
2.1	Vector Extras	9
2.2	Matrix Extras	18
2.3	Vector and Matrix Homomorphism	21
2.4	Zero One injections and homomorphisms	22
3	Micellaneous Design Extras	25
3.1	Extensions to existing Locales and Properties	26
3.2	New Design Locales	35

4	Incidence Vectors and Matrices	39
4.1	Incidence Vectors	39
4.2	Incidence Matrices	40
4.3	0-1 Matrices	45
4.4	Ordered Incidence Systems	53
4.5	Incidence Matrices on Design Subtypes	71
4.6	Zero One Matrix Incidence System Existence	79
4.7	Isomorphisms and Incidence Matrices	84
5	Dual Systems	86
5.1	Dual Blocks	86
5.2	Basic Dual Properties	87
5.3	Incidence System Dual Properties	89
5.4	Dual Properties for Design sub types	91
5.5	Generalise Dual Concept	95
6	Rank Argument - General	97
6.1	Row/Column Operations	97
6.2	Rank and Linear Independence	106
7	Linear Bound Argument - General	107
7.1	Vec Space Extensions	107
7.2	Linear Bound Argument Lemmas	109
8	Fisher's Inequality	111
8.1	Uniform Fisher's Inequality	111
8.2	Generalised Fisher's Inequality	115
9	Matrices/Vectors mod x	119
9.1	Basic Mod Context	120
9.2	Mod Type	123
9.3	Mat mod type	125
10	Variations on Fisher's Inequality	131
10.1	Matrix mod properties	131
10.2	The Odd-town Problem	133

Acknowledgements The authors were supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council.

1 Micellaneous Multiset/Set Extras

theory *Set-Multiset-Extras* **imports** *Design-Theory.Multisets-Extras HOL-Combinatorics.Multiset-Permutat*
begin

1.1 Set extras

Minor set extras on cardinality and filtering

lemma *equal-card-inter-fin-eq-sets*: $finite\ A \implies finite\ B \implies card\ A = card\ B \implies$

$card\ (A \cap B) = card\ A \implies A = B$
by (*metis Int-lower1 Int-lower2 card-subset-eq*)

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

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

1.2 Multiset Extras

Minor multiset extras on size and element reasoning

lemma *obtain-two-items-mset*:

assumes $size\ A > 1$

obtains $x\ y$ **where** $x \in\# A$ **and** $y \in\# A - \{\#x\# \}$

proof –

obtain x **where** $x \in\# A$

by (*metis assms gr-implies-not-zero multiset-nonemptyE size-empty*)

have $size\ (A - \{\#x\# \}) > 0$

by (*metis $\langle x \in\# A \rangle$ assms insert-DiffM less-irrefl-nat nonempty-has-size size-single*)

then obtain $bl2$ **where** $bl2 \in\# A - \{\#x\# \}$

by (*metis less-not-refl multiset-nonemptyE size-empty*)

thus *?thesis*

using $\langle x \in\# A \rangle$ **that** **by** *blast*

qed

lemma *obtain-two-items-mset-filter*:

assumes $size\ \{\#a \in\# A . P\ a\ \} > 1$

obtains $x\ y$ **where** $x \in\# A$ **and** $y \in\# A - \{\#x\# \}$ **and** $P\ x$ **and** $P\ y$

proof –

obtain $x\ y$ **where** $xin: x \in\# \{\#a \in\# A . P\ a\ \}$ **and** $ysin: y \in\# \{\#a \in\# A . P\ a\ \} - \{\#x\# \}$

using *obtain-two-items-mset assms* **by** *blast*

then have $xdets: x \in\# A\ P\ x$ **by** *auto*

then have $ysin2: y \in\# \{\#a \in\# (A - \{\#x\# \}) . P\ a\ \}$ **using** $ysin$

by *force*

then have $y \in\# (A - \{\#x\# \})\ P\ y$

by (metis multiset-partition union-iff) (meson yin2 filter-mset-eq-conv)
 thus ?thesis using xdets that by blast
 qed

lemma size-count-mset-ss:

assumes finite B
 assumes (set-mset A) \subseteq B
 shows size A = ($\sum x \in B . count A x$)
proof –
 obtain C where cdef: B - (set-mset A) = C using assms
 by simp
 have fin: finite (set-mset A) using assms by auto
 have un: C \cup (set-mset A) = B
 using Diff-partition $\langle B - set-mset A = C \rangle$ assms by blast
 have disj: C \cap (set-mset A) = {}
 using $\langle B - set-mset A = C \rangle$ by auto
 have zero: $\bigwedge x . x \in C \implies count A x = 0$
 by (meson count-eq-zero-iff disj disjoint-iff-not-equal)
 then have ($\sum x \in B . count A x$) = ($\sum x \in (C \cup set-mset A) . count A x$)
 using un by simp
 also have ... = ($\sum x \in C . count A x$) + ($\sum x \in (set-mset A) . count A x$)
 using disj fin assms cdef sum.subset-diff by (metis un)
 also have ... = ($\sum x \in (set-mset A) . count A x$) using zero by auto
 finally have ($\sum x \in B . count A x$) = size A
 by (simp add: size-multiset-overloaded-eq)
 thus ?thesis by simp
 qed

lemma mset-list-by-index: mset (xs) = image-mset ($\lambda i . (xs ! i)$) (mset-set
 {.. $length xs$ })
 by (metis map-nth mset-map mset-set-upto-eq-mset-upto)

lemma count-mset-split-image-filter:

assumes $\bigwedge x . x \in \#A \implies a \neq g x$
 shows count (image-mset ($\lambda x . if P x then a else g x$) A) a = size (filter-mset P
 A)
 using image-mset-If image-mset-filter-swap size-image-mset
 by (smt (verit) assms count-size-set-repr filter-mset-cong)

lemma count-mset-split-image-filter-not:

assumes $\bigwedge x . x \in \#A \implies b \neq f x$
 shows count (image-mset ($\lambda x . if P x then f x else b$) A) b = size (filter-mset (λ
 $x . \neg P x$) A)
 using image-mset-If image-mset-filter-swap size-image-mset
 by (smt (verit) assms count-size-set-repr filter-mset-cong)

lemma removeAll-size-lt: size (removeAll-mset C M) \leq size M
 by (simp add: size-mset-mono)

lemma *mset-image-eq-filter-eq*: $A = \text{image-mset } f \ B \implies$
 $\text{filter-mset } P \ A = (\text{image-mset } f \ (\text{filter-mset } (\lambda x. P \ (f \ x)) \ B))$
by (*simp add: filter-mset-image-mset*)

1.3 Permutation on Sets and Multisets

lemma *elem-permutation-of-set-empty-iff*: $\text{finite } A \implies xs \in \text{permutations-of-set } A \implies$

$xs = [] \iff A = \{\}$
using *permutations-of-setD(1)* **by** *fastforce*

lemma *elem-permutation-of-mset-empty-iff*: $xs \in \text{permutations-of-multiset } A \implies$
 $xs = [] \iff A = \{\#\}$

using *permutations-of-multisetD* **by** *fastforce*

1.4 Lists

Further lemmas on the relationship between lists and multisets

lemma *count-distinct-mset-list-index*: $i1 < \text{length } xs \implies i2 < \text{length } xs \implies i1 \neq i2 \implies$

$\text{distinct-mset } (\text{mset } xs) \implies xs ! i1 \neq xs ! i2$
by (*simp add: nth-eq-iff-index-eq*)

lemma *index-remove1-mset-ne*:

assumes $x \in \# (\text{mset } xs)$
assumes $y \in \# \text{remove1-mset } x \ (\text{mset } xs)$
assumes $xs ! j1 = x$
assumes $j1 < \text{length } xs$
obtains $j2$ **where** $xs ! j2 = y$ **and** $j2 < \text{length } xs$ **and** $j1 \neq j2$

proof (*cases* $x = y$)

case *True*

then have $\text{count } (\text{mset } xs) \ x \geq 2$

using *assms(2)* *count-eq-zero-iff* **by** *fastforce*

then have $\text{crm: count } (\text{remove1-mset } x \ (\text{mset } xs)) \ x \geq 1$

by (*metis True assms(2) count-greater-eq-one-iff*)

obtain $ys \ zs$ **where** $xseq: xs = ys @ (x \# \ zs)$ **and** $yseq: ys = \text{take } j1 \ xs$ **and**
 $zseq: zs = \text{drop } (\text{Suc } j1) \ xs$

using *assms(1)* *assms(3)* *id-take-nth-drop in-mset-conv-nth* *assms(4)* **by** *blast*

have $\text{mset } xs = \text{mset } ys + \text{mset } (x \# \ zs)$

by (*simp add: xseq*)

then have $\text{remove1-mset } x \ (\text{mset } xs) = \text{mset } (ys) + \text{mset } (zs)$

using *assms* **by** *simp*

then have $y \in \# (\text{mset } ys + \text{mset } zs)$ **using** *crm*

using *True* $\langle \text{remove1-mset } x \ (\text{mset } xs) = \text{mset } ys + \text{mset } zs \rangle$ *assms(2)* **by**
presburger

then have $y \in \# \text{mset } ys \vee y \in \# \text{mset } zs$ **by** *simp*

then show *?thesis* **proof** (*cases* $y \in \# \text{mset } ys$)

case *True*

then obtain $j2$ **where** $yseq: ys ! j2 = y$ **and** $j2lt: j2 < \text{length } ys$

```

    by (meson in-mset-conv-nth)
  then have 1:  $xs ! j2 = y$  using yseq by simp
  have  $j2 < j1$  using yseq j2lt by simp
  then show ?thesis using that 1 j2lt xseq by simp
next
case False
then have  $y \in \# mset\ zs$  using yinor by simp
then obtain  $j2$  where zsy:  $zs ! j2 = y$  and j2lt:  $j2 < length\ zs$ 
  by (meson in-mset-conv-nth)
then have 1:  $xs ! ((Suc\ j1) + j2) = y$  using zseq zsy assms(4) by simp
have  $length\ xs = (Suc\ j1) + length\ zs$  using zseq xseq
  by (metis Suc-diff-Suc add-Suc-shift add-diff-inverse-nat assms(4) length-drop
less-imp-not-less)
  then have 2:  $(Suc\ j1) + j2 < length\ xs$  using j2lt by simp
  then show ?thesis using 1 that by simp
qed
next
case False
then show ?thesis
  by (metis that assms(2) assms(3) in-diffD in-mset-conv-nth)
qed

```

lemma count-list-mset: $count\ list\ xs\ x = count\ (mset\ xs)\ x$

proof (induct xs)

```

  case Nil
  then show ?case by simp
next
case (Cons a xs)
then show ?case proof (cases a = x)
  case True
  have mset-add-split:  $count\ (mset\ (a\ \#\ xs))\ x = count\ (add-mset\ a\ (mset\ xs))\ x$ 
    by simp
  then have  $count\ (mset\ (a\ \#\ xs))\ x = count\ (mset\ xs)\ x + 1$ 
    by (metis True Suc-eq-plus1 count-add-mset)
  then show ?thesis using True Cons.hyps by simp
next
case False
then show ?thesis using Cons.hyps by simp
qed
qed

```

lemma count-min-2-indices-lt:

assumes $i1 < i2$

assumes $xs ! i1 = x$

assumes $xs ! i2 = x$

assumes $i1 < length\ xs$ $i2 < length\ xs$

shows $count\ (mset\ xs)\ x \geq 2$

proof –

obtain $xs1\ xs2$ where xse: $xs = xs1\ @\ xs2$ and $xs1$: $xs1 = take\ i2\ xs$ and $xs2$:

```

xs2 = drop i2 xs
  by simp
  have i1 < length xs1 using assms length-take
    by (simp add: assms(4) ⟨xs1 = take i2 xs⟩)
  then have xs1in: xs ! i1 ∈# mset xs1
    by (simp add: nth-append xse)
  have i2 ≥ length xs1 using assms length-take xs1 by simp
  then have xs2in: xs ! i2 ∈# mset xs2 using xse nth-append
    by (metis (no-types, lifting) assms(5) in-mset-conv-nth leD leI take-all-iff
take-append)
  have count (mset xs) x = count ((mset xs1) + (mset xs2)) x
    by (simp add: xse)
  then have count (mset xs) x = count (mset xs1) x + count (mset xs2) x by
simp
  thus ?thesis using xs1in xs2in
    by (metis add-mono assms(2) assms(3) count-greater-eq-one-iff nat-1-add-1)
qed

```

```

lemma count-min-2-indices: i1 ≠ i2 ⇒ xs ! i1 = x ⇒ xs ! i2 = x ⇒ i1 <
length xs ⇒
  i2 < length xs ⇒ count (mset xs) x ≥ 2
apply (cases i1 < i2, simp add: count-min-2-indices-lt)
by (metis count-min-2-indices-lt linorder-cases)

```

```

lemma obtain-set-list-item:
  assumes x ∈ set xs
  obtains i where i < length xs and xs ! i = x
  by (meson assms in-set-conv-nth)

```

1.5 Summation Rules

Some lemmas to make it simpler to work with double and triple summations

```

context comm-monoid-add
begin

```

```

lemma sum-reorder-triple: (∑ l ∈ A . (∑ i ∈ B . (∑ j ∈ C . g l i j))) =
(∑ i ∈ B . (∑ j ∈ C . (∑ l ∈ A . g l i j)))
proof –
  have (∑ l ∈ A . (∑ i ∈ B . (∑ j ∈ C . g l i j))) = (∑ i ∈ B . (∑ l ∈ A .
(∑ j ∈ C . g l i j)))
    using sum.swap[of (λ l i . (∑ j ∈ C . g l i j)) B A] by simp
  also have ... = (∑ i ∈ B . (∑ j ∈ C . (∑ l ∈ A . g l i j))) using sum.swap
by metis
  finally show ?thesis by simp
qed

```

```

lemma double-sum-mult-hom:
  fixes k :: 'b :: {comm-ring-1}
  shows (∑ i ∈ A . (∑ j ∈ g i . k * (f i j))) = k * (∑ i ∈ A . (∑ j ∈ g i . f i j))

```

by (*metis* (*mono-tags*, *lifting*) *comm-monoid-add-class.sum.cong sum-distrib-left*)

lemma *double-sum-split-case:*

assumes *finite A*

shows $(\sum i \in A . (\sum j \in A . f i j)) = (\sum i \in A . (f i i)) + (\sum i \in A . (\sum j \in (A - \{i\}) . f i j))$

proof –

have $\bigwedge i . i \in A \implies (\sum j \in A . f i j) = f i i + (\sum j \in (A - \{i\}) . f i j)$

using *sum.remove assms by metis*

then show *?thesis* by (*simp add: sum.distrib*)

qed

lemma *double-sum-split-case2:* $(\sum i \in A . (\sum j \in A . g i j)) =$

$(\sum i \in A . (g i i)) + (\sum i \in A . (\sum j \in \{a \in A . a \neq i\} . g i j))$

proof –

have $\bigwedge i . A = \{a \in A . a = i\} \cup \{a \in A . a \neq i\}$ by *auto*

then have *part*: $\bigwedge i . i \in A \implies A = \{i\} \cup \{a \in A . a \neq i\}$ by *auto*

have *empt*: $\bigwedge i . \{i\} \cap \{a \in A . a \neq i\} = \{\}$

by *simp*

then have $(\sum i \in A . (\sum j \in A . g i j)) =$

$(\sum i \in A . ((\sum j \in \{i\} . g i j) + (\sum j \in \{a \in A . a \neq i\} . g i j)))$ using *part*

by (*smt (verit) finite-Un local.sum.cong local.sum.infinite local.sum.union-disjoint*)

also have $\dots = (\sum i \in A . ((\sum j \in \{i\} . g i j))) + (\sum i \in A . (\sum j \in \{a \in A . a \neq i\} . g i j))$

by (*simp add: local.sum.distrib*)

finally show *?thesis* by *simp*

qed

end

context *comm-ring-1*

begin

lemma *double-sum-split-case-square:*

assumes *finite A*

shows $(\sum i \in A . f i)^2 = (\sum i \in A . (f i * f i)) + (\sum i \in A . (\sum j \in (A - \{i\}) . f i * f j))$

proof –

have $(\sum i \in A . f i)^2 = (\sum i \in A . f i) * (\sum i \in A . f i)$

using *power2-eq-square* by *blast*

then have $(\sum i \in A . f i) * (\sum i \in A . f i) = (\sum i \in A . f i) * (\sum j \in A . f j)$ by *simp*

also have 1: $\dots = (\sum i \in A . f i * (\sum j \in A . f j))$ using *sum-distrib-right* by *simp*

also have 2: $\dots = (\sum i \in A . (\sum j \in A . f i * f j))$ using *sum-distrib-left* by *metis*

finally have $(\sum i \in A . f i) * (\sum i \in A . f i) =$

$(\sum i \in A . (f i * f i)) + (\sum i \in A . (\sum j \in (A - \{i\}) . f i * f j))$

using *assms double-sum-split-case*[of $A \lambda i j . f i * f j$] 1 2 **by** *presburger*
then show *?thesis*
using *power2-eq-square* **by** *presburger*
qed

lemma *double-sum-split-square-diff*: *finite* $\{0..<x\} \implies$
 $(\sum i \in \{0..<x\} . (\sum j \in (\{0..<x\} - \{i\}) . c i * c j)) =$
 $(\sum i \in \{0..<x\} . c i)^2 - (\sum i \in \{0..<x\} . c i * c i)$
using *double-sum-split-case-square*[of $\{0..<x\} \lambda i . c i$] **by** *fastforce*

end
end

2 Matrix and Vector Additions

theory *Matrix-Vector-Extras* **imports** *Set-Multiset-Extras Jordan-Normal-Form.Matrix*

Design-Theory.Multisets-Extras Groebner-Bases.Macaulay-Matrix Polynomial-Factorization.Missing-List
begin

2.1 Vector Extras

For ease of use, a number of additions to the existing vector library as initially developed in the JNF AFP Entry, are given below

We define the concept of summing up elements of a vector

definition (**in** *comm-monoid-add*) *sum-vec* :: '*a* *vec* \Rightarrow '*a* **where**
sum-vec *v* \equiv *sum* ($\lambda i . v \$ i$) $\{0..<dim-vec\ v\}$

lemma *sum-vec-vNil*[*simp*]: *sum-vec* *vNil* = 0
by (*simp add: sum-vec-def*)

lemma *sum-vec-vCons*: *sum-vec* (*vCons* *a* *v*) = *a* + *sum-vec* *v*

proof –

have 0: *a* = (*vCons* *a* *v*) \$ 0

by *simp*

have *sum-vec* *v* = *sum* ($\lambda i . v \$ i$) $\{0..<dim-vec\ v\}$ **by** (*simp add: sum-vec-def*)

also have ... = *sum* ($\lambda i . (vCons\ a\ v)\ \$\ Suc\ i$) $\{0..<dim-vec\ v\}$

by *force*

also have ... = *sum* ($\lambda i . (vCons\ a\ v)\ \$\ i$) $\{Suc\ 0..<(Suc\ (dim-vec\ v))\}$

by (*metis sum.shift-bounds-Suc-ivl*)

finally have *sum*: *sum-vec* *v* = *sum* ($\lambda i . (vCons\ a\ v)\ \$\ i$) $\{Suc\ 0..<dim-vec\ (vCons\ a\ v)\}$ **by** *simp*

have *sum-vec* (*vCons* *a* *v*) = *sum* ($\lambda i . (vCons\ a\ v)\ \$\ i$) $\{0..<dim-vec\ (vCons\ a\ v)\}$

by (*simp add: sum-vec-def*)

then have *sum-vec* (*vCons* *a* *v*) = (*vCons* *a* *v*) \$ 0 + *sum* ($\lambda i . (vCons\ a\ v)\ \$\ i$) $\{Suc\ 0..<dim-vec\ (vCons\ a\ v)\}$

by (*metis dim-vec-vCons sum.atLeast-Suc-lessThan zero-less-Suc*)

thus *?thesis* **using** *sum 0* **by** *simp*
qed

lemma *sum-vec-list*: $\text{sum-list } (\text{list-of-vec } v) = \text{sum-vec } v$
by (*induct v*)(*simp-all add: sum-vec-vCons*)

lemma *sum-vec-mset*: $\text{sum-vec } v = (\sum x \in\# (\text{mset } (\text{list-of-vec } v)) . x)$
by (*simp add: sum-vec-list*)

lemma *dim-vec-vCons-ne-0*: $\text{dim-vec } (v\text{Cons } a \ v) > 0$
by (*cases v*) *simp-all*

lemma *sum-vec-vCons-lt*:
assumes $\bigwedge i. i < \text{dim-vec } (v\text{Cons } a \ v) \implies (v\text{Cons } a \ v) \$ i \leq (n :: \text{int})$
assumes $\text{sum-vec } v \leq m$
shows $\text{sum-vec } (v\text{Cons } a \ v) \leq n + m$

proof –

have *split*: $\text{sum-vec } (v\text{Cons } a \ v) = a + \text{sum-vec } v$ **by** (*simp add: sum-vec-vCons*)
have *a*: $(v\text{Cons } a \ v) \$ 0 = a$ **by** *simp*
then have $0 < \text{dim-vec } (v\text{Cons } a \ v)$ **using** *dim-vec-vCons-ne-0* **by** *simp*
then have $a \leq n$ **using** *assms* **by** (*metis a*)
thus *?thesis* **using** *split assms*
by (*simp add: add-mono*)

qed

lemma *sum-vec-one-zero*:
assumes $\bigwedge i. i < \text{dim-vec } (v :: \text{int vec}) \implies v \$ i \leq (1 :: \text{int})$
shows $\text{sum-vec } v \leq \text{dim-vec } v$
using *assms*

proof (*induct v*)

case *vNil*

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

next

case (*vCons a v*)

then have $\bigwedge i. i < \text{dim-vec } v \implies v \$ i \leq (1 :: \text{int})$

using *vCons.prem*s **by** *force*

then have *lt*: $\text{sum-vec } v \leq \text{int } (\text{dim-vec } v)$ **by** (*simp add: vCons.hyps*)

then show *?case* **using** *sum-vec-vCons-lt lt vCons.prem*s **by** *simp*

qed

Definition to convert a vector to a multiset

definition *vec-mset*:: $'a \ \text{vec} \Rightarrow 'a \ \text{multiset}$ **where**
 $\text{vec-mset } v \equiv \text{image-mset } (\text{vec-index } v) (\text{mset-set } \{.. < \text{dim-vec } v\})$

lemma *vec-elem-exists-mset*: $(\exists i \in \{.. < \text{dim-vec } v\}. v \$ i = x) \longleftrightarrow x \in\# \text{vec-mset } v$
by (*auto simp add: vec-mset-def*)

lemma *mset-vec-same-size*: $\text{dim-vec } v = \text{size } (\text{vec-mset } v)$

by (simp add: vec-mset-def)

lemma *mset-vec-eq-mset-list*: $\text{vec-mset } v = \text{mset } (\text{list-of-vec } v)$
by (auto simp add: vec-mset-def)
(metis list-of-vec-map mset-map mset-set-upto-eq-mset-upto)

lemma *vec-mset-img-map*: $\text{image-mset } f (\text{mset } (xs)) = \text{vec-mset } (\text{map-vec } f (\text{vec-of-list } xs))$
by (metis list-vec mset-map mset-vec-eq-mset-list vec-of-list-map)

lemma *vec-mset-vNil*: $\text{vec-mset } vNil = \{\#\}$
by (simp add: vec-mset-def)

lemma *vec-mset-vCons*: $\text{vec-mset } (vCons x v) = \text{add-mset } x (\text{vec-mset } v)$

proof –

have $\text{vec-mset } (vCons x v) = \text{mset } (\text{list-of-vec } (vCons x v))$
by (simp add: mset-vec-eq-mset-list)
then have $\text{mset } (\text{list-of-vec } (vCons x v)) = \text{add-mset } x (\text{mset } (\text{list-of-vec } v))$
by simp
thus ?thesis
by (metis mset-vec-eq-mset-list)

qed

lemma *vec-mset-set*: $\text{vec-set } v = \text{set-mset } (\text{vec-mset } v)$
by (simp add: mset-vec-eq-mset-list set-list-of-vec)

lemma *vCons-set-contains-in*: $a \in \text{set}_v v \implies \text{set}_v (vCons a v) = \text{set}_v v$
by (metis remdups-mset-singleton-sum set-mset-remdups-mset vec-mset-set vec-mset-vCons)

lemma *vCons-set-contains-add*: $a \notin \text{set}_v v \implies \text{set}_v (vCons a v) = \text{set}_v v \cup \{a\}$
using *vec-mset-set* *vec-mset-vCons*
by (metis Un-insert-right set-mset-add-mset-insert sup-bot-right)

lemma *setv-vec-mset-not-in-iff*: $a \notin \text{set}_v v \iff a \notin \#\text{vec-mset } v$
by (simp add: vec-mset-set)

Abbreviation for counting occurrences of an element in a vector

abbreviation *count-vec* $v a \equiv \text{count } (\text{vec-mset } v) a$

lemma *vec-count-lt-dim*: $\text{count-vec } v a \leq \text{dim-vec } v$
by (metis mset-vec-same-size order-refl set-count-size-min)

lemma *count-vec-empty*: $\text{dim-vec } v = 0 \implies \text{count-vec } v a = 0$
by (simp add: mset-vec-same-size)

lemma *count-vec-vNil*: $\text{count-vec } vNil a = 0$
by (simp add: vec-mset-def)

lemma *count-vec-vCons*: $\text{count-vec } (vCons aa v) a = (\text{if } (aa = a) \text{ then } \text{count-vec } v a + 1 \text{ else } \text{count-vec } v a)$

$v\ a + 1$ else $\text{count-vec } v\ a$)
by (*simp add: vec-mset-vCons*)

lemma *elem-exists-count-min*: $\exists i \in \{..< \text{dim-vec } v\}. v\ \$\ i = x \implies \text{count-vec } v\ x \geq 1$
by (*simp add: vec-elem-exists-mset*)

lemma *count-vec-count-mset*: $\text{vec-mset } v = \text{image-mset } f\ A \implies \text{count-vec } v\ a = \text{count } (\text{image-mset } f\ A)\ a$
by (*simp*)

lemma *count-vec-alt-list*: $\text{count-vec } v\ a = \text{length } (\text{filter } (\lambda y. a = y) (\text{list-of-vec } v))$
by (*simp add: mset-vec-eq-mset-list*) (*metis count-mset*)

lemma *count-vec-alt*: $\text{count-vec } v\ x = \text{card } \{i. v\ \$\ i = x \wedge i < \text{dim-vec } v\}$
proof –

have $\text{count-vec } v\ x = \text{count } (\text{image-mset } ((\$)\ v) (\text{mset-set } \{..< \text{dim-vec } v\}))\ x$ **by**
(*simp add: vec-mset-def*)

also have $\dots = \text{size } \{\#a \in \# (\text{image-mset } ((\$)\ v) (\text{mset-set } \{..< \text{dim-vec } v\})) . x = a\ \# \}$

by (*simp add: filter-mset-eq*)

also have $\dots = \text{size } \{\#a \in \# (\text{mset-set } \{..< \text{dim-vec } v\}) . x = (v\ \$\ a)\ \# \}$

by (*simp add: filter-mset-image-mset*)

finally have $\text{count-vec } v\ x = \text{card } \{a \in \{..< \text{dim-vec } v\} . x = (v\ \$\ a)\}$ **by** *simp*
thus *?thesis* **by** (*smt (verit) Collect-cong lessThan-iff*)

qed

lemma *count-vec-sum-ones*:

fixes $v :: 'a :: \{\text{ring-1}\}\ \text{vec}$

assumes $\bigwedge i. i < \text{dim-vec } v \implies v\ \$\ i = 1 \vee v\ \$\ i = 0$

shows $\text{of-nat } (\text{count-vec } v\ 1) = \text{sum-vec } v$

using *assms*

proof (*induct v*)

case *vNil*

then show *?case*

by (*simp add: vec-mset-vNil*)

next

case (*vCons a v*)

then have *lim*: $\text{dim-vec } (v\ \text{Cons } a\ v) \geq 1$

by *simp*

have $(\bigwedge i. i < \text{dim-vec } v \implies v\ \$\ i = 1 \vee v\ \$\ i = 0)$

using *vCons.prem*s **by** *force*

then have *hyp*: $\text{of-nat } (\text{count-vec } v\ 1) = \text{sum-vec } v$

using *vCons.hyps* **by** *blast*

have $\text{sum } ((\$)\ (v\ \text{Cons } a\ v)) \{0..< \text{dim-vec } (v\ \text{Cons } a\ v)\} = \text{sum-vec } (v\ \text{Cons } a\ v)$

by (*simp add: sum-vec-def*)

then have *sv*: $\text{sum } ((\$)\ (v\ \text{Cons } a\ v)) \{0..< \text{dim-vec } (v\ \text{Cons } a\ v)\} = \text{sum-vec } (v)$

+ *a*

by (*simp add: sum-vec-vCons*)

then show *?case using count-vec-vCons dim-vec-vCons-ne-0 sum-vec-vCons vCons.prem*s

by (*metis add.commute add-0 hyp of-nat-1 of-nat-add vec-index-vCons-0*)
qed

lemma *count-vec-two-elems*:

fixes $v :: 'a :: \{\text{zero-neq-one}\}$ *vec*
assumes $\bigwedge i. i < \text{dim-vec } v \implies v \$ i = 1 \vee v \$ i = 0$
shows $\text{count-vec } v \ 1 + \text{count-vec } v \ 0 = \text{dim-vec } v$

proof –

have $ss: \text{vec-set } v \subseteq \{0, 1\}$ **using** *assms* **by** (*auto simp add: vec-set-def*)

have $\text{dim-vec } v = \text{size } (\text{vec-mset } v)$

by (*simp add: mset-vec-same-size*)

have $\text{size } (\text{vec-mset } v) = (\sum x \in (\text{vec-set } v) . \text{count } (\text{vec-mset } v) \ x)$

by (*simp add: vec-mset-set size-multiset-overloaded-eq*)

also have $\dots = (\sum x \in \{0, 1\} . \text{count } (\text{vec-mset } v) \ x)$

using *size-count-mset-ss ss*

by (*metis calculation finite.emptyI finite.insertI vec-mset-set*)

finally have $\text{size } (\text{vec-mset } v) = \text{count } (\text{vec-mset } v) \ 0 + \text{count } (\text{vec-mset } v) \ 1$

by *simp*

thus *?thesis*

by (*simp add: ‹dim-vec v = size (vec-mset v)›*)

qed

lemma *count-vec-sum-zeros*:

fixes $v :: 'a :: \{\text{ring-1}\}$ *vec*
assumes $\bigwedge i. i < \text{dim-vec } v \implies v \$ i = 1 \vee v \$ i = 0$
shows $\text{of-nat } (\text{count-vec } v \ 0) = \text{of-nat } (\text{dim-vec } v) - \text{sum-vec } v$
using *count-vec-two-elems assms count-vec-sum-ones*
by (*metis add-diff-cancel-left' of-nat-add*)

lemma *count-vec-sum-ones-alt*:

fixes $v :: 'a :: \{\text{ring-1}\}$ *vec*
assumes $\text{vec-set } v \subseteq \{0, 1\}$
shows $\text{of-nat } (\text{count-vec } v \ 1) = \text{sum-vec } v$

proof –

have $\bigwedge i. i < \text{dim-vec } v \implies v \$ i = 1 \vee v \$ i = 0$ **using** *assms*

by (*meson insertE singletonD subsetD vec-setI*)

thus *?thesis using count-vec-sum-ones*

by *blast*

qed

lemma *setv-not-in-count0-iff*: $a \notin \text{set}_v \ v \iff \text{count-vec } v \ a = 0$

using *setv-vec-mset-not-in-iff*

by (*metis count-eq-zero-iff*)

lemma *sum-count-vec*:

assumes *finite (set_v v)*

shows $(\sum i \in \text{set}_v \ v. \text{count-vec } v \ i) = \text{dim-vec } v$

```

using assms proof (induct v)
  case vNil
  then show ?case
    by (simp add: count-vec-empty)
next
  case (vCons a v)
  then show ?case proof (cases a ∈ set_v v)
    case True
    have cv:  $\bigwedge x. x \in (\text{set}_v v) - \{a\} \implies \text{count-vec } (vCons a v) x = \text{count-vec } v x$ 
      using count-vec-vCons by (metis DiffD2 singletonI)
    then have  $\text{sum } (\text{count-vec } (vCons a v)) (\text{set}_v (vCons a v)) = \text{sum } (\text{count-vec } (vCons a v)) (\text{set}_v v)$ 
      using vCons-set-contains-in True by metis
    also have  $\dots = \text{count-vec } (vCons a v) a + \text{sum } (\text{count-vec } (vCons a v)) ((\text{set}_v v) - \{a\})$ 
      using sum.remove True vCons.prem1 by (metis vCons-set-contains-in)
    also have  $\dots = \text{count-vec } v a + 1 + \text{sum } (\text{count-vec } v) ((\text{set}_v v) - \{a\})$ 
      using cv count-vec-vCons by (metis sum.cong)
    also have  $\dots = 1 + \text{sum } (\text{count-vec } v) ((\text{set}_v v))$ 
      using sum.remove add commute vCons.prem1 vCons-set-contains-in True
      by (metis (no-types, opaque-lifting) ab-semigroup-add-class.add-ac(1))
    also have  $\dots = 1 + \text{dim-vec } v$  using vCons.hyps
      by (metis True vCons.prem1 vCons-set-contains-in)
    finally show ?thesis by simp
  next
  case False
  then have cv:  $\bigwedge x. x \in (\text{set}_v v) \implies \text{count-vec } (vCons a v) x = \text{count-vec } v x$ 
    using count-vec-vCons by (metis)
  have f: finite (set_v v)
    using vCons.prem1 False vCons-set-contains-add by (metis Un-infinite)
  have  $\text{sum } (\text{count-vec } (vCons a v)) (\text{set}_v (vCons a v)) = \text{count-vec } (vCons a v) a + \text{sum } (\text{count-vec } (vCons a v)) (\text{set}_v v)$ 
    using False vCons-set-contains-add
    by (metis Un-insert-right finite-Un sum.insert sup-bot-right vCons.prem1)
  also have  $\dots = \text{count-vec } v a + 1 + \text{sum } (\text{count-vec } v) ((\text{set}_v v))$ 
    using cv count-vec-vCons by (metis sum.cong)
  also have  $\dots = 1 + \text{sum } (\text{count-vec } v) ((\text{set}_v v))$ 
    using False setv-not-in-count0-iff by (metis add-0)
  finally show ?thesis using vCons.hyps f by simp
qed
qed

```

```

lemma sum-setv-subset-eq:
  assumes finite A
  assumes  $\text{set}_v v \subseteq A$ 
  shows  $(\sum i \in \text{set}_v v. \text{count-vec } v i) = (\sum i \in A. \text{count-vec } v i)$ 
proof -
  have ni:  $\bigwedge x. x \notin \text{set}_v v \implies \text{count-vec } v x = 0$ 
    by (simp add: setv-not-in-count0-iff)

```

have $(\sum i \in A. \text{count-vec } v \ i) = (\sum i \in A - (\text{set}_v \ v). \text{count-vec } v \ i) + (\sum i \in (\text{set}_v \ v). \text{count-vec } v \ i)$
using *sum.subset-diff* *assms* **by** *auto*
then show *?thesis* **using** *ni*
by *simp*
qed

lemma *sum-count-vec-subset*: $\text{finite } A \implies \text{set}_v \ v \subseteq A \implies (\sum i \in A. \text{count-vec } v \ i) = \text{dim-vec } v$
using *sum-setv-subset-eq* *sum-count-vec* *finite-subset* **by** *metis*

An abbreviation for checking if an element is in a vector

abbreviation *vec-contains* :: $'a \Rightarrow 'a \text{ vec} \Rightarrow \text{bool}$ (**infix** $\in\$\ 50$) **where**
 $a \in\$\ v \equiv a \in \text{set}_v \ v$

lemma *vec-set-mset-contains-iff*: $a \in\$\ v \longleftrightarrow a \in\# \text{vec-mset } v$
by (*simp* *add: vec-mset-def* *vec-set-def*)

lemma *vec-contains-count-gt1-iff*: $a \in\$\ v \longleftrightarrow \text{count-vec } v \ a \geq 1$
by (*simp* *add: vec-set-mset-contains-iff*)

lemma *vec-contains-obtains-index*:
assumes $a \in\$\ v$
obtains i **where** $i < \text{dim-vec } v$ **and** $v \ \$ \ i = a$
by (*metis* *assms* *vec-setE*)

lemma *vec-count-eq-list-count*: $\text{count } (\text{mset } xs) \ a = \text{count-vec } (\text{vec-of-list } xs) \ a$
by (*simp* *add: list-vec* *mset-vec-eq-mset-list*)

lemma *vec-contains-col-elements-mat*:
assumes $j < \text{dim-col } M$
assumes $a \in\$\ \text{col } M \ j$
shows $a \in \text{elements-mat } M$
proof –
have $\text{dim-vec } (\text{col } M \ j) = \text{dim-row } M$ **by** *simp*
then obtain i **where** $i < \text{dim-row } M$ **and** $(\text{col } M \ j) \ \$ \ i = a$
using *vec-setE* **by** (*metis* *assms*(2))
then have $M \ \$\ (i, j) = a$
by (*simp* *add: assms*(1))
thus *?thesis* **using** *assms*(1) *ilt*
by *blast*
qed

lemma *vec-contains-row-elements-mat*:
assumes $i < \text{dim-row } M$
assumes $a \in\$\ \text{row } M \ i$
shows $a \in \text{elements-mat } M$
proof –
have $\text{dim-vec } (\text{row } M \ i) = \text{dim-col } M$ **by** *simp*

then obtain j where $jlt: j < \dim\text{-col } M$ and $(\text{row } M \ i) \ \$ \ j = a$ using vec-setE
by $(\text{metis } \text{assms}(2))$
then have $M \ \$ \ (i, j) = a$
by $(\text{simp } \text{add: } \text{assms}(1))$
thus $?thesis$ using $\text{assms}(1)$ jlt
by blast
qed

lemma $\text{vec-contains-img}: a \in \$ \ v \implies f \ a \in \$ \ (\text{map-vec } f \ v)$
by $(\text{metis } \text{index-map-vec}(1) \ \text{index-map-vec}(2) \ \text{vec-contains-obtains-index } \text{vec-setI})$

The existing vector library contains the identity and zero vectors, but no definition of a vector where all elements are 1, as defined below

definition $\text{all-ones-vec} :: \text{nat} \Rightarrow 'a :: \{\text{zero, one}\} \ \text{vec} \ (u_v)$ where
 $u_v \ n \equiv \text{vec } n \ (\lambda \ i. \ 1)$

lemma $\text{dim-vec-all-ones}[\text{simp}]: \text{dim-vec} \ (u_v \ n) = n$
by $(\text{simp } \text{add: } \text{all-ones-vec-def})$

lemma $\text{all-ones-index} [\text{simp}]: i < n \implies u_v \ n \ \$ \ i = 1$
by $(\text{simp } \text{add: } \text{all-ones-vec-def})$

lemma $\text{dim-vec-mult-vec-mat} [\text{simp}]: \text{dim-vec} \ (v \cdot_v \ * \ A) = \text{dim-col } A$
unfolding mult-vec-mat-def by simp

lemma $\text{all-ones-vec-smult}[\text{simp}]: i < n \implies ((k :: ('a :: \{\text{one, zero, monoid-mult}\}))$
 $\cdot_v \ (u_v \ n)) \ \$ \ i = k$
by $(\text{simp } \text{add: } \text{smult-vec-def})$

Extra lemmas on existing vector operations

lemma $\text{smult-scalar-prod-sum}$:

fixes $x :: 'a :: \{\text{comm-ring-1}\}$

assumes $vx \in \text{carrier-vec } n$

assumes $vy \in \text{carrier-vec } n$

shows $(\sum \ i \in \{0..<n\} \ .((x \cdot_v \ vx) \ \$ \ i) * ((y \cdot_v \ vy) \ \$ \ i)) = x * y * (vx \cdot \ vy)$

proof –

have $\bigwedge \ i. \ i < n \implies ((x \cdot_v \ vx) \ \$ \ i) * ((y \cdot_v \ vy) \ \$ \ i) = x * y * (vx \ \$ \ i) * (vy \ \$ \ i)$

using assms by simp

then have $(\sum \ i \in \{0..<n\} \ .((x \cdot_v \ vx) \ \$ \ i) * ((y \cdot_v \ vy) \ \$ \ i)) =$
 $(\sum \ i \in \{0..<n\} \ .x * y * (vx \ \$ \ i) * (vy \ \$ \ i))$

by simp

also have $\dots = x * y * (\sum \ i \in \{0..<n\} \ .(vx \ \$ \ i) * (vy \ \$ \ i))$

using sum-distrib-left [of $x * y \ (\lambda \ i. \ (vx \ \$ \ i) * (vy \ \$ \ i)) \ \{0..<n\}$]

by $(\text{metis } (\text{no-types}, \ \text{lifting}) \ \text{mult.assoc } \text{sum.cong})$

finally have $(\sum \ i \in \{0..<n\} \ .((x \cdot_v \ vx) \ \$ \ i) * ((y \cdot_v \ vy) \ \$ \ i)) = x * y * (vx \cdot \ vy)$

using $\text{scalar-prod-def } \text{assms}$ by $(\text{metis } \text{carrier-vecD})$

thus $?thesis$ by simp

qed

lemma *scalar-prod-double-sum-fn-vec*:

fixes $c :: \text{nat} \Rightarrow ('a :: \{\text{comm-semiring-0}\})$

fixes $f :: \text{nat} \Rightarrow 'a \text{ vec}$

assumes $\bigwedge j . j < k \implies \text{dim-vec } (f j) = n$

shows $(\text{vec } n (\lambda i . \sum j = 0..<k . c j * (f j) \$ i)) \cdot (\text{vec } n (\lambda i . \sum j = 0..<k . c j * (f j) \$ i)) =$

$(\sum j1 \in \{0..<k\} . c j1 * c j1 * ((f j1) \cdot (f j1))) +$

$(\sum j1 \in \{0..<k\} . (\sum j2 \in (\{0..<k\} - \{j1\}) . c j1 * c j2 * ((f j1) \cdot (f j2))))$

proof –

have *sum-simp*: $\bigwedge j1 j2 . (\sum l \in \{0..<n\} . c j1 * (f j1) \$ l * (c j2 * (f j2) \$ l))$

=

$c j1 * c j2 * (\sum l \in \{0..<n\} . (f j1) \$ l * (f j2) \$ l)$

proof –

fix $j1 j2$

have $(\sum l \in \{0..<n\} . c j1 * (f j1) \$ l * (c j2 * (f j2) \$ l)) =$

$(\sum l \in \{0..<n\} . c j1 * c j2 * (f j1) \$ l * (f j2) \$ l)$

using *mult.commute sum.cong*

by (*smt* (z3) *ab-semigroup-mult-class.mult-ac(1)*)

then show $(\sum l \in \{0..<n\} . c j1 * (f j1) \$ l * (c j2 * (f j2) \$ l)) =$

$c j1 * c j2 * (\sum l \in \{0..<n\} . (f j1) \$ l * (f j2) \$ l)$

using *sum-distrib-left[of c j1 * c j2 λ l. (f j1) \\$ l * (f j2) \\$ l {0..<n}]*

by (*metis* (*no-types, lifting*) *mult.assoc sum.cong*)

qed

have $(\text{vec } n (\lambda i . \sum j = 0..<k . c j * (f j) \$ i)) \cdot (\text{vec } n (\lambda i . \sum j = 0..<k . c j * (f j) \$ i))$

= $(\sum l = 0..<n . (\sum j1 = 0..<k . c j1 * (f j1) \$ l) * (\sum j2 = 0..<k . c j2 * (f j2) \$ l))$

unfolding *scalar-prod-def* **by** *simp*

also have ... = $(\sum l \in \{0..<n\} . (\sum j1 \in \{0..<k\} . (\sum j2 \in \{0..<k\} . c j1 * (f j1) \$ l * (c j2 * (f j2) \$ l))))$

by (*metis* (*no-types*) *sum-product*)

also have ... = $(\sum j1 \in \{0..<k\} . (\sum j2 \in \{0..<k\} . (\sum l \in \{0..<n\} . c j1 * (f j1) \$ l * (c j2 * (f j2) \$ l))))$

using *sum-reorder-triple[of λ l j1 j2 . (c j1 * (f j1) \\$ l * (c j2 * (f j2) \\$ l)) {0..<k} {0..<k} {0..<n}]*

by *simp*

also have ... = $(\sum j1 \in \{0..<k\} . (\sum j2 \in \{0..<k\} . c j1 * c j2 * (\sum l \in \{0..<n\} . (f j1) \$ l * (f j2) \$ l)))$

using *sum-simp* **by** *simp*

also have ... = $(\sum j1 \in \{0..<k\} . (\sum j2 \in \{0..<k\} . c j1 * c j2 * ((f j1) \cdot (f j2))))$

unfolding *scalar-prod-def* **using** *dim-col assms* **by** *simp*

finally show *?thesis*

using *double-sum-split-case* **by** *fastforce*

qed

lemma *vec-prod-zero*: $(0_v \ n) \cdot (0_v \ n) = 0$

by *simp*

lemma *map-vec-compose*: $\text{map-vec } f (\text{map-vec } g \ v) = \text{map-vec } (f \circ g) \ v$
by *auto*

2.2 Matrix Extras

As with vectors, the all ones mat definition defines the concept of a matrix where all elements are 1

definition *all-ones-mat* :: $\text{nat} \Rightarrow 'a :: \{\text{zero}, \text{one}\} \text{ mat } (J_m)$ **where**
 $J_m \ n \equiv \text{mat } n \ n \ (\lambda \ (i,j). \ 1)$

lemma *all-ones-mat-index*[*simp*]: $i < \text{dim-row } (J_m \ n) \Longrightarrow j < \text{dim-col } (J_m \ n) \Longrightarrow J_m \ n \ \$\$ \ (i, j) = 1$
by (*simp add: all-ones-mat-def*)

lemma *all-ones-mat-dim-row*[*simp*]: $\text{dim-row } (J_m \ n) = n$
by (*simp add: all-ones-mat-def*)

lemma *all-ones-mat-dim-col*[*simp*]: $\text{dim-col } (J_m \ n) = n$
by (*simp add: all-ones-mat-def*)

Basic lemmas on existing matrix operations

lemma *index-mult-vec-mat*[*simp*]: $j < \text{dim-col } A \Longrightarrow (v \ v^* \ A) \ \$ \ j = v \cdot \text{col } A \ j$
by (*auto simp: mult-vec-mat-def*)

lemma *transpose-mat-mult-entries*: $i < \text{dim-row } A \Longrightarrow j < \text{dim-row } A \Longrightarrow (A * A^T) \ \$\$ \ (i, j) = (\sum k \in \{0..<(\text{dim-col } A)\}. (A \ \$\$ \ (i, k)) * (A \ \$\$ \ (j, k)))$
by (*simp add: times-mat-def scalar-prod-def*)

lemma *transpose-mat-elems*: $\text{elements-mat } A = \text{elements-mat } A^T$
by *fastforce*

lemma *row-elems-subset-mat*: $i < \text{dim-row } N \Longrightarrow \text{vec-set } (\text{row } N \ i) \subseteq \text{elements-mat } N$
by (*auto simp add: vec-set-def elements-matI*)

lemma *col-elems-subset-mat*: $i < \text{dim-col } N \Longrightarrow \text{vec-set } (\text{col } N \ i) \subseteq \text{elements-mat } N$
by (*auto simp add: vec-set-def elements-matI*)

lemma *obtain-row-index*:
assumes $r \in \text{set } (\text{rows } M)$
obtains i **where** $\text{row } M \ i = r$ **and** $i < \text{dim-row } M$
by (*metis assms in-set-conv-nth length-rows nth-rows*)

lemma *row-prop-cond*: $(\bigwedge i. i < \text{dim-row } M \Longrightarrow P (\text{row } M \ i)) \Longrightarrow r \in \text{set } (\text{rows } M) \Longrightarrow P \ r$
using *obtain-row-index* by *metis*

lemma *obtain-col-index*:

assumes $c \in \text{set } (\text{cols } M)$

obtains j **where** $\text{col } M j = c$ **and** $j < \text{dim-col } M$

by (*metis assms cols-length cols-nth obtain-set-list-item*)

lemma *col-prop-cond*: $(\bigwedge j. j < \text{dim-col } M \implies P (\text{col } M j)) \implies c \in \text{set } (\text{cols } M) \implies P c$

using *obtain-col-index* **by** *metis*

Lemmas on the *map-mat* definition

lemma *row-map-mat[simp]*:

assumes $i < \text{dim-row } A$ **shows** $\text{row } (\text{map-mat } f A) i = \text{map-vec } f (\text{row } A i)$

unfolding *map-mat-def map-vec-def* **using** *assms* **by** *auto*

lemma *map-vec-mat-rows*: $\text{map } (\text{map-vec } f) (\text{rows } M) = \text{rows } ((\text{map-mat } f) M)$

by (*simp add: map-nth-eq-conv*)

lemma *map-vec-mat-cols*: $\text{map } (\text{map-vec } f) (\text{cols } M) = \text{cols } ((\text{map-mat } f) M)$

by (*simp add: map-nth-eq-conv*)

lemma *map-mat-compose*: $\text{map-mat } f (\text{map-mat } g A) = \text{map-mat } (f \circ g) A$

by (*auto*)

lemmas *map-mat-elements = elements-mat-map*

Reasoning on sets and multisets of matrix elements

lemma *set-cols-carrier*: $A \in \text{carrier-mat } m n \implies v \in \text{set } (\text{cols } A) \implies v \in \text{carrier-vec } m$

by (*auto simp: cols-def*)

lemma *mset-cols-index-map*: $\text{image-mset } (\lambda j. \text{col } M j) (\text{mset-set } \{0..< \text{dim-col } M\}) = \text{mset } (\text{cols } M)$

by (*simp add: cols-def*)

lemma *mset-rows-index-map*: $\text{image-mset } (\lambda i. \text{row } M i) (\text{mset-set } \{0..< \text{dim-row } M\}) = \text{mset } (\text{rows } M)$

by (*simp add: rows-def*)

lemma *index-to-col-card-size-prop*:

assumes $i < \text{dim-row } M$

assumes $\bigwedge j. j < \text{dim-col } M \implies P j \longleftrightarrow Q (\text{col } M j)$

shows $\text{card } \{j . j < \text{dim-col } M \wedge P j\} = \text{size } \{\#c \in \# (\text{mset } (\text{cols } M)) . Q c \#\}$

proof –

have $\text{card } \{j . j < \text{dim-col } M \wedge P j\} = \text{size } (\text{mset-set}(\{j \in \{0..< \text{dim-col } M\} . P j\}))$

by *simp*

also have $\dots = \text{size } (\text{mset-set}(\{j \in \{0..< \text{dim-col } M\} . Q (\text{col } M j)\}))$

using *assms(2)*

by (*metis lessThan-atLeast0 lessThan-iff*)
 also have ... = size (image-mset ($\lambda j. \text{col } M j$) $\{\# j \in \# \text{ mset-set } \{0..< \text{dim-col } M\} . Q (\text{col } M j) \#\}$)
 by *simp*
 also have ... = size ($\{\# c \in \# (\text{image-mset } (\lambda j. \text{col } M j) (\text{mset-set } \{0..< \text{dim-col } M\})) . Q c \#\}$)
 using *image-mset-filter-swap*[of ($\lambda j. \text{col } M j$) $Q (\text{mset-set } \{0..< \text{dim-col } M\})$]
 by *simp*
 finally have $\text{card } \{j . j < \text{dim-col } M \wedge P j\} = \text{size } (\{\# c \in \# (\text{mset } (\text{cols } M)) . Q c \#\}$
 using *mset-cols-index-map* by *metis*
 thus ?thesis by *simp*
 qed

lemma *index-to-row-card-size-prop*:

assumes $j < \text{dim-col } M$
 assumes $\bigwedge i. i < \text{dim-row } M \implies P i \longleftrightarrow Q (\text{row } M i)$
 shows $\text{card } \{i . i < \text{dim-row } M \wedge P i\} = \text{size } \{\# r \in \# (\text{mset } (\text{rows } M)) . Q r \#\}$
 proof –
 have $\text{card } \{i . i < \text{dim-row } M \wedge P i\} = \text{size } (\text{mset-set}(\{i \in \{0..< \text{dim-row } M\} . P i\}))$
 by *simp*
 also have ... = size (mset-set($\{i \in \{0..< \text{dim-row } M\} . Q (\text{row } M i)\}$))
 using *assms*(2)
 by (*metis lessThan-atLeast0 lessThan-iff*)
 also have ... = size (image-mset ($\lambda i. \text{row } M i$) $\{\# i \in \# \text{ mset-set } \{0..< \text{dim-row } M\} . Q (\text{row } M i) \#\}$)
 by *simp*
 also have ... = size ($\{\# r \in \# (\text{image-mset } (\lambda i. \text{row } M i) (\text{mset-set } \{0..< \text{dim-row } M\})) . Q r \#\}$)
 using *image-mset-filter-swap*[of ($\lambda j. \text{row } M j$) $Q (\text{mset-set } \{0..< \text{dim-row } M\})$]
 by *simp*
 finally have $\text{card } \{j . j < \text{dim-row } M \wedge P j\} = \text{size } (\{\# c \in \# (\text{mset } (\text{rows } M)) . Q c \#\}$
 using *mset-rows-index-map* by *metis*
 thus ?thesis by *simp*
 qed

lemma *setv-row-subset-mat-elems*:

assumes $v \in \text{set } (\text{rows } M)$
 shows $\text{set}_v v \subseteq \text{elements-mat } M$
 proof (*intro subsetI*)
 fix x assume $x \in \text{set}_v v$
 then obtain i where $v = \text{row } M i$ and $i < \text{dim-row } M$
 by (*metis assms obtain-row-index*)
 then show $x \in \text{elements-mat } M$
 by (*metis* $\langle x \in \text{set}_v v \rangle \text{vec-contains-row-elements-mat}$)
 qed

```

lemma setv-col-subset-mat-elems:
  assumes  $v \in \text{set } (\text{cols } M)$ 
  shows  $\text{set}_v v \subseteq \text{elements-mat } M$ 
proof (intro subsetI)
  fix  $x$  assume  $x \in v$ 
  then obtain  $i$  where  $v = \text{col } M \ i$  and  $i < \text{dim-col } M$ 
    by (metis assms obtain-col-index)
  then show  $x \in \text{elements-mat } M$ 
    by (metis ⟨x ∈ v⟩ vec-contains-col-elements-mat)
qed

```

2.3 Vector and Matrix Homomorphism

We extend on the existing lemmas on homomorphism mappings as applied to vectors and matrices

```

context semiring-hom
begin

```

```

lemma vec-hom-smult2:
  assumes  $\text{dim-vec } v2 \leq \text{dim-vec } v1$ 
  shows  $\text{hom } (v1 \cdot v2) = \text{vec}_h v1 \cdot \text{vec}_h v2$ 
  unfolding scalar-prod-def using index-map-vec assms by (auto simp add: hom-distrib)
end

```

```

lemma map-vec-vCons:  $v\text{Cons } (f \ a) \ (\text{map-vec } f \ v) = \text{map-vec } f \ (v\text{Cons } a \ v)$ 
  by (intro eq-vecI, simp-all add: vec-index-vCons)

```

```

context inj-zero-hom
begin

```

```

lemma vec-hom-zero-iff[simp]:  $(\text{map-vec } \text{hom } x = 0_v \ n) = (x = 0_v \ n)$ 
proof –
  {
    fix  $i$ 
    assume  $i: i < n \ \text{dim-vec } x = n$ 
    hence  $\text{map-vec } \text{hom } x \ \$ \ i = 0 \longleftrightarrow x \ \$ \ i = 0$ 
      using index-map-vec(1)[of i x] by simp
    } note main = this
  show ?thesis unfolding vec-eq-iff by (simp, insert main, auto)
qed

```

```

lemma mat-hom-inj:  $\text{map-mat } \text{hom } A = \text{map-mat } \text{hom } B \implies A = B$ 
  unfolding mat-eq-iff by auto

```

```

lemma vec-hom-inj:  $\text{map-vec } \text{hom } v = \text{map-vec } \text{hom } w \implies v = w$ 
  unfolding vec-eq-iff by auto

```

```

lemma vec-hom-set-distinct-iff:
  fixes xs :: 'a vec list
  shows distinct xs  $\longleftrightarrow$  distinct (map (map-vec hom) xs)
  using vec-hom-inj by (induct xs) (auto)

lemma vec-hom-mset: image-mset hom (vec-mset v) = vec-mset (map-vec hom v)
  apply (induction v)
  apply (metis mset.simps(1) vec-mset-img-map vec-mset-vNil vec-of-list-Nil)
  by (metis mset-vec-eq-mset-list vec-list vec-mset-img-map)

lemma vec-hom-set: hom ` set_v v = set_v (map-vec hom v)
proof (induct v)
  case vNil
  then show ?case by (metis image-mset-empty set-image-mset vec-hom-zero-iff
vec-mset-set vec-mset-vNil zero-vec-zero)
next
  case (vCons a v)
  have hom ` set_v (vCons a v) = hom ` ({a}  $\cup$  set_v v)
  by (metis Un-commute insert-absorb insert-is-Un vCons-set-contains-add vCons-set-contains-in)

  also have ... = {hom a}  $\cup$  (hom ` (set_v v)) by simp
  also have ... = {hom a}  $\cup$  (set_v (map-vec hom v)) using vCons.hyps by simp
  also have ... = set_v (vCons (hom a) (map-vec hom v))
  by (metis Un-commute insert-absorb insert-is-Un vCons-set-contains-add vCons-set-contains-in)

  finally show ?case using map-vec-vCons
  by metis
qed

end

```

2.4 Zero One injections and homomorphisms

Define a locale to encapsulate when a function is injective on a certain set (i.e. not a universal homomorphism for the type)

```

locale injective-lim =
  fixes A :: 'a set
  fixes f :: 'a  $\Rightarrow$  'b assumes injectivity-lim:  $\bigwedge x y. x \in A \Longrightarrow y \in A \Longrightarrow f x = f y \Longrightarrow x = y$ 
begin
  lemma eq-iff[simp]:  $x \in A \Longrightarrow y \in A \Longrightarrow f x = f y \longleftrightarrow x = y$  using injectivity-lim
by auto
  lemma inj-on-f: inj-on f A by (auto intro: inj-onI)
end

```

```

sublocale injective  $\subseteq$  injective-lim Univ
by(unfold-locales) simp

```

```

context injective-lim
begin

lemma mat-hom-inj-lim:
  assumes elements-mat  $M \subseteq A$  and elements-mat  $N \subseteq A$ 
  shows map-mat  $f M = \text{map-mat } f N \implies M = N$ 
  unfolding mat-eq-iff apply auto
  using assms injectivity-lim by blast

lemma vec-hom-inj-lim: assumes set_v  $v \subseteq A$  set_v  $w \subseteq A$ 
  shows map-vec  $f v = \text{map-vec } f w \implies v = w$ 
  unfolding vec-eq-iff apply (auto)
  using vec-setI in-mono assms injectivity-lim by metis

lemma lim-inj-hom-count-vec:
  assumes set_v  $v \subseteq A$ 
  assumes  $x \in A$ 
  shows count-vec  $v x = \text{count-vec } (\text{map-vec } f v) (f x)$ 
using assms proof (induct v)
  case vNil
  have  $(\text{map-vec } f \text{vNil}) = \text{vNil}$  by auto
  then show ?case
    by (smt (verit) count-vec-vNil)
next
  case (vCons a v)
  have  $1: \text{map-vec } f (\text{vCons } a v) = \text{vCons } (f a) (\text{map-vec } f v)$ 
    by (simp add: map-vec-vCons)
  then show ?case proof (cases a = x)
    case True
    have  $\text{count-vec } (\text{vCons } a v) x = \text{count-vec } v x + 1$ 
      by (simp add: True count-vec-vCons)
    then show ?thesis using  $Un\text{-subset-iff } 1$  count-vec-vCons vCons.hyps vCons.prem(1)
       $vCons.prem(2)$  vCons-set-contains-add vCons-set-contains-in
      by metis
    next
    case False
    then have  $\text{count-vec } (\text{vCons } a v) x = \text{count-vec } v x$ 
      by (simp add: count-vec-vCons)
    then show ?thesis using  $1$  Un-empty-right Un-insert-right count-vec-vCons
      insert-absorb insert-subset
       $vCons.hyps vCons.prem(1) vCons.prem(2) vCons-set-contains-add$ 
       $vCons-set-contains-in$ 
      by (metis (no-types, lifting) injectivity-lim)
  qed
qed

lemma vec-hom-lim-set-distinct-iff:

```

fixes $xs :: 'a \text{ vec list}$
assumes $\bigwedge v . v \in \text{set } (xs) \implies \text{set}_v v \subseteq A$
shows $\text{distinct } xs \longleftrightarrow \text{distinct } (\text{map } (\text{map-vec } f) xs)$
using $\text{assms } \text{vec-hom-inj-lim}$ **by** $(\text{induct } xs, \text{simp-all})$ $(\text{metis } (\text{no-types}, \text{lifting}) \text{image-iff})$

lemma $\text{mat-rows-hom-lim-distinct-iff}$:
assumes $\text{elements-mat } M \subseteq A$
shows $\text{distinct } (\text{rows } M) \longleftrightarrow \text{distinct } (\text{map } (\text{map-vec } f) (\text{rows } M))$
apply $(\text{intro } \text{vec-hom-lim-set-distinct-iff})$
using $\text{setv-row-subset-mat-elems assms}$ **by** blast

lemma $\text{mat-cols-hom-lim-distinct-iff}$:
assumes $\text{elements-mat } M \subseteq A$
shows $\text{distinct } (\text{cols } M) \longleftrightarrow \text{distinct } (\text{map } (\text{map-vec } f) (\text{cols } M))$
apply $(\text{intro } \text{vec-hom-lim-set-distinct-iff})$
using $\text{setv-col-subset-mat-elems assms}$ **by** blast

end

locale $\text{inj-on-01-hom} = \text{zero-hom} + \text{one-hom} + \text{injective-lim } \{0, 1\} \text{ hom}$
begin

lemma inj-0-iff : $x \in \{0, 1\} \implies \text{hom } x = 0 \longleftrightarrow x = 0$
by $(\text{metis } \text{hom-zero } \text{insertI1 } \text{local.eq-iff})$

lemma inj-1-iff : $x \in \{0, 1\} \implies \text{hom } x = 1 \longleftrightarrow x = 1$
using inj-0-iff **by** fastforce

end

context zero-neq-one
begin

definition $\text{of-zero-neq-one} :: 'b :: \{ \text{zero-neq-one} \} \Rightarrow 'a$ **where**
 $\text{of-zero-neq-one } x \equiv \text{if } (x = 0) \text{ then } 0 \text{ else } 1$

lemma of-zero-neq-one-1 $[\text{simp}]$: $\text{of-zero-neq-one } 1 = 1$
by $(\text{simp add: } \text{of-zero-neq-one-def})$

lemma of-zero-neq-one-0 $[\text{simp}]$: $\text{of-zero-neq-one } 0 = 0$
by $(\text{simp add: } \text{of-zero-neq-one-def})$

lemma $\text{of-zero-neq-one-0-iff}$ $[\text{iff}]$: $\text{of-zero-neq-one } x = 0 \longleftrightarrow x = 0$
by $(\text{simp add: } \text{of-zero-neq-one-def})$

lemma $\text{of-zero-neq-one-lim-eq}$: $x \in \{0, 1\} \implies y \in \{0, 1\} \implies \text{of-zero-neq-one } x = \text{of-zero-neq-one } y \longleftrightarrow x = y$
by $(\text{auto simp add: } \text{of-zero-neq-one-def})$

end

interpretation *of-zero-hom*: *zero-hom-0 of-zero-neq-one*
by (*unfold-locales*) (*simp-all*)

interpretation *of-injective-lim*: *injective-lim {0, 1} of-zero-neq-one*
by (*unfold-locales*) (*simp-all add: of-zero-neq-one-lim-eq*)

interpretation *of-inj-on-01-hom*: *inj-on-01-hom of-zero-neq-one*
by (*unfold-locales*) (*simp-all add: of-zero-neq-one-lim-eq*)

We want the ability to transform any 0-1 vector or matrix to another 'c type

definition *lift-01-vec* :: 'b :: {zero-neq-one} vec \Rightarrow 'c :: {zero-neq-one} vec **where**
lift-01-vec v \equiv *map-vec of-zero-neq-one v*

lemma *lift-01-vec-simp*[*simp*]: *dim-vec (lift-01-vec v) = dim-vec v*
i < dim-vec v \implies (lift-01-vec v) \$ i = of-zero-neq-one (v \$ i)
by (*simp-all add: lift-01-vec-def*)

lemma *lift-01-vec-count*:
assumes *set_v*, *v* \subseteq {0, 1}
assumes *x* \in {0, 1}
shows *count-vec v x = count-vec (lift-01-vec v) (of-zero-neq-one x)*
using *of-injective-lim.lim-inj-hom-count-vec*
by (*metis assms(1) assms(2) lift-01-vec-def*)

definition *lift-01-mat* :: 'b :: {zero-neq-one} mat \Rightarrow 'c :: {zero-neq-one} mat **where**

lift-01-mat M \equiv *map-mat of-zero-neq-one M*

lemma *lift-01-mat-simp*[*simp*]: *dim-row (lift-01-mat M) = dim-row M*
dim-col (lift-01-mat M) = dim-col M
i < dim-row M \implies j < dim-col M \implies (lift-01-mat M) \$\$ (i, j) = of-zero-neq-one (M \$\$ (i, j))
by (*simp-all add: lift-01-mat-def*)

lemma *lift-01-mat-carrier*: *lift-01-mat M* \in *carrier-mat (dim-row M) (dim-col M)*
using *lift-01-mat-def by auto*

end

3 Micellaneous Design Extras

Extension's to the author's previous entry on Design Theory

theory *Design-Extras* **imports** *Set-Multiset-Extras Design-Theory.BIBD*

begin

3.1 Extensions to existing Locales and Properties

Extend lemmas on intersection number

lemma *inter-num-max-bound*:

assumes *finite b1 finite b2*

shows $b1 \mid \cap \mid b2 \leq \text{card } b1$ $b1 \mid \cap \mid b2 \leq \text{card } b2$

by (*simp-all add: assms intersection-number-def card-mono*)

lemma *inter-eq-blocks-eq-card*: $\text{card } b1 = \text{card } b2 \implies \text{finite } b1 \implies \text{finite } b2 \implies b1 \mid \cap \mid b2 = \text{card } b1$

$\implies b1 = b2$

using *equal-card-inter-fin-eq-sets intersection-number-def* **by** (*metis*)

lemma *inter-num-of-eq-blocks*: $b1 = b2 \implies b1 \mid \cap \mid b2 = \text{card } b1$

by (*simp add: intersection-number-def*)

lemma *intersect-num-same-eq-size*[*simp*]: $bl \mid \cap \mid bl = \text{card } bl$

by (*simp add: intersection-number-def*)

lemma *index-lt-rep-general*: $x \in ps \implies B \text{ index } ps \leq B \text{ rep } x$

by (*simp add: points-index-def point-replication-number-def*)

(*metis filter-filter-mset-cond-simp size-filter-mset-lesseq subset-iff*)

context *incidence-system*

begin

lemma *block-size-alt*:

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

shows $\text{card } bl = \text{card } \{x \in \mathcal{V} . x \in bl\}$

proof –

have $\bigwedge x. x \in bl \implies x \in \mathcal{V}$ **using** *wellformed assms* **by** *auto*

thus *?thesis*

by (*metis (no-types, lifting) Collect-cong Collect-mem-eq*)

qed

lemma *complement-image*: $\mathcal{B}^C = \text{image-mset block-complement } \mathcal{B}$

by (*simp add: complement-blocks-def*)

lemma *point-in-block-rep-min-iff*:

assumes $x \in \mathcal{V}$

shows $\exists bl . bl \in \# \mathcal{B} \wedge x \in bl \iff (B \text{ rep } x > 0)$

using *rep-number-g0-exists*

by (*metis block-complement-elem-iff block-complement-inv wellformed*)

lemma *points-inter-num-rep*:

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

shows $\text{card } \{v \in \mathcal{V} . v \in b1 \wedge v \in b2\} = b1 \mid \cap \mid b2$

proof –
have $\bigwedge x. x \in b1 \cap b2 \implies x \in \mathcal{V}$ **using** *wellformed assms* **by** *auto*
then have $\{v \in \mathcal{V} . v \in (b1 \cap b2)\} = (b1 \cap b2)$
by *blast*
then have $\text{card } \{v \in \mathcal{V} . v \in b1 \wedge v \in b2\} = \text{card } (b1 \cap b2)$
by *simp*
thus *?thesis* **using** *assms intersection-number-def* **by** *metis*
qed

Extensions on design operation lemmas

lemma *del-block-b*:
 $bl \in \# \mathcal{B} \implies \text{size } (\text{del-block } bl) = b - 1$
 $bl \notin \# \mathcal{B} \implies \text{size } (\text{del-block } bl) = b$
by (*simp-all add: del-block-def size-Diff-singleton*)

lemma *del-block-points-index*:
assumes $ps \subseteq \mathcal{V}$
assumes $\text{card } ps = 2$
assumes $bl \in \# \mathcal{B}$
shows $ps \subseteq bl \implies \text{points-index } (\text{del-block } bl) ps = \text{points-index } \mathcal{B} ps - 1$
 $\neg (ps \subseteq bl) \implies \text{points-index } (\text{del-block } bl) ps = \text{points-index } \mathcal{B} ps$

proof –
assume $ps \subseteq bl$
then show $\text{points-index } (\text{del-block } bl) ps = \text{points-index } \mathcal{B} ps - 1$
using *point-index-diff del-block-def*
by (*metis assms(3) insert-DiffM2 points-index-singleton*)
next
assume $\neg ps \subseteq bl$
then show $\text{del-block } bl \text{ index } ps = \mathcal{B} \text{ index } ps$
using *point-index-diff del-block-def*
by (*metis add-block-def add-block-index-not-in assms(3) insert-DiffM2*)
qed

end

Extensions to properties of design sub types

context *finite-incidence-system*
begin

lemma *complete-block-size-eq-points*: $bl \in \# \mathcal{B} \implies \text{card } bl = v \implies bl = \mathcal{V}$
using *wellformed* **by** (*simp add: card-subset-eq finite-sets*)

lemma *complete-block-all-subsets*: $bl \in \# \mathcal{B} \implies \text{card } bl = v \implies ps \subseteq \mathcal{V} \implies ps \subseteq bl$
using *complete-block-size-eq-points* **by** *auto*

lemma *del-block-complete-points-index*: $ps \subseteq \mathcal{V} \implies \text{card } ps = 2 \implies bl \in \# \mathcal{B} \implies \text{card } bl = v \implies \text{points-index } (\text{del-block } bl) ps = \text{points-index } \mathcal{B} ps - 1$

```

using complete-block-size-eq-points del-block-points-index(1) by blast

end

context design
begin

lemma block-num-rep-bound: b ≤ (∑ x ∈ V. B rep x)
proof -
  have exists: ∧ bl. bl ∈# B ⇒ (∃ x ∈ V . bl ∈# {#b ∈# B. x ∈ b#}) using
  wellformed
  using blocks-nempty by fastforce
  then have bss: B ⊆# ∑# (image-mset (λ v. {#b ∈# B. v ∈ b#}) (mset-set
  V))
  proof (intro mset-subset-eqI)
    fix bl
    show count B bl ≤ count (∑ v ∈# mset-set V. filter-mset ((∈) v) B) bl
    proof (cases bl ∈# B)
      case True
      then obtain x where xin: x ∈ V and blin: bl ∈# filter-mset ((∈) x) B using
      exists by auto
      then have eq: count B bl = count (filter-mset ((∈) x) B) bl by simp
      have (∑ v ∈# mset-set V. filter-mset ((∈) v) B) = (filter-mset ((∈) x) B) +
      (∑ v ∈# (mset-set V) - {#x#}. filter-mset ((∈) v) B)
      using xin by (simp add: finite-sets mset-set.remove)
      then have count (∑ v ∈# mset-set V. filter-mset ((∈) v) B) bl = count
      (filter-mset ((∈) x) B) bl
      + count (∑ v ∈# (mset-set V) - {#x#}. filter-mset ((∈) v) B) bl
      by simp
      then show ?thesis using eq by linarith
    next
      case False
      then show ?thesis by (metis count-eq-zero-iff le0)
    qed
  qed
  have (∑ x ∈ V. B rep x) = (∑ x ∈ V. size ({#b ∈# B. x ∈ b#}))
  by (simp add: point-replication-number-def)
  also have ... = (∑ x ∈# (mset-set V). size ({#b ∈# B. x ∈ b#}))
  by (simp add: sum-unfold-sum-mset)
  also have ... = (∑ x ∈# (image-mset (λ v. {#b ∈# B. v ∈ b#}) (mset-set V))
  . size x)
  by auto
  finally have (∑ x ∈ V. B rep x) = size (∑# (image-mset (λ v. {#b ∈# B. v
  ∈ b#}) (mset-set V)))
  using size-big-union-sum by metis
  then show ?thesis using bss
  by (simp add: size-mset-mono)
qed

```

```

end

context proper-design
begin

lemma del-block-proper:
  assumes b > 1
  shows proper-design  $\mathcal{V}$  (del-block bl)
proof -
  interpret d: design  $\mathcal{V}$  (del-block bl)
  using delete-block-design by simp
  have d.b > 0 using del-block-b assms
  by (metis b-positive zero-less-diff)
  then show ?thesis by (unfold-locales) (auto)
qed

end

context simple-design
begin

lemma inter-num-lt-block-size-strict:
  assumes bl1  $\in\#\mathcal{B}$ 
  assumes bl2  $\in\#\mathcal{B}$ 
  assumes bl1  $\neq$  bl2
  assumes card bl1 = card bl2
  shows bl1  $\cap$  bl2 < card bl1 bl1  $\cap$  bl2 < card bl2
proof -
  have lt: bl1  $\cap$  bl2  $\leq$  card bl1 using finite-blocks
  by (simp add:  $\langle$ bl1  $\in\#\mathcal{B}\rangle$   $\langle$ bl2  $\in\#\mathcal{B}\rangle$  inter-num-max-bound(1))
  have ne: bl1  $\cap$  bl2  $\neq$  card bl1
  proof (rule ccontr, simp)
    assume bl1  $\cap$  bl2 = card bl1
    then have bl1 = bl2 using assms(4) inter-eq-blocks-eq-card assms(1) assms(2)
    finite-blocks
    by blast
  then show False using assms(3) by simp
  qed
  then show bl1  $\cap$  bl2 < card bl1 using lt by simp
  have bl1  $\cap$  bl2  $\neq$  card bl2 using ne by (simp add: assms(4))
  then show bl1  $\cap$  bl2 < card bl2 using lt assms(4) by simp
qed

lemma block-mset-distinct: distinct-mset  $\mathcal{B}$  using simple
  by (simp add: distinct-mset-def)

end

context constant-rep-design

```

begin

lemma *index- lt -const- rep :*

assumes $ps \subseteq \mathcal{V}$

assumes $ps \neq \{\}$

shows $\mathcal{B} \text{ index } ps \leq r$

proof –

obtain x **where** $xin: x \in ps$ **using** *assms* **by** *auto*

then have $\mathcal{B} \text{ rep } x = r$

by (*meson assms(1) in-mono rep-number-alt-def-all*)

thus *?thesis* **using** *index- lt - rep -general* xin **by** *auto*

qed

end

context *t-wise-balance*

begin

lemma *obtain- t -subset-with-point:*

assumes $x \in \mathcal{V}$

obtains ps **where** $ps \subseteq \mathcal{V}$ **and** $\text{card } ps = t$ **and** $x \in ps$

proof (*cases t = 1*)

case *True*

have $\{x\} \subseteq \mathcal{V}$ $\text{card } \{x\} = 1$ $x \in \{x\}$

using *assms* **by** *simp-all*

then show *?thesis*

using *True* **that** **by** *blast*

next

case *False*

have $t - 1 \leq \text{card } (\mathcal{V} - \{x\})$

by (*simp add: assms diff-le-mono finite-sets t- lt -order*)

then obtain ps' **where** $psss: ps' \subseteq (\mathcal{V} - \{x\})$ **and** $psc: \text{card } ps' = t - 1$

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

then have $xs: (\text{insert } x \text{ } ps') \subseteq \mathcal{V}$

using *assms* **by** *blast*

have $xnotin: x \notin ps'$ **using** *psss*

by *blast*

then have $\text{card } (\text{insert } x \text{ } ps') = \text{Suc } (\text{card } ps')$

by (*meson <insert x ps' \subseteq \mathcal{V} > finite-insert card-insert-disjoint finite-sets finite-subset*)

then have $\text{card } (\text{insert } x \text{ } ps') = \text{card } ps' + 1$

by *presburger*

then have $xc: \text{card } (\text{insert } x \text{ } ps') = t$ **using** *psc*

using *add.commute add-diff-inverse t-non-zero* **by** *linarith*

have $x \in (\text{insert } x \text{ } ps')$ **by** *simp*

then show *?thesis* **using** xs xc **that** **by** *blast*

qed

lemma *const-index- lt - rep :*

assumes $x \in \mathcal{V}$
shows $\Lambda_t \leq \mathcal{B}$ *rep* x
proof –
obtain ps **where** $psin: ps \subseteq \mathcal{V}$ **and** $card\ ps = t$ **and** $xin: x \in ps$
using *assms t-lt-order obtain-t-subset-with-point* **by** *auto*
then have \mathcal{B} *index* $ps = \Lambda_t$ **using** *balanced* **by** *simp*
thus *?thesis* **using** *index-lt-rep-general xin*
by *(meson)*
qed
end

context *pairwise-balance*
begin

lemma *index-zero-iff*: $\Lambda = 0 \longleftrightarrow (\forall bl \in \# \mathcal{B} . card\ bl = 1)$
proof *(auto)*
fix bl **assume** $l0: \Lambda = 0$ **assume** $blin: bl \in \# \mathcal{B}$
have $card\ bl = 1$
proof *(rule ccontr)*
assume $card\ bl \neq 1$
then have $card\ bl \geq 2$ **using** *block-size-gt-0*
by *(metis Suc-1 Suc-leI blin less-one nat-neq-iff)*
then obtain ps **where** $psss: ps \subseteq bl$ **and** $pscard: card\ ps = 2$
by *(meson obtain-subset-with-card-n)*
then have $psin: \mathcal{B}$ *index* $ps \geq 1$
using *blin points-index-count-min* **by** *auto*
have $ps \subseteq \mathcal{V}$ **using** *wellformed psss blin* **by** *auto*
then show *False* **using** *balanced l0 psin pscard* **by** *auto*
qed
thus $card\ bl = (Suc\ 0)$ **by** *simp*
next
assume $a: \forall bl \in \# \mathcal{B} . card\ bl = Suc\ 0$
obtain ps **where** $psss: ps \subseteq \mathcal{V}$ **and** $ps2: card\ ps = 2$
by *(meson obtain-t-subset-points)*
then have $\bigwedge bl. bl \in \# \mathcal{B} \implies (card\ ps > card\ bl)$ **using** a
by *simp*
then have $cond: \bigwedge bl. bl \in \# \mathcal{B} \implies \neg(ps \subseteq bl)$
by *(metis card-mono finite-blocks le-antisym less-imp-le-nat less-not-refl3)*
have \mathcal{B} *index* $ps = size\ \{ \# bl \in \# \mathcal{B} . ps \subseteq bl \# \}$ **by** *(simp add:points-index-def)*
then have \mathcal{B} *index* $ps = size\ \{ \# \}$ **using** $cond$
by *(metis points-index-0-iff size-empty)*
thus $\Lambda = 0$ **using** $psss\ ps2$ **balanced** **by** *simp*
qed

lemma *count-complete-lt-balance*: $count\ \mathcal{B}\ \mathcal{V} \leq \Lambda$
proof *(rule ccontr)*
assume $a: \neg count\ \mathcal{B}\ \mathcal{V} \leq \Lambda$
then have $assm: count\ \mathcal{B}\ \mathcal{V} > \Lambda$

by *simp*
 then have *gt*: $\text{size } \{\# \text{ bl} \in \# \mathcal{B} . \text{bl} = \mathcal{V}\# \} > \Lambda$
 by (*simp add: count-size-set-repr*)
 obtain *ps* where *psss*: $ps \subseteq \mathcal{V}$ and *pscard*: $\text{card } ps = 2$ using *t-lt-order*
 by (*meson obtain-t-subset-points*)
 then have $\{\# \text{ bl} \in \# \mathcal{B} . \text{bl} = \mathcal{V}\# \} \subseteq \# \{\# \text{ bl} \in \# \mathcal{B} . ps \subseteq \text{bl} \# \}$
 by (*metis a balanced le-refl points-index-count-min*)
 then have $\text{size } \{\# \text{ bl} \in \# \mathcal{B} . \text{bl} = \mathcal{V}\# \} \leq \mathcal{B} \text{ index } ps$
 using *points-index-def*[of \mathcal{B} *ps*] *size-mset-mono* by *simp*
 thus *False* using *pscard psss balanced gt* by *auto*
 qed

lemma *eq-index-rep-imp-complete*:

assumes $\Lambda = \mathcal{B} \text{ rep } x$
 assumes $x \in \mathcal{V}$
 assumes $\text{bl} \in \# \mathcal{B}$
 assumes $x \in \text{bl}$
 shows $\text{card } \text{bl} = \mathcal{V}$
 proof –
 have $\bigwedge y. y \in \mathcal{V} \implies y \neq x \implies \text{card } \{x, y\} = 2 \wedge \{x, y\} \subseteq \mathcal{V}$ using *assms* by *simp*
 then have *size-eq*: $\bigwedge y. y \in \mathcal{V} \implies y \neq x \implies \text{size } \{\# b \in \# \mathcal{B} . \{x, y\} \subseteq b\# \}$
 = $\text{size } \{\# b \in \# \mathcal{B} . x \in b\# \}$
 using *point-replication-number-def* *balanced points-index-def* *assms* by *metis*
 have $\bigwedge y b. y \in \mathcal{V} \implies y \neq x \implies b \in \# \mathcal{B} \implies \{x, y\} \subseteq b \longrightarrow x \in b$ by *simp*
 then have $\bigwedge y. y \in \mathcal{V} \implies y \neq x \implies \{\# b \in \# \mathcal{B} . \{x, y\} \subseteq b\# \} \subseteq \# \{\# b \in \# \mathcal{B} . x \in b\# \}$
 using *multiset-filter-mono2* *assms* by *auto*
 then have *eq-sets*: $\bigwedge y. y \in \mathcal{V} \implies y \neq x \implies \{\# b \in \# \mathcal{B} . \{x, y\} \subseteq b\# \} = \{\# b \in \# \mathcal{B} . x \in b\# \}$
 using *size-eq* by (*smt (z3) Diff-eq-empty-iff-mset cancel-comm-monoid-add-class.diff-cancel*

 $\text{size-Diff-submset size-empty size-eq-0-iff-empty subset-mset.antisym}$)
 have $\text{bl} \in \# \{\# b \in \# \mathcal{B} . x \in b\# \}$ using *assms* by *simp*
 then have $\bigwedge y. y \in \mathcal{V} \implies y \neq x \implies \{x, y\} \subseteq \text{bl}$ using *eq-sets*
 by (*metis (no-types, lifting) Multiset.set-mset-filter mem-Collect-eq*)
 then have $\bigwedge y. y \in \mathcal{V} \implies y \in \text{bl}$ using *assms* by *blast*
 then have $\text{bl} = \mathcal{V}$ using *wellformed assms(3)* by *blast*
 thus *?thesis* by *simp*
 qed

lemma *incomplete-index-strict-lt-rep*:

assumes $\bigwedge \text{bl}. \text{bl} \in \# \mathcal{B} \implies \text{incomplete-block } \text{bl}$
 assumes $x \in \mathcal{V}$
 assumes $\Lambda > 0$
 shows $\Lambda < \mathcal{B} \text{ rep } x$
 proof (*rule ccontr*)
 assume $\neg (\Lambda < \mathcal{B} \text{ rep } x)$
 then have *a*: $\Lambda \geq \mathcal{B} \text{ rep } x$

by *simp*
then have $\Lambda = \mathcal{B}$ *rep* x **using** *const-index- lt -rep*
using *assms(2)* *le-antisym* **by** *blast*
then obtain bl **where** $xin: x \in bl$ **and** $blin: bl \in \# \mathcal{B}$
by (*metis* *assms(3)* *rep-number- $g0$ -exists*)
thus *False* **using** *assms* *eq-index- rep -imp-complete* *incomplete-alt-size*
using $\langle \Lambda = \mathcal{B} \text{ rep } x \rangle$ *nat-less-le* **by** *blast*
qed

Construct new PBD's from existing PBD's

lemma *remove-complete-block-pbd:*

assumes $b \geq 2$
assumes $bl \in \# \mathcal{B}$
assumes $card\ bl = v$
shows *pairwise-balance* \mathcal{V} (*del-block* bl) ($\Lambda - 1$)
proof –
interpret *pd: proper-design* \mathcal{V} (*del-block* bl) **using** *assms(1)* *del-block-proper* **by**
simp
show *?thesis* **using** *t- lt -order* *assms* *del-block-complete-points-index*
by (*unfold-locales*) (*simp-all*)
qed

lemma *remove-complete-block-pbd-alt:*

assumes $b \geq 2$
assumes $bl \in \# \mathcal{B}$
assumes $bl = \mathcal{V}$
shows *pairwise-balance* \mathcal{V} (*del-block* bl) ($\Lambda - 1$)
using *remove-complete-block-pbd* *assms* **by** *blast*

lemma *b-gt-index:* $b \geq \Lambda$

proof (*rule ccontr*)
assume $bll: \neg b \geq \Lambda$
obtain ps **where** $card\ ps = 2$ **and** $ps \subseteq \mathcal{V}$ **using** *t- lt -order*
by (*meson* *obtain-t-subset-points*)
then have $size\ \{\#bl \in \# \mathcal{B}. ps \subseteq bl\#\} = \Lambda$ **using** *balanced* **by** (*simp* *add:*
points-index-def)
thus *False* **using** bll **by** *auto*
qed

lemma *remove-complete-blocks-set-pbd:*

assumes $x < \Lambda$
assumes $size\ A = x$
assumes $A \subset \# \mathcal{B}$
assumes $\bigwedge a. a \in \# A \implies a = \mathcal{V}$
shows *pairwise-balance* \mathcal{V} ($\mathcal{B} - A$) ($\Lambda - x$)
using *assms* **proof** (*induct* x *arbitrary: A*)
case 0
then have *beq:* $\mathcal{B} - A = \mathcal{B}$ **by** *simp*
have *pairwise-balance* \mathcal{V} \mathcal{B} Λ **by** (*unfold-locales*)

```

then show ?case using beq by simp
next
case (Suc x)
then have size A > 0 by simp
let ?A' = A - {#V#}
have ss: ?A'  $\subset$  # B
  using Suc.prem(3) by (metis diff-subset-eq-self subset-mset.le-less-trans)
have sx: size ?A' = x
  by (metis Suc.prem(2) Suc.prem(4) Suc-inject size-Suc-Diff1 size-eq-Suc-imp-elem)
have xlt: x <  $\Lambda$ 
  by (simp add: Suc.prem(1) Suc-lessD)
have av:  $\bigwedge a. a \in \# ?A' \implies a = V$  using Suc.prem(4)
  by (meson in-remove1-mset-neq)
then interpret pbd: pairwise-balance  $\mathcal{V} (B - ?A') (\Lambda - x)$  using Suc.hyps sx
ss xlt by simp
have Suc x < b using Suc.prem(3)
  by (metis Suc.prem(2) mset-subset-size)
then have b - x  $\geq$  2
  by linarith
then have bgt: size (B - ?A')  $\geq$  2 using ss size-Diff-submset
  by (metis subset-msetE sx)
have ar: add-mset  $\mathcal{V} (remove1-mset \mathcal{V} A) = A$  using Suc.prem(2) Suc.prem(4)
  by (metis insert-DiffM size-eq-Suc-imp-elem)
then have db: pbd.del-block  $\mathcal{V} = B - A$  by (simp add: pbd.del-block-def)
then have B - ?A' = B - A + {#V#} using Suc.prem(2) Suc.prem(4)
  by (metis (no-types, lifting) Suc.prem(3) ar add-diff-cancel-left' add-mset-add-single
add-right-cancel
  pbd.del-block-def remove-1-mset-id-iff-notin ss subset-mset.lessE trivial-add-mset-remove-iff)

then have  $\mathcal{V} \in \# (B - ?A')$  by simp
then have pairwise-balance  $\mathcal{V} (B - A) (\Lambda - (Suc x))$  using db bgt diff-Suc-eq-diff-pred

  diff-commute pbd.remove-complete-block-pbd-alt by presburger
then show ?case by simp
qed

lemma remove-all-complete-blocks-pbd:
  assumes count B  $\mathcal{V} <$   $\Lambda$ 
  shows pairwise-balance  $\mathcal{V} (removeAll-mset \mathcal{V} B) (\Lambda - (count B \mathcal{V}))$  (is pairwise-balance  $\mathcal{V} ?B ?\Lambda$ )
proof -
let ?A = replicate-mset (count B  $\mathcal{V}$ )  $\mathcal{V}$ 
let ?x = size ?A
have blt: count B  $\mathcal{V} \neq$  b using b-gt-index assms
  by linarith
have xeq: ?x = count B  $\mathcal{V}$  by simp
have av:  $\bigwedge a. a \in \# ?A \implies a = V$ 
  by (metis in-replicate-mset)
have ?A  $\subseteq$  # B

```

```

    by (meson count-le-replicate-mset-subset-eq le-eq-less-or-eq)
  then have ?A  $\subset$  #  $\mathcal{B}$  using blt
    by (metis subset-mset.nless-le xeq)
  thus ?thesis using assms av xeq remove-complete-blocks-set-pbd
    by presburger
qed

end

```

```

context bibd
begin
lemma symmetric-bibdIII:  $r = k \implies$  symmetric-bibd  $\mathcal{V} \mathcal{B} k \Lambda$ 
  using necessary-condition-one symmetric-condition-1 by (unfold-locales) (simp)
end

```

3.2 New Design Locales

We establish a number of new locales and link them to the existing locale hierarchy in order to reason in contexts requiring specific combinations of contexts

Regular t -wise balance

```

locale regular-t-wise-balance = t-wise-balance + constant-rep-design
begin

```

```

lemma reg-index- $lt$ -rep:
  shows  $\Lambda_t \leq r$ 
proof -
  obtain  $ps$  where  $psin$ :  $ps \subseteq \mathcal{V}$  and  $pst$ :  $card\ ps = t$ 
    by (metis obtain-t-subset-points)
  then have  $ne$ :  $ps \neq \{\}$  using t-non-zero by auto
  then have  $\mathcal{B}$  index  $ps = \Lambda_t$  using balanced  $pst\ psin$  by simp
  thus ?thesis using index- $lt$ -const-rep
    using  $ne\ psin$  by auto
qed

```

```

end

```

```

locale regular-pairwise-balance = regular-t-wise-balance  $\mathcal{V} \mathcal{B} 2 \Lambda r$  + pairwise-balance
 $\mathcal{V} \mathcal{B} \Lambda$ 
  for  $\mathcal{V}$  and  $\mathcal{B}$  and  $\Lambda$  and  $r$ 

```

Const Intersect Design

This is the dual of a balanced design, and used extensively in the remaining formalisation

```

locale const-intersect-design = proper-design +
  fixes  $m :: nat$ 
  assumes const-intersect:  $b1 \in \# \mathcal{B} \implies b2 \in \# (\mathcal{B} - \{\#b1\}) \implies b1 \cap b2 = m$ 

```

sublocale *symmetric-bibd* \subseteq *const-intersect-design* $\mathcal{V} \mathcal{B} \Lambda$
by (*unfold-locales*) (*simp*)

context *const-intersect-design*
begin

lemma *inter-num-le-block-size*:

assumes $bl \in \# \mathcal{B}$
assumes $b \geq 2$
shows $m \leq \text{card } bl$
proof (*rule ccontr*)
assume $a: \neg (m \leq \text{card } bl)$
obtain bl' **where** $blin: bl' \in \# \mathcal{B} - \{\#bl\# \}$
using *assms* **by** (*metis add-mset-add-single diff-add-inverse2 diff-is-0-eq' multiset-nonemptyE*
nat-1-add-1 remove1-mset-eqE size-single zero-neq-one)
then have $const: bl \cap bl' = m$ **using** *const-intersect assms* **by** *auto*
thus *False* **using** *inter-num-max-bound(1) finite-blocks*
by (*metis a blin assms(1) finite-blocks in-diffD*)
qed

lemma *const-inter-multiplicity-one*:

assumes $bl \in \# \mathcal{B}$
assumes $m < \text{card } bl$
shows $\text{multiplicity } bl = 1$
proof (*rule ccontr*)
assume $\text{multiplicity } bl \neq 1$
then have $\text{multiplicity } bl > 1$ **using** *assms*
by (*simp add: le-neq-implies-less*)
then obtain $bl2$ **where** $bl = bl2$ **and** $bl2 \in \# \mathcal{B} - \{\#bl\# \}$
by (*metis count-single in-diff-count*)
then have $bl \cap bl2 = \text{card } bl$
using *inter-num-of-eq-blocks* **by** *blast*
thus *False* **using** *assms const-intersect*
by (*simp add: ⟨bl2 ∈ # remove1-mset bl B⟩*)
qed

lemma *mult-blocks-const-inter*:

assumes $bl \in \# \mathcal{B}$
assumes $\text{multiplicity } bl > 1$
assumes $b \geq 2$
shows $m = \text{card } bl$
proof (*rule ccontr*)
assume $m \neq \text{card } bl$
then have $m < \text{card } bl$ **using** *inter-num-le-block-size assms*
using *nat-less-le* **by** *blast*
then have $\text{multiplicity } bl = 1$ **using** *const-inter-multiplicity-one assms* **by** *simp*
thus *False* **using** *assms(2)* **by** *simp*

qed

lemma *simple-const-inter-block-size*: $(\bigwedge bl. bl \in \# \mathcal{B} \implies m < \text{card } bl) \implies \text{simple-design } \mathcal{V} \mathcal{B}$
using *const-inter-multiplicity-one* by (*unfold-locales*) (*simp*)

lemma *simple-const-inter-iff*:

assumes $b \geq 2$

shows $\text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \} \leq 1 \iff \text{simple-design } \mathcal{V} \mathcal{B}$

proof (*intro iffI*)

assume a : $\text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \} \leq 1$

show *simple-design* $\mathcal{V} \mathcal{B}$

proof (*unfold-locales*)

fix bl assume $blin$: $bl \in \# \mathcal{B}$

show *multiplicity* $bl = 1$

proof (*cases card bl = m*)

case *True*

then have m : *multiplicity* $bl = \text{size } \{\#b \in \# \mathcal{B} . b = bl \# \}$

by (*simp add: count-size-set-repr*)

then have $\{\#b \in \# \mathcal{B} . b = bl \# \} \subseteq \# \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \}$ using

True

by (*simp add: mset-subset-eqI*)

then have $\text{size } \{\#b \in \# \mathcal{B} . b = bl \# \} \leq \text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \}$

by (*simp add: size-mset-mono*)

then show *?thesis* using a $blin$

by (*metis count-eq-zero-iff le-neq-implies-less le-trans less-one m*)

next

case *False*

then have $m < \text{card } bl$ using *assms*

by (*simp add: blin inter-num-le-block-size le-neq-implies-less*)

then show *?thesis* using *const-inter-multiplicity-one*

by (*simp add: blin*)

qed

qed

next

assume *simp*: *simple-design* $\mathcal{V} \mathcal{B}$

then have *mult*: $\bigwedge bl. bl \in \# \mathcal{B} \implies \text{multiplicity } bl = 1$

using *simple-design.axioms(2)* *simple-incidence-system.simple-alt-def-all* by

blast

show $\text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \} \leq 1$

proof (*rule ccontr*)

assume $\neg \text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \} \leq 1$

then have $\text{size } \{\#bl \in \# \mathcal{B} . \text{card } bl = m \# \} > 1$ by *simp*

then obtain $bl1$ $bl2$ where $blin$: $bl1 \in \# \mathcal{B}$ and $bl2in$: $bl2 \in \# \mathcal{B} - \{\#bl1 \# \}$

and

$\text{card}1$: $\text{card } bl1 = m$ and $\text{card}2$: $\text{card } bl2 = m$

using *obtain-two-items-mset-filter* by *blast*

then have $bl1 \cap bl2 = m$ using *const-intersect* by *simp*

then have $bl1 = bl2$

by (metis blin bl2in card1 card2 finite-blocks in-diffD inter-eq-blocks-eq-card)
 then have multiplicity bl1 > 1
 using ⟨bl2 ∈# remove1-mset bl1 B⟩ count-eq-zero-iff by force
 thus False using mult blin by simp
 qed
 qed

lemma empty-inter-implies-rep-one:

assumes m = 0
 assumes x ∈ V
 shows B rep x ≤ 1
 proof (rule ccontr)
 assume a: ¬ B rep x ≤ 1
 then have gt1: B rep x > 1 by simp
 then obtain bl1 where blin1: bl1 ∈# B and xin1: x ∈ bl1
 by (metis gr-implies-not0 linorder-neqE-nat rep-number-g0-exists)
 then have (B - {#bl1#}) rep x > 0 using gt1 point-rep-number-split point-rep-singleton-val
 by (metis a add-0 eq-imp-le neq0-conv remove1-mset-eqE)
 then obtain bl2 where blin2: bl2 ∈# (B - {#bl1#}) and xin2: x ∈ bl2
 by (metis rep-number-g0-exists)
 then have x ∈ (bl1 ∩ bl2) using xin1 by simp
 then have bl1 |∩| bl2 ≠ 0
 by (metis blin1 empty-iff finite-blocks intersection-number-empty-iff)
 thus False using const-intersect assms blin1 blin2 by simp
 qed

lemma empty-inter-implies-b-lt-v:

assumes m = 0
 shows b ≤ v
 proof -
 have le1: $\bigwedge x. x \in V \implies B \text{ rep } x \leq 1$ using empty-inter-implies-rep-one assms
 by simp
 have disj: $\{v \in V . B \text{ rep } v = 0\} \cap \{v \in V . \neg (B \text{ rep } v = 0)\} = \{\}$ by auto
 have eqv: $V = (\{v \in V . B \text{ rep } v = 0\} \cup \{v \in V . \neg (B \text{ rep } v = 0)\})$ by auto
 have b ≤ (∑ x ∈ V . B rep x) using block-num-rep-bound by simp
 also have 1: ... ≤ (∑ x ∈ ({v ∈ V . B rep v = 0} ∪ {v ∈ V . ¬ (B rep v = 0)})
 . B rep x)
 using eqv by simp
 also have ... ≤ (∑ x ∈ ({v ∈ V . B rep v = 0}) . B rep x) + (∑ x ∈ ({v ∈ V
 . ¬ (B rep v = 0)}) . B rep x)
 using sum.union-disjoint finite-sets eqv disj
 by (metis (no-types, lifting) 1 finite-Un)
 also have ... ≤ (∑ x ∈ ({v ∈ V . ¬ (B rep v = 0)}) . B rep x) by simp
 also have ... ≤ (∑ x ∈ ({v ∈ V . ¬ (B rep v = 0)}) . 1) using le1
 by (metis (mono-tags, lifting) mem-Collect-eq sum-mono)
 also have ... ≤ card {v ∈ V . ¬ (B rep v = 0)} by simp
 also have ... ≤ card V using finite-sets
 using card-mono eqv by blast
 finally show ?thesis by simp

qed

end

locale *simple-const-intersect-design* = *const-intersect-design* + *simple-design*

end

4 Incidence Vectors and Matrices

Incidence Matrices are an important representation for any incidence set system. The majority of basic definitions and properties proved in this theory are based on Stinson [8] and Colbourn [3].

theory *Incidence-Matrices* **imports** *Design-Extras Matrix-Vector-Extras List-Index.List-Index Design-Theory.Design-Isomorphisms*
begin

4.1 Incidence Vectors

A function which takes an ordered list of points, and a block, returning a 0-1 vector v where there is a 1 in the i th position if point i is in that block

definition *inc-vec-of* :: 'a list \Rightarrow 'a set \Rightarrow ('b :: {ring-1}) vec **where**
inc-vec-of Vs bl \equiv vec (length Vs) (λ i . if (Vs ! i) \in bl then 1 else 0)

lemma *inc-vec-one-zero-elems*: set_v (*inc-vec-of* Vs bl) \subseteq {0, 1}
by (auto simp add: vec-set-def *inc-vec-of-def*)

lemma *finite-inc-vec-elems*: finite (set_v (*inc-vec-of* Vs bl))
using *finite-subset inc-vec-one-zero-elems* **by** blast

lemma *inc-vec-elems-max-two*: $card$ (set_v (*inc-vec-of* Vs bl)) \leq 2
using *card-mono inc-vec-one-zero-elems finite.insertI card-0-eq card-2-iff*
by (smt (verit) *insert-absorb2 linorder-le-cases linordered-nonzero-semiring-class.zero-le-one*

obtain-subset-with-card-n one-add-one subset-singletonD trans-le-add1)

lemma *inc-vec-dim*: dim_vec (*inc-vec-of* Vs bl) = length Vs
by (simp add: *inc-vec-of-def*)

lemma *inc-vec-index*: $i < length$ Vs \implies *inc-vec-of* Vs bl \$ i = (if (Vs ! i) \in bl then 1 else 0)
by (simp add: *inc-vec-of-def*)

lemma *inc-vec-index-one-iff*: $i < length$ Vs \implies *inc-vec-of* Vs bl \$ i = 1 \iff Vs ! i \in bl
by (auto simp add: *inc-vec-of-def*)

lemma *inc-vec-index-zero-iff*: $i < \text{length } Vs \implies \text{inc-vec-of } Vs \text{ bl } \$ i = 0 \iff Vs ! i \notin \text{bl}$

by (*auto simp add: inc-vec-of-def*)

lemma *inc-vec-of-bij-betw*:

assumes *inj-on* f (*set* Vs)

assumes $\text{bl} \subseteq (\text{set } Vs)$

shows $\text{inc-vec-of } Vs \text{ bl} = \text{inc-vec-of } (\text{map } f \text{ } Vs) (f \text{ ' } \text{bl})$

proof (*intro eq-vecI, simp-all add: inc-vec-dim*)

fix i **assume** $i < \text{length } Vs$

then have $Vs ! i \in \text{bl} \iff (\text{map } f \text{ } Vs) ! i \in (f \text{ ' } \text{bl})$

by (*metis assms(1) assms(2) inj-on-image-mem-iff nth-map nth-mem*)

then show $\text{inc-vec-of } Vs \text{ bl } \$ i = \text{inc-vec-of } (\text{map } f \text{ } Vs) (f \text{ ' } \text{bl}) \$ i$

using *inc-vec-index* **by** (*metis <i < length Vs> length-map*)

qed

4.2 Incidence Matrices

A function which takes a list of points, and list of sets of points, and returns a $v \times b$ 0-1 matrix M , where v is the number of points, and b the number of sets, such that there is a 1 in the i, j position if and only if point i is in block j . The matrix has type *'b mat* to allow for operations commonly used on matrices [8]

definition *inc-mat-of* :: *'a list* \Rightarrow *'a set list* \Rightarrow (*'b* :: {*ring-1*}) *mat* **where**
inc-mat-of $Vs \ Bs \equiv \text{mat } (\text{length } Vs) (\text{length } Bs) (\lambda (i,j) . \text{if } (Vs ! i) \in (Bs ! j) \text{ then } 1 \text{ else } 0)$

Basic lemmas on the *inc-mat-of* matrix result (elements/dimensions/indexing)

lemma *inc-mat-one-zero-elems*: $\text{elements-mat } (\text{inc-mat-of } Vs \ Bs) \subseteq \{0, 1\}$

by (*auto simp add: inc-mat-of-def elements-mat-def*)

lemma *fin-incidence-mat-elems*: *finite* ($\text{elements-mat } (\text{inc-mat-of } Vs \ Bs)$)

using *finite-subset inc-mat-one-zero-elems* **by** *auto*

lemma *inc-matrix-elems-max-two*: $\text{card } (\text{elements-mat } (\text{inc-mat-of } Vs \ Bs)) \leq 2$

using *inc-mat-one-zero-elems order-trans card-2-iff*

by (*smt (verit, del-insts) antisym bot.extremum card.empty insert-commute insert-subsetI*)

is-singletonI is-singleton-altdef linorder-le-cases not-one-le-zero one-le-numeral subset-insert)

lemma *inc-mat-of-index* [*simp*]: $i < \text{dim-row } (\text{inc-mat-of } Vs \ Bs) \implies j < \text{dim-col } (\text{inc-mat-of } Vs \ Bs) \implies$

$\text{inc-mat-of } Vs \ Bs \ \$\$ (i, j) = (\text{if } (Vs ! i) \in (Bs ! j) \text{ then } 1 \text{ else } 0)$

by (*simp add: inc-mat-of-def*)

lemma *inc-mat-dim-row*: $\text{dim-row } (\text{inc-mat-of } Vs \ Bs) = \text{length } Vs$

by (simp add: inc-mat-of-def)

lemma *inc-mat-dim-vec-row*: $\dim\text{-vec} (\text{row} (\text{inc-mat-of } Vs \ Bs) \ i) = \text{length } Bs$
by (simp add: inc-mat-of-def)

lemma *inc-mat-dim-col*: $\dim\text{-col} (\text{inc-mat-of } Vs \ Bs) = \text{length } Bs$
by (simp add: inc-mat-of-def)

lemma *inc-mat-dim-vec-col*: $\dim\text{-vec} (\text{col} (\text{inc-mat-of } Vs \ Bs) \ i) = \text{length } Vs$
by (simp add: inc-mat-of-def)

lemma *inc-matrix-point-in-block-one*: $i < \text{length } Vs \implies j < \text{length } Bs \implies Vs ! i \in Bs ! j$
 $\implies (\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 1$
by (simp add: inc-mat-of-def)

lemma *inc-matrix-point-not-in-block-zero*: $i < \text{length } Vs \implies j < \text{length } Bs \implies Vs ! i \notin Bs ! j \implies$
 $(\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 0$
by (simp add: inc-mat-of-def)

lemma *inc-matrix-point-in-block*: $i < \text{length } Vs \implies j < \text{length } Bs \implies (\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 1$
 $\implies Vs ! i \in Bs ! j$
using *inc-matrix-point-not-in-block-zero* by (metis zero-neq-one)

lemma *inc-matrix-point-not-in-block*: $i < \text{length } Vs \implies j < \text{length } Bs \implies$
 $(\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 0 \implies Vs ! i \notin Bs ! j$
using *inc-matrix-point-in-block-one* by (metis zero-neq-one)

lemma *inc-matrix-point-not-in-block-iff*: $i < \text{length } Vs \implies j < \text{length } Bs \implies$
 $(\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 0 \iff Vs ! i \notin Bs ! j$
using *inc-matrix-point-not-in-block* *inc-matrix-point-not-in-block-zero* by blast

lemma *inc-matrix-point-in-block-iff*: $i < \text{length } Vs \implies j < \text{length } Bs \implies$
 $(\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 1 \iff Vs ! i \in Bs ! j$
using *inc-matrix-point-in-block* *inc-matrix-point-in-block-one* by blast

lemma *inc-matrix-subset-implies-one*:
assumes $I \subseteq \{.. < \text{length } Vs\}$
assumes $j < \text{length } Bs$
assumes (!) $Vs \ ' I \subseteq Bs ! j$
assumes $i \in I$
shows $(\text{inc-mat-of } Vs \ Bs) \ \$(i, j) = 1$
proof –
have *iin*: $Vs ! i \in Bs ! j$ **using** *assms(3)* *assms(4)* **by** *auto*
have $i < \text{length } Vs$ **using** *assms(1)* *assms(4)* **by** *auto*
thus *thesis* **using** *iin* *inc-matrix-point-in-block-iff* *assms(2)* **by** *blast*
qed

lemma *inc-matrix-one-implies-membership*: $I \subseteq \{.. < \text{length } Vs\} \implies j < \text{length } Bs \implies$

$(\bigwedge i. i \in I \implies (\text{inc-mat-of } Vs \ Bs) \ \S\ \$ (i, j) = 1) \implies i \in I \implies Vs ! i \in Bs ! j$
using *inc-matrix-point-in-block subset-iff* **by** *blast*

lemma *inc-matrix-elems-one-zero*: $i < \text{length } Vs \implies j < \text{length } Bs \implies$

$(\text{inc-mat-of } Vs \ Bs) \ \S\ \$ (i, j) = 0 \vee (\text{inc-mat-of } Vs \ Bs) \ \S\ \$ (i, j) = 1$
using *inc-matrix-point-in-block-one inc-matrix-point-not-in-block-zero* **by** *blast*

Reasoning on Rows/Columns of the incidence matrix

lemma *inc-mat-col-def*: $j < \text{length } Bs \implies i < \text{length } Vs \implies$

$(\text{col } (\text{inc-mat-of } Vs \ Bs) \ j) \ \$ i = (\text{if } (Vs ! i \in Bs ! j) \ \text{then } 1 \ \text{else } 0)$
by (*simp add: inc-mat-of-def*)

lemma *inc-mat-col-list-map-elem*: $j < \text{length } Bs \implies i < \text{length } Vs \implies$

$\text{col } (\text{inc-mat-of } Vs \ Bs) \ j \ \$ i = \text{map-vec } (\lambda x. \text{if } (x \in (Bs ! j)) \ \text{then } 1 \ \text{else } 0)$
 $(\text{vec-of-list } Vs) \ \$ i$
by (*simp add: inc-mat-of-def index-vec-of-list*)

lemma *inc-mat-col-list-map*: $j < \text{length } Bs \implies$

$\text{col } (\text{inc-mat-of } Vs \ Bs) \ j = \text{map-vec } (\lambda x. \text{if } (x \in (Bs ! j)) \ \text{then } 1 \ \text{else } 0)$
 $(\text{vec-of-list } Vs)$
by (*intro eq-vecI*)
(simp-all add: inc-mat-dim-row inc-mat-dim-col inc-mat-col-list-map-elem index-vec-of-list)

lemma *inc-mat-row-def*: $j < \text{length } Bs \implies i < \text{length } Vs \implies$

$(\text{row } (\text{inc-mat-of } Vs \ Bs) \ i) \ \$ j = (\text{if } (Vs ! i \in Bs ! j) \ \text{then } 1 \ \text{else } 0)$
by (*simp add: inc-mat-of-def*)

lemma *inc-mat-row-list-map-elem*: $j < \text{length } Bs \implies i < \text{length } Vs \implies$

$\text{row } (\text{inc-mat-of } Vs \ Bs) \ i \ \$ j = \text{map-vec } (\lambda bl. \text{if } ((Vs ! i) \in bl) \ \text{then } 1 \ \text{else } 0)$
 $(\text{vec-of-list } Bs) \ \$ j$
by (*simp add: inc-mat-of-def vec-of-list-index*)

lemma *inc-mat-row-list-map*: $i < \text{length } Vs \implies$

$\text{row } (\text{inc-mat-of } Vs \ Bs) \ i = \text{map-vec } (\lambda bl. \text{if } ((Vs ! i) \in bl) \ \text{then } 1 \ \text{else } 0)$
 $(\text{vec-of-list } Bs)$
by (*intro eq-vecI*)
(simp-all add: inc-mat-dim-row inc-mat-dim-col inc-mat-row-list-map-elem index-vec-of-list)

Connecting *inc-vec-of* and *inc-mat-of*

lemma *inc-mat-col-inc-vec*: $j < \text{length } Bs \implies \text{col } (\text{inc-mat-of } Vs \ Bs) \ j = \text{inc-vec-of } Vs \ (Bs ! j)$

by (*auto simp add: inc-mat-of-def inc-vec-of-def*)

lemma *inc-mat-of-cols-inc-vecs*: $\text{cols } (\text{inc-mat-of } Vs \ Bs) = \text{map } (\lambda j. \text{inc-vec-of } Vs \ j) \ Bs$

proof (*intro nth-equalityI*)
have $l1$: $\text{length} (\text{cols} (\text{inc-mat-of } Vs \ Bs)) = \text{length } Bs$
using *inc-mat-dim-col by simp*
have $l2$: $\text{length} (\text{map} (\lambda j . \text{inc-vec-of } Vs \ j) \ Bs) = \text{length } Bs$
using *length-map by simp*
then show $\text{length} (\text{cols} (\text{inc-mat-of } Vs \ Bs)) = \text{length} (\text{map} (\text{inc-vec-of } Vs) \ Bs)$
using $l1 \ l2$ **by** *simp*
show $\bigwedge i . i < \text{length} (\text{cols} (\text{inc-mat-of } Vs \ Bs)) \implies$
 $(\text{cols} (\text{inc-mat-of } Vs \ Bs) ! i) = (\text{map} (\lambda j . \text{inc-vec-of } Vs \ j) \ Bs) ! i$
using *inc-mat-col-inc-vec l1 by (metis cols-nth inc-mat-dim-col nth-map)*
qed

lemma *inc-mat-of-bij-betw*:
assumes *inj-on f (set Vs)*
assumes $\bigwedge bl . bl \in (\text{set } Bs) \implies bl \subseteq (\text{set } Vs)$
shows $\text{inc-mat-of } Vs \ Bs = \text{inc-mat-of} (\text{map } f \ Vs) (\text{map} ((\cdot) f) \ Bs)$
proof (*intro eq-matI, simp-all add: inc-mat-dim-row inc-mat-dim-col, intro impI*)
fix $i \ j$ **assume** *ilt: $i < \text{length } Vs$ and jlt: $j < \text{length } Bs$ and $Vs ! i \notin Bs ! j$*
then show $f (Vs ! i) \notin f ' Bs ! j$
by (*meson assms(1) assms(2) ilt inj-on-image-mem-iff jlt nth-mem*)
qed

Definitions for the incidence matrix representation of common incidence system properties

definition *non-empty-col* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
non-empty-col $M \ j \equiv \exists k . k \neq 0 \wedge k \in \$ \text{col } M \ j$

definition *proper-inc-mat* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{bool}$ **where**
proper-inc-mat $M \equiv (\text{dim-row } M > 0 \wedge \text{dim-col } M > 0)$

Matrix version of the representation number property (*rep*)

definition *mat-rep-num* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
mat-rep-num $M \ i \equiv \text{count-vec} (\text{row } M \ i) \ 1$

Matrix version of the points index property (*index*)

definition *mat-point-index* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{nat} \ \text{set} \Rightarrow \text{nat}$ **where**
mat-point-index $M \ I \equiv \text{card} \{j . j < \text{dim-col } M \wedge (\forall i \in I . M \ \$\$ (i, j) = 1)\}$

definition *mat-inter-num* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
mat-inter-num $M \ j1 \ j2 \equiv \text{card} \{i . i < \text{dim-row } M \wedge M \ \$\$ (i, j1) = 1 \wedge M \ \$\$ (i, j2) = 1\}$

Matrix version of the block size property

definition *mat-block-size* :: $('a :: \{\text{zero-neq-one}\}) \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
mat-block-size $M \ j \equiv \text{count-vec} (\text{col } M \ j) \ 1$

lemma *non-empty-col-obtains*:
assumes *non-empty-col* $M \ j$
obtains i **where** $i < \text{dim-row } M$ **and** $(\text{col } M \ j) \ \$ \ i \neq 0$

proof –

have $d: \text{dim-vec } (\text{col } M j) = \text{dim-row } M$ **by** *simp*
from *assms* **obtain** k **where** $k \neq 0$ **and** $k \in \text{col } M j$
by (*auto simp add: non-empty-col-def*)
thus *?thesis* **using** *vec-contains-obtains-index d*
by (*metis that*)

qed

lemma *non-empty-col-alt-def*:

assumes $j < \text{dim-col } M$
shows $\text{non-empty-col } M j \iff (\exists i. i < \text{dim-row } M \wedge M \$\$ (i, j) \neq 0)$

proof (*intro iffI*)

show $\text{non-empty-col } M j \implies \exists i < \text{dim-row } M. M \$\$ (i, j) \neq 0$
by (*metis assms index-col non-empty-col-obtains*)

next

assume $\exists i < \text{dim-row } M. M \$\$ (i, j) \neq 0$
then obtain i **where** *ilt*: $i < \text{dim-row } M$ **and** *ne*: $M \$\$ (i, j) \neq 0$ **by** *blast*
then have *ilt2*: $i < \text{dim-vec } (\text{col } M j)$ **by** *simp*
then have $(\text{col } M j) \$ i \neq 0$ **using** *ne* **by** (*simp add: assms*)
then obtain k **where** $(\text{col } M j) \$ i = k$ **and** $k \neq 0$
by *simp*
then show $\text{non-empty-col } M j$ **using** *non-empty-col-def*
by (*metis ilt2 vec-setI*)

qed

lemma *proper-inc-mat-map*: $\text{proper-inc-mat } M \implies \text{proper-inc-mat } (\text{map-mat } f M)$
by (*simp add: proper-inc-mat-def*)

lemma *mat-point-index-alt*: $\text{mat-point-index } M I = \text{card } \{j \in \{0..<\text{dim-col } M\} . (\forall i \in I. M \$\$ (i, j) = 1)\}$
by (*simp add: mat-point-index-def*)

lemma *mat-block-size-sum-alt*:

fixes $M :: 'a :: \{\text{ring-1}\} \text{ mat}$
shows $\text{elements-mat } M \subseteq \{0, 1\} \implies j < \text{dim-col } M \implies \text{of-nat } (\text{mat-block-size } M j) = \text{sum-vec } (\text{col } M j)$
unfolding *mat-block-size-def* **using** *count-vec-sum-ones-alt col-elems-subset-mat subset-trans*
by *metis*

lemma *mat-rep-num-sum-alt*:

fixes $M :: 'a :: \{\text{ring-1}\} \text{ mat}$
shows $\text{elements-mat } M \subseteq \{0, 1\} \implies i < \text{dim-row } M \implies \text{of-nat } (\text{mat-rep-num } M i) = \text{sum-vec } (\text{row } M i)$
using *count-vec-sum-ones-alt*
by (*metis mat-rep-num-def row-elems-subset-mat subset-trans*)

lemma *mat-point-index-two-alt*:

assumes $i1 < \text{dim-row } M$

```

assumes  $i2 < \dim\text{-row } M$ 
shows  $\text{mat-point-index } M \{i1, i2\} = \text{card } \{j . j < \dim\text{-col } M \wedge M \text{ \$(}i1, j) = 1 \wedge M \text{ \$(}i2, j) = 1\}$ 
proof –
  let  $?I = \{i1, i2\}$ 
  have  $ss: \{i1, i2\} \subseteq \{.. < \dim\text{-row } M\}$  using assms by blast
  have  $\text{filter}: \bigwedge j . j < \dim\text{-col } M \implies (\forall i \in ?I . M \text{ \$(}i, j) = 1) \longleftrightarrow M \text{ \$(}i1, j) = 1 \wedge M \text{ \$(}i2, j) = 1$ 
  by auto
  have  $?I \subseteq \{.. < \dim\text{-row } M\}$  using assms(1) assms(2) by fastforce
  thus ?thesis using filter ss unfolding mat-point-index-def
  by meson
qed

```

Transpose symmetries

```

lemma trans-mat-rep-block-size-sym:  $j < \dim\text{-col } M \implies \text{mat-block-size } M j = \text{mat-rep-num } M^T j$ 
 $i < \dim\text{-row } M \implies \text{mat-rep-num } M i = \text{mat-block-size } M^T i$ 
unfolding mat-block-size-def mat-rep-num-def by simp-all

```

```

lemma trans-mat-point-index-inter-sym:
 $i1 < \dim\text{-row } M \implies i2 < \dim\text{-row } M \implies \text{mat-point-index } M \{i1, i2\} = \text{mat-inter-num } M^T i1 i2$ 
 $j1 < \dim\text{-col } M \implies j2 < \dim\text{-col } M \implies \text{mat-inter-num } M j1 j2 = \text{mat-point-index } M^T \{j1, j2\}$ 
apply (simp-all add: mat-inter-num-def mat-point-index-two-alt)
apply (metis (no-types, lifting) index-transpose-mat(1))
by (metis (no-types, lifting) index-transpose-mat(1))

```

4.3 0-1 Matrices

Incidence matrices contain only two elements: 0 and 1. We define a locale which provides a context to work in for matrices satisfying this condition for any 'b type.

```

locale zero-one-matrix =
  fixes matrix :: 'b :: {zero-neq-one} mat (M)
  assumes elems01:  $\text{elements-mat } M \subseteq \{0, 1\}$ 
begin

```

Row and Column Properties of the Matrix

```

lemma row-elems-ss01:  $i < \dim\text{-row } M \implies \text{vec-set } (\text{row } M i) \subseteq \{0, 1\}$ 
using row-elems-subset-mat elems01 by blast

```

```

lemma col-elems-ss01:
  assumes  $j < \dim\text{-col } M$ 
  shows  $\text{vec-set } (\text{col } M j) \subseteq \{0, 1\}$ 
proof –
  have  $\text{vec-set } (\text{col } M j) \subseteq \text{elements-mat } M$  using assms
  by (simp add: col-elems-subset-mat assms)

```

thus *?thesis* using *elems01* by *blast*
qed

lemma *col-nth-0-or-1-iff*:
assumes $j < \dim\text{-col } M$
assumes $i < \dim\text{-row } M$
shows $\text{col } M j \$ i = 0 \longleftrightarrow \text{col } M j \$ i \neq 1$
proof (*intro iffI*)
have $dv: i < \dim\text{-vec } (\text{col } M j)$ using *assms* by *simp*
have $sv: \text{set}_v (\text{col } M j) \subseteq \{0, 1\}$ using *col-elems-ss01* *assms* by *simp*
then show $\text{col } M j \$ i = 0 \implies \text{col } M j \$ i \neq 1$ using *dv* by *simp*
show $\text{col } M j \$ i \neq 1 \implies \text{col } M j \$ i = 0$ using *dv sv*
by (*meson insertE singletonD subset-eq vec-setI*)
qed

lemma *row-nth-0-or-1-iff*:
assumes $j < \dim\text{-col } M$
assumes $i < \dim\text{-row } M$
shows $\text{row } M i \$ j = 0 \longleftrightarrow \text{row } M i \$ j \neq 1$
proof (*intro iffI*)
have $dv: j < \dim\text{-vec } (\text{row } M i)$ using *assms* by *simp*
have $sv: \text{set}_v (\text{row } M i) \subseteq \{0, 1\}$ using *row-elems-ss01* *assms* by *simp*
then show $\text{row } M i \$ j = 0 \implies \text{row } M i \$ j \neq 1$ by *simp*
show $\text{row } M i \$ j \neq 1 \implies \text{row } M i \$ j = 0$ using *dv sv*
by (*meson insertE singletonD subset-eq vec-setI*)
qed

lemma *transpose-entries*: $\text{elements-mat } (M^T) \subseteq \{0, 1\}$
using *elems01* *transpose-mat-elems* by *metis*

lemma *M-not-zero-simp*: $j < \dim\text{-col } M \implies i < \dim\text{-row } M \implies M \$\$ (i, j) \neq 0$
 $\implies M \$\$ (i, j) = 1$
using *elems01* by *auto*

lemma *M-not-one-simp*: $j < \dim\text{-col } M \implies i < \dim\text{-row } M \implies M \$\$ (i, j) \neq 1$
 $\implies M \$\$ (i, j) = 0$
using *elems01* by *auto*

Definition for mapping a column to a block

definition *map-col-to-block* :: 'a :: {zero-neq-one} vec \Rightarrow nat set **where**
map-col-to-block $c \equiv \{ i \in \{..<\dim\text{-vec } c\} . c \$ i = 1 \}$

lemma *map-col-to-block-alt*: $\text{map-col-to-block } c = \{i . i < \dim\text{-vec } c \wedge c \$ i = 1\}$
by (*simp add: map-col-to-block-def*)

lemma *map-col-to-block-elem*: $i < \dim\text{-vec } c \implies i \in \text{map-col-to-block } c \longleftrightarrow c \$ i = 1$
by (*simp add: map-col-to-block-alt*)

lemma *in-map-col-valid-index*: $i \in \text{map-col-to-block } c \implies i < \text{dim-vec } c$
by (*simp add: map-col-to-block-alt*)

lemma *map-col-to-block-size*: $j < \text{dim-col } M \implies \text{card } (\text{map-col-to-block } (\text{col } M j))$
 $= \text{mat-block-size } M j$
unfolding *mat-block-size-def map-col-to-block-alt* **using** *count-vec-alt*[*of col M j*
1] *Collect-cong*
by (*metis (no-types, lifting)*)

lemma *in-map-col-valid-index-M*: $j < \text{dim-col } M \implies i \in \text{map-col-to-block } (\text{col } M j) \implies i < \text{dim-row } M$
using *in-map-col-valid-index* **by** (*metis dim-col*)

lemma *map-col-to-block-elem-not*: $c \in \text{set } (\text{cols } M) \implies i < \text{dim-vec } c \implies i \notin \text{map-col-to-block } c \longleftrightarrow c \$ i = 0$
apply (*auto simp add: map-col-to-block-alt*)
using *elems01* **by** (*metis col-nth-0-or-1-iff dim-col obtain-col-index*)

lemma *obtain-block-index-map-block-set*:
assumes $bl \in \# \{ \# \text{map-col-to-block } c . c \in \# \text{mset } (\text{cols } M) \# \}$
obtains j **where** $j < \text{dim-col } M$ **and** $bl = \text{map-col-to-block } (\text{col } M j)$
proof –
obtain c **where** *bleq*: $bl = \text{map-col-to-block } c$ **and** $c \in \# \text{mset } (\text{cols } M)$
using *assms* **by** *blast*
then have $c \in \text{set } (\text{cols } M)$ **by** *simp*
thus *?thesis* **using** *bleq obtain-col-index*
by (*metis that*)
qed

lemma *mat-ord-inc-sys-point*[*simp*]: $x < \text{dim-row } M \implies [0..<(\text{dim-row } M)] ! x = x$
by *simp*

lemma *mat-ord-inc-sys-block*[*simp*]: $j < \text{dim-col } M \implies$
 $(\text{map } (\text{map-col-to-block}) (\text{cols } M)) ! j = \text{map-col-to-block } (\text{col } M j)$
by *auto*

lemma *ordered-to-mset-col-blocks*:
 $\{ \# \text{map-col-to-block } c . c \in \# \text{mset } (\text{cols } M) \# \} = \text{mset } (\text{map } (\text{map-col-to-block}) (\text{cols } M))$
by *simp*

Lemmas on incidence matrix properties

lemma *non-empty-col-01*:
assumes $j < \text{dim-col } M$
shows *non-empty-col* $M j \longleftrightarrow 1 \in \$ \text{col } M j$
proof (*intro iffI*)
assume *non-empty-col* $M j$
then obtain k **where** *kn0*: $k \neq 0$ **and** *kin*: $k \in \$ \text{col } M j$ **using** *non-empty-col-def*

by *blast*
then have $k \in \text{elements-mat } M$ **using** *vec-contains-col-elements-mat assms*
by *metis*
then have $k = 1$ **using** *kn0*
using *elems01* **by** *blast*
thus $1 \in \$ \text{col } M j$ **using** *kin* **by** *simp*
next
assume $1 \in \$ \text{col } M j$
then show *non-empty-col M j* **using** *non-empty-col-def*
by (*metis zero-neq-one*)
qed

lemma *mat-rep-num-alt:*
assumes $i < \text{dim-row } M$
shows $\text{mat-rep-num } M i = \text{card } \{j . j < \text{dim-col } M \wedge M \$\$ (i, j) = 1\}$
proof (*simp add: mat-rep-num-def*)
have $\text{eq: } \bigwedge j. (j < \text{dim-col } M \wedge M \$\$ (i, j) = 1) = (\text{row } M i \$ j = 1 \wedge j < \text{dim-vec } (\text{row } M i))$
using *assms* **by** *auto*
have $\text{count-vec } (\text{row } M i) 1 = \text{card } \{j. (\text{row } M i) \$ j = 1 \wedge j < \text{dim-vec } (\text{row } M i)\}$
using *count-vec-alt[of row M i 1]* **by** *simp*
thus $\text{count-vec } (\text{row } M i) 1 = \text{card } \{j. j < \text{dim-col } M \wedge M \$\$ (i, j) = 1\}$
using *eq Collect-cong* **by** *simp*
qed

lemma *mat-rep-num-alt-col:* $i < \text{dim-row } M \implies \text{mat-rep-num } M i = \text{size } \{\#c \in \# (\text{mset } (\text{cols } M)) . c \$ i = 1\#\}$
using *mat-rep-num-alt index-to-col-card-size-prop[of i M]* **by** *auto*

A zero one matrix is an incidence system

lemma *map-col-to-block-wf:* $\bigwedge c. c \in \text{set } (\text{cols } M) \implies \text{map-col-to-block } c \subseteq \{0..<\text{dim-row } M\}$
by (*auto simp add: map-col-to-block-def*)(*metis dim-col obtain-col-index*)

lemma *one-implies-block-nempty:* $j < \text{dim-col } M \implies 1 \in \$ (\text{col } M j) \implies \text{map-col-to-block } (\text{col } M j) \neq \{\}$
unfolding *map-col-to-block-def* **using** *vec-setE* **by** *force*

interpretation *incidence-sys:* *incidence-system* $\{0..<\text{dim-row } M\}$
 $\{\# \text{map-col-to-block } c . c \in \# \text{mset } (\text{cols } M)\#\}$
using *map-col-to-block-wf* **by** (*unfold-locales*) *auto*

interpretation *fin-incidence-sys:* *finite-incidence-system* $\{0..<\text{dim-row } M\}$
 $\{\# \text{map-col-to-block } c . c \in \# \text{mset } (\text{cols } M)\#\}$
by (*unfold-locales*) (*simp*)

lemma *block-nempty-implies-all-zeros:* $j < \text{dim-col } M \implies \text{map-col-to-block } (\text{col } M j) = \{\} \implies$

$i < \dim\text{-row } M \implies \text{col } M j \ \$ \ i = 0$
by (*metis col-nth-0-or-1-iff dim-col one-implies-block-empty vec-setI*)

lemma *block-empty-implies-no-one*: $j < \dim\text{-col } M \implies \text{map-col-to-block } (\text{col } M j) = \{\} \implies \neg (1 \in \$ (\text{col } M j))$
using *one-implies-block-empty* **by** *blast*

lemma *mat-is-design*:

assumes $\bigwedge j. j < \dim\text{-col } M \implies 1 \in \$ (\text{col } M j)$
shows *design* $\{0..<\dim\text{-row } M\} \{\#\ \text{map-col-to-block } c . c \in \#\ \text{mset } (\text{cols } M)\#\}$
proof (*unfold-locales*)
fix *bl*
assume $bl \in \#\ \{\#\ \text{map-col-to-block } c . c \in \#\ \text{mset } (\text{cols } M)\#\}$
then obtain *j* **where** $j < \dim\text{-col } M$ **and** *map*: $bl = \text{map-col-to-block } (\text{col } M j)$
using *obtain-block-index-map-block-set* **by** *auto*
thus $bl \neq \{\}$ **using** *assms one-implies-block-empty*
by *simp*
qed

lemma *mat-is-proper-design*:

assumes $\bigwedge j. j < \dim\text{-col } M \implies 1 \in \$ (\text{col } M j)$
assumes $\dim\text{-col } M > 0$
shows *proper-design* $\{0..<\dim\text{-row } M\} \{\#\ \text{map-col-to-block } c . c \in \#\ \text{mset } (\text{cols } M)\#\}$
proof –
interpret *des*: *design* $\{0..<\dim\text{-row } M\} \{\#\ \text{map-col-to-block } c . c \in \#\ \text{mset } (\text{cols } M)\#\}$
using *mat-is-design assms* **by** *simp*
show *?thesis* **proof** (*unfold-locales*)
have $\text{length } (\text{cols } M) \neq 0$ **using** *assms(2)* **by** *auto*
then have $\text{size } \{\#\ \text{map-col-to-block } c . c \in \#\ \text{mset } (\text{cols } M)\#\} \neq 0$ **by** *auto*
then show *incidence-sys.b* $\neq 0$ **by** *simp*
qed
qed

Show the 01 injective function preserves system properties

lemma *inj-on-01-hom-index*:

assumes *inj-on-01-hom* *f*
assumes $i < \dim\text{-row } M \ j < \dim\text{-col } M$
shows $M \ \$\$ (i, j) = 1 \longleftrightarrow (\text{map-mat } f \ M) \ \$\$ (i, j) = 1$
and $M \ \$\$ (i, j) = 0 \longleftrightarrow (\text{map-mat } f \ M) \ \$\$ (i, j) = 0$
proof –
interpret *hom*: *inj-on-01-hom* *f* **using** *assms* **by** *simp*
show $M \ \$\$ (i, j) = 1 \longleftrightarrow (\text{map-mat } f \ M) \ \$\$ (i, j) = 1$
using *assms col-nth-0-or-1-iff*
by (*simp add: hom.inj-1-iff*)
show $M \ \$\$ (i, j) = 0 \longleftrightarrow (\text{map-mat } f \ M) \ \$\$ (i, j) = 0$
using *assms col-nth-0-or-1-iff*
by (*simp add: hom.inj-0-iff*)

qed

lemma *preserve-non-empty*:

assumes *inj-on-01-hom* *f*

assumes $j < \dim\text{-col } M$

shows $\text{non-empty-col } M j \longleftrightarrow \text{non-empty-col } (\text{map-mat } f M) j$

proof(*simp add: non-empty-col-def, intro iffI*)

interpret *hom*: *inj-on-01-hom* *f* **using** *assms(1)* **by** *simp*

assume $\exists k. k \neq 0 \wedge k \in \$ \text{col } M j$

then obtain *k* **where** *kneq*: $k \neq 0$ **and** *kin*: $k \in \$ \text{col } M j$ **by** *blast*

then have $f k \in \$ \text{col } (\text{map-mat } f M) j$ **using** *vec-contains-img*

by (*metis assms(2) col-map-mat*)

then have $f k \neq 0$ **using** *assms(1) kneq kin assms(2) col-elems-ss01 hom.inj-0-iff*
by *blast*

thus $\exists k. k \neq 0 \wedge k \in \$ \text{col } (\text{map-mat } f M) j$

using $\langle f k \in \$ \text{col } (\text{map-mat } f M) j \rangle$ **by** *blast*

next

interpret *hom*: *inj-on-01-hom* *f* **using** *assms(1)* **by** *simp*

assume $\exists k. k \neq 0 \wedge k \in \$ \text{col } (\text{map-mat } f M) j$

then obtain *k* **where** *kneq*: $k \neq 0$ **and** *kin*: $k \in \$ \text{col } (\text{map-mat } f M) j$ **by** *blast*

then have $k \in \$ \text{map-vec } f (\text{col } M j)$ **using** *assms(2) col-map-mat* **by** *simp*

then have $k \in f \text{' set}_v (\text{col } M j)$

by (*smt (verit) image-eqI index-map-vec(1) index-map-vec(2) vec-setE vec-setI*)

then obtain *k'* **where** *keq*: $k = f k'$ **and** *kin2*: $k' \in \text{set}_v (\text{col } M j)$

by *blast*

then have $k' \neq 0$ **using** *assms(1) kneq hom.inj-0-iff* **by** *blast*

thus $\exists k. k \neq 0 \wedge k \in \$ \text{col } M j$ **using** *kin2* **by** *auto*

qed

lemma *preserve-mat-rep-num*:

assumes *inj-on-01-hom* *f*

assumes $i < \dim\text{-row } M$

shows $\text{mat-rep-num } M i = \text{mat-rep-num } (\text{map-mat } f M) i$

unfolding *mat-rep-num-def* **using** *injective-lim.lim-inj-hom-count-vec inj-on-01-hom-def*
row-map-mat

by (*metis assms(1) assms(2) inj-on-01-hom.inj-1-iff insert-iff row-elems-ss01*)

lemma *preserve-mat-block-size*:

assumes *inj-on-01-hom* *f*

assumes $j < \dim\text{-col } M$

shows $\text{mat-block-size } M j = \text{mat-block-size } (\text{map-mat } f M) j$

unfolding *mat-block-size-def* **using** *injective-lim.lim-inj-hom-count-vec inj-on-01-hom-def*
col-map-mat

by (*metis assms(1) assms(2) inj-on-01-hom.inj-1-iff insert-iff col-elems-ss01*)

lemma *preserve-mat-point-index*:

assumes *inj-on-01-hom* *f*

assumes $\bigwedge i. i \in I \implies i < \dim\text{-row } M$
shows $\text{mat-point-index } M I = \text{mat-point-index } (\text{map-mat } f M) I$
proof –
have $\bigwedge i j. i \in I \implies j < \dim\text{-col } M \wedge M \text{ } \$\$ (i, j) = 1 \longleftrightarrow$
 $j < \dim\text{-col } (\text{map-mat } f M) \wedge (\text{map-mat } f M) \text{ } \$\$ (i, j) = 1$
using $\text{assms}(2)$ $\text{inj-on-01-hom-index}(1)$ $\text{assms}(1)$ **by** $(\text{metis index-map-mat}(3))$

thus *?thesis* **unfolding** $\text{mat-point-index-def}$
by $(\text{metis } (\text{no-types, opaque-lifting}) \text{index-map-mat}(3))$
qed

lemma $\text{preserve-mat-inter-num}$:
assumes $\text{inj-on-01-hom } f$
assumes $j1 < \dim\text{-col } M \ j2 < \dim\text{-col } M$
shows $\text{mat-inter-num } M j1 j2 = \text{mat-inter-num } (\text{map-mat } f M) j1 j2$
unfolding mat-inter-num-def **using** assms
by $(\text{metis } (\text{no-types, opaque-lifting}) \text{index-map-mat}(2) \text{inj-on-01-hom-index}(1))$

lemma $\text{lift-mat-01-index-iff}$:
 $i < \dim\text{-row } M \implies j < \dim\text{-col } M \implies (\text{lift-01-mat } M) \text{ } \$\$ (i, j) = 0 \longleftrightarrow M \text{ } \$\$$
 $(i, j) = 0$
 $i < \dim\text{-row } M \implies j < \dim\text{-col } M \implies (\text{lift-01-mat } M) \text{ } \$\$ (i, j) = 1 \longleftrightarrow M \text{ } \$\$$
 $(i, j) = 1$
by (simp) $(\text{metis col-nth-0-or-1-iff index-col lift-01-mat-simp}(3) \text{of-zero-neq-one-def zero-neq-one})$

lemma lift-mat-elems : $\text{elements-mat } (\text{lift-01-mat } M) \subseteq \{0, 1\}$
proof –
have $\text{elements-mat } (\text{lift-01-mat } M) = \text{of-zero-neq-one } ' (\text{elements-mat } M)$
by $(\text{simp add: lift-01-mat-def map-mat-elements})$
then have $\text{elements-mat } (\text{lift-01-mat } M) \subseteq \text{of-zero-neq-one } ' \{0, 1\}$ **using** elems01
by fastforce
thus *?thesis*
by simp
qed

lemma lift-mat-is-0-1 : $\text{zero-one-matrix } (\text{lift-01-mat } M)$
using lift-mat-elems **by** (unfold-locales)

lemma $\text{lift-01-mat-distinct-cols}$: $\text{distinct } (\text{cols } M) \implies \text{distinct } (\text{cols } (\text{lift-01-mat } M))$
using $\text{of-injective-lim.mat-cols-hom-lim-distinct-iff lift-01-mat-def}$
by $(\text{metis elems01 map-vec-mat-cols})$

end

Some properties must be further restricted to matrices having a 'a type

locale $\text{zero-one-matrix-ring-1} = \text{zero-one-matrix } M$ **for** $M :: 'b :: \{\text{ring-1}\} \text{ mat}$
begin

lemma *map-col-block-eq*:
assumes $c \in \text{set}(\text{cols } M)$
shows $\text{inc-vec-of } [0..<\text{dim-vec } c] (\text{map-col-to-block } c) = c$
proof (*intro eq-vecI, simp add: map-col-to-block-def inc-vec-of-def, intro impI*)
show $\bigwedge i. i < \text{dim-vec } c \implies c \$ i \neq 1 \implies c \$ i = 0$
using *assms map-col-to-block-elem map-col-to-block-elem-not* **by** *auto*
show $\text{dim-vec } (\text{inc-vec-of } [0..<\text{dim-vec } c] (\text{map-col-to-block } c)) = \text{dim-vec } c$
unfolding *inc-vec-of-def* **by** *simp*
qed

lemma *inc-mat-of-map-rev*: $\text{inc-mat-of } [0..<\text{dim-row } M] (\text{map } \text{map-col-to-block } (\text{cols } M)) = M$
proof (*intro eq-matI, simp-all add: inc-mat-of-def, intro conjI impI*)
show $\bigwedge i j. i < \text{dim-row } M \implies j < \text{dim-col } M \implies i \in \text{map-col-to-block } (\text{col } M j) \implies M \$\$ (i, j) = 1$
by (*simp add: map-col-to-block-elem*)
show $\bigwedge i j. i < \text{dim-row } M \implies j < \text{dim-col } M \implies i \notin \text{map-col-to-block } (\text{col } M j) \implies M \$\$ (i, j) = 0$
by (*metis col-nth-0-or-1-iff dim-col index-col map-col-to-block-elem*)
qed

lemma *M-index-square-itself*: $j < \text{dim-col } M \implies i < \text{dim-row } M \implies (M \$\$ (i, j))^2 = M \$\$ (i, j)$
using *M-not-zero-simp* **by** (*cases M \\$\\$ (i, j) = 0*)(*simp-all, metis power-one*)

lemma *M-col-index-square-itself*: $j < \text{dim-col } M \implies i < \text{dim-row } M \implies ((\text{col } M j) \$ i)^2 = (\text{col } M j) \$ i$
using *index-col M-index-square-itself* **by** *auto*

Scalar Prod Alternative definitions for matrix properties

lemma *scalar-prod-inc-vec-block-size-mat*:
assumes $j < \text{dim-col } M$
shows $(\text{col } M j) \cdot (\text{col } M j) = \text{of-nat } (\text{mat-block-size } M j)$
proof –
have $(\text{col } M j) \cdot (\text{col } M j) = (\sum i \in \{0..<\text{dim-row } M\} . (\text{col } M j) \$ i * (\text{col } M j) \$ i)$
using *assms scalar-prod-def sum.cong* **by** (*smt (verit, ccfv-threshold) dim-col*)

also have $\dots = (\sum i \in \{0..<\text{dim-row } M\} . ((\text{col } M j) \$ i)^2)$
by (*simp add: power2-eq-square*)
also have $\dots = (\sum i \in \{0..<\text{dim-row } M\} . ((\text{col } M j) \$ i))$
using *M-col-index-square-itself assms* **by** *auto*
finally show *?thesis* **using** *sum-vec-def mat-block-size-sum-alt*
by (*metis assms dim-col elems01*)
qed

lemma *scalar-prod-inc-vec-mat-inter-num*:
assumes $j1 < \text{dim-col } M$ $j2 < \text{dim-col } M$

shows $(\text{col } M \ j1) \cdot (\text{col } M \ j2) = \text{of-nat } (\text{mat-inter-num } M \ j1 \ j2)$
proof –
have *split*: $\{0..<\text{dim-row } M\} = \{i \in \{0..<\text{dim-row } M\} \cdot (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\} \cup$
 $\{i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) = 0 \vee M \ \$\$ (i, j2) = 0\}$ **using** *assms*
M-not-zero-simp **by** *auto*
have *inter*: $\{i \in \{0..<\text{dim-row } M\} \cdot (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\}$
 \cap
 $\{i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) = 0 \vee M \ \$\$ (i, j2) = 0\} = \{\}$ **by** *auto*
have $(\text{col } M \ j1) \cdot (\text{col } M \ j2) = (\sum i \in \{0..<\text{dim-row } M\} \cdot (\text{col } M \ j1) \ \$ i * (\text{col } M \ j2) \ \$ i)$
using *assms* *scalar-prod-def* **by** (*metis* (*full-types*) *dim-col*)
also **have** $\dots = (\sum i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) * M \ \$\$ (i, j2))$
using *assms* **by** *simp*
also **have** $\dots = (\sum i \in \{i \in \{0..<\text{dim-row } M\} \cdot (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\} \cdot M \ \$\$ (i, j1) * M \ \$\$ (i, j2))$
 $+ (\sum i \in \{i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) = 0 \vee M \ \$\$ (i, j2) = 0\} \cdot M \ \$\$ (i, j1) * M \ \$\$ (i, j2))$
using *split* *inter* *sum.union-disjoint*[of $\{i \in \{0..<\text{dim-row } M\} \cdot (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\}$
 $\{i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) = 0 \vee M \ \$\$ (i, j2) = 0\} \lambda i \cdot M \ \$\$ (i, j1) * M \ \$\$ (i, j2)$]
by (*metis* (*no-types*, *lifting*) *finite-Un* *finite-atLeastLessThan*)
also **have** $\dots = (\sum i \in \{i \in \{0..<\text{dim-row } M\} \cdot (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\} \cdot 1)$
 $+ (\sum i \in \{i \in \{0..<\text{dim-row } M\} \cdot M \ \$\$ (i, j1) = 0 \vee M \ \$\$ (i, j2) = 0\} \cdot 0)$
using *sum.cong* *mem-Collect-eq* **by** (*smt* (*z3*) *mult.right-neutral* *mult-not-zero*)

finally **have** $(\text{col } M \ j1) \cdot (\text{col } M \ j2) =$
 $\text{of-nat } (\text{card } \{i \cdot i < \text{dim-row } M \wedge (M \ \$\$ (i, j1) = 1) \wedge (M \ \$\$ (i, j2) = 1)\})$
by *simp*
then **show** *?thesis* **using** *mat-inter-num-def*[of $M \ j1 \ j2$] **by** *simp*
qed

end

Any matrix generated by *inc-mat-of* is a 0-1 matrix.

lemma *inc-mat-of-01-mat: zero-one-matrix-ring-1* (*inc-mat-of* $Vs \ Bs$)
by (*unfold-locales*) (*simp* *add: inc-mat-one-zero-elems*)

4.4 Ordered Incidence Systems

We impose an arbitrary ordering on the point set and block collection to enable matrix reasoning. Note that this is also common in computer algebra representations of designs

locale *ordered-incidence-system* =
fixes $\mathcal{V}s :: 'a \text{ list}$ **and** $\mathcal{B}s :: 'a \text{ set list}$
assumes *wf-list*: $b \in \# (\text{mset } \mathcal{B}s) \implies b \subseteq \text{set } \mathcal{V}s$
assumes *distinct*: *distinct* $\mathcal{V}s$

An ordered incidence system, as it is defined on lists, can only represent finite incidence systems

sublocale *ordered-incidence-system* \subseteq *finite-incidence-system* *set* \mathcal{V} *mset* \mathcal{B}
by (*unfold-locales*) (*auto simp add: wf-list*)

lemma *ordered-incidence-sysI*:

assumes *finite-incidence-system* \mathcal{V} \mathcal{B}

assumes $\mathcal{V}s \in$ *permutations-of-set* \mathcal{V} **and** $\mathcal{B}s \in$ *permutations-of-multiset* \mathcal{B}

shows *ordered-incidence-system* $\mathcal{V}s$ $\mathcal{B}s$

proof –

have *veq*: $\mathcal{V} =$ *set* $\mathcal{V}s$ **using** *assms permutations-of-setD(1)* **by** *auto*

have *beq*: $\mathcal{B} =$ *mset* $\mathcal{B}s$ **using** *assms permutations-of-multisetD* **by** *auto*

interpret *fisys*: *finite-incidence-system* *set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ **using** *assms(1)* *veq beq*

by *simp*

show *?thesis* **proof** (*unfold-locales*)

show $\bigwedge b. b \in \#$ *mset* $\mathcal{B}s \implies b \subseteq$ *set* $\mathcal{V}s$ **using** *fisys.wellformed*

by *simp*

show *distinct* $\mathcal{V}s$ **using** *assms permutations-of-setD(2)* **by** *auto*

qed

qed

lemma *ordered-incidence-sysII*:

assumes *finite-incidence-system* \mathcal{V} \mathcal{B} **and** *set* $\mathcal{V}s = \mathcal{V}$ **and** *distinct* $\mathcal{V}s$ **and** *mset* $\mathcal{B}s = \mathcal{B}$

shows *ordered-incidence-system* $\mathcal{V}s$ $\mathcal{B}s$

proof –

interpret *fisys*: *finite-incidence-system* *set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ **using** *assms* **by** *simp*

show *?thesis* **using** *fisys.wellformed* *assms* **by** (*unfold-locales*) (*simp-all*)

qed

context *ordered-incidence-system*

begin

For ease of notation, establish the same notation as for incidence systems

abbreviation $\mathcal{V} \equiv$ *set* $\mathcal{V}s$

abbreviation $\mathcal{B} \equiv$ *mset* $\mathcal{B}s$

Basic properties on ordered lists

lemma *points-indexing*: $\mathcal{V}s \in$ *permutations-of-set* \mathcal{V}

by (*simp add: permutations-of-set-def distinct*)

lemma *blocks-indexing*: $\mathcal{B}s \in$ *permutations-of-multiset* \mathcal{B}

by (*simp add: permutations-of-multiset-def*)

lemma *points-list-empty-iff*: $\mathcal{V}s = [] \iff \mathcal{V} = \{\}$

using *finite-sets points-indexing*

by (*simp add: elem-permutation-of-set-empty-iff*)

lemma *points-indexing-inj*: $\forall i \in I. i < \text{length } \mathcal{V}s \implies \text{inj-on } (!) \mathcal{V}s) I$

by (*simp add: distinct inj-on-nth*)

lemma *blocks-list-empty-iff*: $\mathcal{B}s = [] \iff \mathcal{B} = \{\#\}$
using *blocks-indexing* **by** (*simp*)

lemma *blocks-list-nempty: proper-design* $\mathcal{V} \mathcal{B} \implies \mathcal{B}s \neq []$
using *mset.simps(1) proper-design.design-blocks-nempty* **by** *blast*

lemma *points-list-nempty: proper-design* $\mathcal{V} \mathcal{B} \implies \mathcal{V}s \neq []$
using *proper-design.design-points-nempty points-list-empty-iff* **by** *blast*

lemma *points-list-length*: $\text{length } \mathcal{V}s = v$
using *points-indexing*
by (*simp add: length-finite-permutations-of-set*)

lemma *blocks-list-length*: $\text{length } \mathcal{B}s = b$
using *blocks-indexing length-finite-permutations-of-multiset* **by** *blast*

lemma *valid-points-index*: $i < v \implies \mathcal{V}s ! i \in \mathcal{V}$
using *points-list-length* **by** *simp*

lemma *valid-points-index-cons*: $x \in \mathcal{V} \implies \exists i. \mathcal{V}s ! i = x \wedge i < v$
using *points-list-length* **by** (*auto simp add: in-set-conv-nth*)

lemma *valid-points-index-obtains*:
assumes $x \in \mathcal{V}$
obtains i **where** $\mathcal{V}s ! i = x \wedge i < v$
using *valid-points-index-cons* *assms* **by** *auto*

lemma *valid-blocks-index*: $j < b \implies \mathcal{B}s ! j \in \#\mathcal{B}$
using *blocks-list-length* **by** (*metis nth-mem-mset*)

lemma *valid-blocks-index-cons*: $bl \in \#\mathcal{B} \implies \exists j. \mathcal{B}s ! j = bl \wedge j < b$
by (*auto simp add: in-set-conv-nth*)

lemma *valid-blocks-index-obtains*:
assumes $bl \in \#\mathcal{B}$
obtains j **where** $\mathcal{B}s ! j = bl \wedge j < b$
using *assms valid-blocks-index-cons* **by** *auto*

lemma *block-points-valid-point-index*:
assumes $bl \in \#\mathcal{B} \ x \in bl$
obtains i **where** $i < \text{length } \mathcal{V}s \wedge \mathcal{V}s ! i = x$
using *wellformed valid-points-index-obtains* *assms*
by (*metis points-list-length wf-invalid-point*)

lemma *points-set-index-img*: $\mathcal{V} = \text{image}(\lambda i. (\mathcal{V}s ! i)) (\{..<v\})$
using *valid-points-index-cons valid-points-index* **by** *auto*

lemma *blocks-mset-image*: $\mathcal{B} = \text{image-mset } (\lambda i . (\mathcal{B}s ! i)) (\text{mset-set } \{..<b\})$
by (*simp add: mset-list-by-index*)

lemma *incidence-cond-indexed*[*simp*]: $i < v \implies j < b \implies \text{incident } (\mathcal{V}s ! i) (\mathcal{B}s ! j)$
 $j \longleftrightarrow (\mathcal{V}s ! i) \in (\mathcal{B}s ! j)$
using *incidence-alt-def valid-points-index valid-blocks-index* **by** *simp*

lemma *bij-betw-points-index*: *bij-betw* $(\lambda i . \mathcal{V}s ! i) \{0..<v\} \mathcal{V}$
proof (*simp add: bij-betw-def, intro conjI*)

show *inj-on* $(!) \mathcal{V}s \{0..<v\}$

by (*simp add: points-indexing-inj points-list-length*)

show $(!) \mathcal{V}s ' \{0..<v\} = \mathcal{V}$

proof (*intro subset-antisym subsetI*)

fix x **assume** $x \in (!) \mathcal{V}s ' \{0..<v\}$

then obtain i **where** $x = \mathcal{V}s ! i$ **and** $i < v$ **by** *auto*

then show $x \in \mathcal{V}$

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

next

fix x **assume** $x \in \mathcal{V}$

then obtain i **where** $\mathcal{V}s ! i = x$ **and** $i < v$

using *valid-points-index-cons* **by** *auto*

then show $x \in (!) \mathcal{V}s ' \{0..<v\}$ **by** *auto*

qed

qed

Some lemmas on cardinality due to different set descriptor filters

lemma *card-filter-point-indices*: $\text{card } \{i \in \{0..<v\}. P (\mathcal{V}s ! i)\} = \text{card } \{v \in \mathcal{V} . P v\}$

proof –

have $\{v \in \mathcal{V} . P v\} = (\lambda i . \mathcal{V}s ! i) ' \{i \in \{0..<v\}. P (\mathcal{V}s ! i)\}$

by (*metis Compr-image-eq lessThan-atLeast0 points-set-index-img*)

thus *?thesis* **using** *inj-on-nth points-list-length*

by (*metis (no-types, lifting) card-image distinct lessThan-atLeast0 lessThan-iff mem-Collect-eq*)

qed

lemma *card-block-points-filter*:

assumes $j < b$

shows $\text{card } (\mathcal{B}s ! j) = \text{card } \{i \in \{0..<v\} . (\mathcal{V}s ! i) \in (\mathcal{B}s ! j)\}$

proof –

obtain bl **where** $bl \in \# \mathcal{B}$ **and** *blis*: $bl = \mathcal{B}s ! j$

using *assms* **by** *auto*

then have *cbl*: $\text{card } bl = \text{card } \{v \in \mathcal{V} . v \in bl\}$ **using** *block-size-alt* **by** *simp*

have $\mathcal{V} = (\lambda i . \mathcal{V}s ! i) ' \{0..<v\}$ **using** *bij-betw-points-index*

using *lessThan-atLeast0 points-set-index-img* **by** *presburger*

then have *Set.filter* $(\lambda v . v \in bl) \mathcal{V} = \text{Set.filter } (\lambda v . v \in bl) ((\lambda i . \mathcal{V}s ! i) ' \{0..<v\})$

by *presburger*

have $\text{card } \{i \in \{0..<v\} . (\mathcal{V}s ! i) \in bl\} = \text{card } \{v \in \mathcal{V} . v \in bl\}$

using *card-filter-point-indices* by *simp*
 thus *?thesis* using *cbl blis* by *simp*
 qed

lemma *obtains-two-diff-block-indices*:

assumes $j1 < b$
 assumes $j2 < b$
 assumes $j1 \neq j2$
 assumes $b \geq 2$
 obtains $bl1\ bl2$ where $bl1 \in\# \mathcal{B}$ and $\mathcal{B}s ! j1 = bl1$ and $bl2 \in\# \mathcal{B} - \{\#bl1\#}$
 and $\mathcal{B}s ! j2 = bl2$
 proof –
 have $j1lt$: $min\ j1\ (length\ \mathcal{B}s) = j1$ using *assms* by *auto*
 obtain $bl1$ where $bl1in$: $bl1 \in\# \mathcal{B}$ and $bl1eq$: $\mathcal{B}s ! j1 = bl1$
 using *assms(1) valid-blocks-index* by *blast*
 then have $split$: $\mathcal{B}s = take\ j1\ \mathcal{B}s\ @\ \mathcal{B}s!j1\ \#\ drop\ (Suc\ j1)\ \mathcal{B}s$
 using *assms id-take-nth-drop* by *auto*
 then have $lj1$: $length\ (take\ j1\ \mathcal{B}s) = j1$ using $j1lt$ by (*simp add: length-take[of*
j1 B_s])
 have $\mathcal{B} = mset\ (take\ j1\ \mathcal{B}s\ @\ \mathcal{B}s!j1\ \#\ drop\ (Suc\ j1)\ \mathcal{B}s)$ using $split$ *assms(1)*
 by *presburger*
 then have $bsplit$: $\mathcal{B} = mset\ (take\ j1\ \mathcal{B}s) + \{\#bl1\#\} + mset\ (drop\ (Suc\ j1)\ \mathcal{B}s)$
 by (*simp add: bl1eq*)
 then have $btake$: $\mathcal{B} - \{\#bl1\#\} = mset\ (take\ j1\ \mathcal{B}s) + mset\ (drop\ (Suc\ j1)\ \mathcal{B}s)$
 by *simp*
 thus *?thesis* proof (*cases j2 < j1*)
 case *True*
 then have $j2 < length\ (take\ j1\ \mathcal{B}s)$ using $lj1$ by *simp*
 then obtain $bl2$ where $bl2eq$: $bl2 = (take\ j1\ \mathcal{B}s) ! j2$ by *auto*
 then have $bl2eq2$: $bl2 = \mathcal{B}s ! j2$
 by (*simp add: True*)
 then have $bl2 \in\# \mathcal{B} - \{\#bl1\#\}$ using $btake$
 by (*metis bl2eq <j2 < length (take j1 B_s)> nth-mem-mset union-iff*)
 then show *?thesis* using $bl2eq2\ bl1in\ bl1eq$ that by *auto*
 next
 case *False*
 then have $j2gt$: $j2 \geq Suc\ j1$ using *assms* by *simp*
 then obtain i where ieq : $i = j2 - Suc\ j1$
 by *simp*
 then have $j2eq$: $j2 = (Suc\ j1) + i$ using $j2gt$ by *presburger*
 have $length\ (drop\ (Suc\ j1)\ \mathcal{B}s) = b - (Suc\ j1)$ using *blocks-list-length* by *auto*
 then have $i < length\ (drop\ (Suc\ j1)\ \mathcal{B}s)$ using ieq *assms blocks-list-length*
 using *diff-less-mono j2gt* by *presburger*
 then obtain $bl2$ where $bl2eq$: $bl2 = (drop\ (Suc\ j1)\ \mathcal{B}s) ! i$ by *auto*
 then have $bl2in$: $bl2 \in\# \mathcal{B} - \{\#bl1\#\}$ using $btake\ nth-mem-mset\ union-iff$
 by (*metis <i < length (drop (Suc j1) B_s)>*)
 then have $bl2 = \mathcal{B}s ! j2$ using $bl2eq\ nth-drop\ blocks-list-length\ assms\ j2eq$
 by (*metis Suc-leI*)
 then show *?thesis* using $bl1in\ bl1eq\ bl2in$ that by *auto*

qed
qed

lemma *filter-size-blocks-eq-card-indexes*: $\text{size } \{\# b \in \# \mathcal{B} . P b \# \} = \text{card } \{j \in \{..<(b)\} . P (\mathcal{B}s ! j)\}$

proof –

have $\mathcal{B} = \text{image-mset } (\lambda j . \mathcal{B}s ! j) (\text{mset-set } \{..<(b)\})$
 using *blocks-mset-image* **by** *simp*
 then have *helper*: $\{\# b \in \# \mathcal{B} . P b \# \} = \text{image-mset } (\lambda j . \mathcal{B}s ! j) \{\# j \in \# (\text{mset-set } \{..<b\}) . P (\mathcal{B}s ! j) \# \}$
 by (*simp add: filter-mset-image-mset*)
 have $\text{card } \{j \in \{..<b\} . P (\mathcal{B}s ! j)\} = \text{size } \{\# j \in \# (\text{mset-set } \{..<b\}) . P (\mathcal{B}s ! j) \# \}$
 using *card-size-filter-eq* [of $\{..<b\}$] **by** *simp*
 thus *?thesis* **using** *helper* **by** *simp*
qed

lemma *blocks-index-ne-belong*:

assumes $i1 < \text{length } \mathcal{B}s$
 assumes $i2 < \text{length } \mathcal{B}s$
 assumes $i1 \neq i2$
 shows $\mathcal{B}s ! i2 \in \# \mathcal{B} - \{\#(\mathcal{B}s ! i1)\# \}$
proof (*cases* $\mathcal{B}s ! i1 = \mathcal{B}s ! i2$)
 case *True*
 then have $\text{count } (\text{mset } \mathcal{B}s) (\mathcal{B}s ! i1) \geq 2$ **using** *count-min-2-indices* *assms* **by** *fastforce*
 then have $\text{count } ((\text{mset } \mathcal{B}s) - \{\#(\mathcal{B}s ! i1)\# \}) (\mathcal{B}s ! i1) \geq 1$
 by (*metis* *Nat.le-diff-conv2* *add-leD2* *count-diff* *count-single* *nat-1-add-1*)
 then show *?thesis*
 by (*metis* *True* *count-inI* *not-one-le-zero*)
next
 case *False*
 have $\mathcal{B}s ! i2 \in \# \mathcal{B}$ **using** *assms*
 by *simp*
 then show *?thesis* **using** *False*
 by (*metis* *in-remove1-mset-neq*)
qed

lemma *inter-num-points-filter-def*:

assumes $j1 < b$ $j2 < b$ $j1 \neq j2$
 shows $\text{card } \{x \in \{0..<v\} . ((\mathcal{V}s ! x) \in (\mathcal{B}s ! j1) \wedge (\mathcal{V}s ! x) \in (\mathcal{B}s ! j2)) \} = (\mathcal{B}s ! j1) \cap (\mathcal{B}s ! j2)$
proof –
 have *inter*: $\bigwedge v . v \in \mathcal{V} \implies v \in (\mathcal{B}s ! j1) \wedge v \in (\mathcal{B}s ! j2) \iff v \in (\mathcal{B}s ! j1) \cap (\mathcal{B}s ! j2)$
 by *simp*
 obtain *bl1* *bl2* **where** *bl1in*: $bl1 \in \# \mathcal{B}$ **and** *bl1eq*: $\mathcal{B}s ! j1 = bl1$ **and** *bl2in*: $bl2 \in \# \mathcal{B} - \{\#bl1\# \}$
 and *bl2eq*: $\mathcal{B}s ! j2 = bl2$

using *assms obtains-two-diff-block-indexes*
by (*metis blocks-index-ne-belong size-mset valid-blocks-index*)
have $\text{card } \{x \in \{0..<v\} . (\mathcal{V}s ! x) \in (\mathcal{B}s ! j1) \wedge (\mathcal{V}s ! x) \in (\mathcal{B}s ! j2)\} =$
 $\text{card } \{v \in \mathcal{V} . v \in (\mathcal{B}s ! j1) \wedge v \in (\mathcal{B}s ! j2)\}$
using *card-filter-point-indices* **by** *simp*
also have $\dots = \text{card } \{v \in \mathcal{V} . v \in bl1 \wedge v \in bl2\}$ **using** *bl1eq bl2eq* **by** *simp*
finally show *?thesis* **using** *points-inter-num-rep bl1in bl2in*
by (*simp add: bl1eq bl2eq*)
qed

Define an incidence matrix for this ordering of an incidence system

abbreviation $N :: \text{int mat}$ **where**
 $N \equiv \text{inc-mat-of } \mathcal{V}s \ \mathcal{B}s$

sublocale *zero-one-matrix-ring-1* N
using *inc-mat-of-01-mat* .

lemma *N-alt-def-dim*: $N = \text{mat } v \ b \ (\lambda (i,j) . \text{if } (\text{incident } (\mathcal{V}s ! i) (\mathcal{B}s ! j)) \text{ then } 1 \text{ else } 0)$
using *incidence-cond-indexed inc-mat-of-def*
by (*intro eq-matI*) (*simp-all add: inc-mat-dim-row inc-mat-dim-col inc-matrix-point-in-block-one*
 $\text{inc-matrix-point-not-in-block-zero points-list-length}$)

Matrix Dimension related lemmas

lemma *N-carrier-mat*: $N \in \text{carrier-mat } v \ b$
by (*simp add: N-alt-def-dim*)

lemma *dim-row-is-v*[*simp*]: $\text{dim-row } N = v$
by (*simp add: N-alt-def-dim*)

lemma *dim-col-is-b*[*simp*]: $\text{dim-col } N = b$
by (*simp add: N-alt-def-dim*)

lemma *dim-vec-row-N*: $\text{dim-vec } (\text{row } N \ i) = b$
by (*simp add: N-alt-def-dim*)

lemma *dim-vec-col-N*: $\text{dim-vec } (\text{col } N \ i) = v$ **by** *simp*

lemma *dim-vec-N-col*:
assumes $j < b$
shows $\text{dim-vec } (\text{cols } N \ ! \ j) = v$
proof –
have $\text{cols } N \ ! \ j = \text{col } N \ j$ **using** *assms dim-col-is-b* **by** *simp*
then have $\text{dim-vec } (\text{cols } N \ ! \ j) = \text{dim-vec } (\text{col } N \ j)$ **by** *simp*
thus *?thesis* **using** *dim-col assms* **by** (*simp*)
qed

lemma *N-carrier-mat-01-lift*: $\text{lift-01-mat } N \in \text{carrier-mat } v \ b$

by auto

Transpose properties

lemma *transpose-N-mult-dim*: $\dim\text{-row } (N * N^T) = v \ \dim\text{-col } (N * N^T) = v$
by (*simp-all*)

lemma *N-trans-index-val*: $i < \dim\text{-col } N \implies j < \dim\text{-row } N \implies$
 $N^T \ \ \$\$ (i, j) = (\text{if } (\mathcal{V}s ! j) \in (\mathcal{B}s ! i) \text{ then } 1 \text{ else } 0)$
by (*simp add: inc-mat-of-def*)

Matrix element and index related lemmas

lemma *mat-row-elems*: $i < v \implies \text{vec-set } (\text{row } N \ i) \subseteq \{0, 1\}$
using *points-list-length*
by (*simp add: row-elems-ss01*)

lemma *mat-col-elems*: $j < b \implies \text{vec-set } (\text{col } N \ j) \subseteq \{0, 1\}$
using *blocks-list-length* by (*metis col-elems-ss01 dim-col-is-b*)

lemma *matrix-elems-one-zero*: $i < v \implies j < b \implies N \ \ \$\$ (i, j) = 0 \vee N \ \ \$\$ (i, j) = 1$
by (*metis blocks-list-length inc-matrix-elems-one-zero points-list-length*)

lemma *matrix-point-in-block-one*: $i < v \implies j < b \implies (\mathcal{V}s ! i) \in (\mathcal{B}s ! j) \implies N \ \ \$\$ (i, j) = 1$
by (*metis inc-matrix-point-in-block-one points-list-length blocks-list-length*)

lemma *matrix-point-not-in-block-zero*: $i < v \implies j < b \implies \mathcal{V}s ! i \notin \mathcal{B}s ! j \implies N \ \ \$\$ (i, j) = 0$
by(*metis inc-matrix-point-not-in-block-zero points-list-length blocks-list-length*)

lemma *matrix-point-in-block*: $i < v \implies j < b \implies N \ \ \$\$ (i, j) = 1 \implies \mathcal{V}s ! i \in \mathcal{B}s ! j$
by (*metis blocks-list-length points-list-length inc-matrix-point-in-block*)

lemma *matrix-point-not-in-block*: $i < v \implies j < b \implies N \ \ \$\$ (i, j) = 0 \implies \mathcal{V}s ! i \notin \mathcal{B}s ! j$
by (*metis blocks-list-length points-list-length inc-matrix-point-not-in-block*)

lemma *matrix-point-not-in-block-iff*: $i < v \implies j < b \implies N \ \ \$\$ (i, j) = 0 \iff \mathcal{V}s ! i \notin \mathcal{B}s ! j$
by (*metis blocks-list-length points-list-length inc-matrix-point-not-in-block-iff*)

lemma *matrix-point-in-block-iff*: $i < v \implies j < b \implies N \ \ \$\$ (i, j) = 1 \iff \mathcal{V}s ! i \in \mathcal{B}s ! j$
by (*metis blocks-list-length points-list-length inc-matrix-point-in-block-iff*)

lemma *matrix-subset-implies-one*: $I \subseteq \{.. < v\} \implies j < b \implies (!) \mathcal{V}s \ ' I \subseteq \mathcal{B}s ! j \implies i \in I \implies N \ \ \$\$ (i, j) = 1$

by (metis blocks-list-length points-list-length inc-matrix-subset-implies-one)

lemma *matrix-one-implies-membership*:

$I \subseteq \{..< v\} \implies j < \text{size } \mathcal{B} \implies \forall i \in I. N \$ (i, j) = 1 \implies i \in I \implies \mathcal{V}s ! i \in \mathcal{B}s$
 $! j$

by (simp add: matrix-point-in-block-iff subset-iff)

Incidence Vector's of Incidence Matrix columns

lemma *col-inc-vec-of*: $j < \text{length } \mathcal{B}s \implies \text{inc-vec-of } \mathcal{V}s (\mathcal{B}s ! j) = \text{col } N j$

by (simp add: inc-mat-col-inc-vec)

lemma *inc-vec-eq-iff-blocks*:

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

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

shows $\text{inc-vec-of } \mathcal{V}s bl = \text{inc-vec-of } \mathcal{V}s bl' \longleftrightarrow bl = bl'$

proof (intro iffI eq-vecI, simp-all add: inc-vec-dim assms)

define $v1 :: 'c :: \{\text{ring-1}\} \text{vec}$ where $v1 = \text{inc-vec-of } \mathcal{V}s bl$

define $v2 :: 'c :: \{\text{ring-1}\} \text{vec}$ where $v2 = \text{inc-vec-of } \mathcal{V}s bl'$

assume $a: v1 = v2$

then have $\text{dim-vec } v1 = \text{dim-vec } v2$

by (simp add: inc-vec-dim)

then have $\bigwedge i. i < \text{dim-vec } v1 \implies v1 \$ i = v2 \$ i$ using a by simp

then have $\bigwedge i. i < \text{length } \mathcal{V}s \implies v1 \$ i = v2 \$ i$ by (simp add: v1-def
inc-vec-dim)

then have $\bigwedge i. i < \text{length } \mathcal{V}s \implies (\mathcal{V}s ! i) \in bl \longleftrightarrow (\mathcal{V}s ! i) \in bl'$

using inc-vec-index-one-iff v1-def v2-def by metis

then have $\bigwedge x. x \in \mathcal{V} \implies x \in bl \longleftrightarrow x \in bl'$

using points-list-length valid-points-index-cons by auto

then show $bl = bl'$ using wellformed assms

by (meson subset-antisym subset-eq)

qed

Incidence matrix column properties

lemma *N-col-def*: $j < b \implies i < v \implies (\text{col } N j) \$ i = (\text{if } (\mathcal{V}s ! i \in \mathcal{B}s ! j) \text{ then } 1 \text{ else } 0)$

by (metis inc-mat-col-def points-list-length blocks-list-length)

lemma *N-col-def-indiv*: $j < b \implies i < v \implies \mathcal{V}s ! i \in \mathcal{B}s ! j \implies (\text{col } N j) \$ i = 1$

$j < b \implies i < v \implies \mathcal{V}s ! i \notin \mathcal{B}s ! j \implies (\text{col } N j) \$ i = 0$

by (simp-all add: inc-matrix-point-in-block-one inc-matrix-point-not-in-block-zero
points-list-length)

lemma *N-col-list-map-elem*: $j < b \implies i < v \implies$

$\text{col } N j \$ i = \text{map-vec } (\lambda x. \text{if } (x \in (\mathcal{B}s ! j)) \text{ then } 1 \text{ else } 0) (\text{vec-of-list } \mathcal{V}s) \$ i$

by (metis inc-mat-col-list-map-elem points-list-length blocks-list-length)

lemma *N-col-list-map*: $j < b \implies \text{col } N j = \text{map-vec } (\lambda x. \text{if } (x \in (\mathcal{B}s ! j)) \text{ then } 1 \text{ else } 0) (\text{vec-of-list } \mathcal{V}s)$

by (metis inc-mat-col-list-map blocks-list-length)

lemma *N-col-mset-point-set-img*: $j < b \implies$
 $vec\text{-mset } (col\ N\ j) = image\text{-mset } (\lambda\ x.\ if\ (x \in (\mathcal{B}s\ !\ j))\ then\ 1\ else\ 0)$ (*mset-set*
 \mathcal{V})
using *vec-mset-img-map N-col-list-map points-indexing*
by (*metis (no-types, lifting) finite-sets permutations-of-multisetD permutations-of-set-altdef*)

lemma *matrix-col-to-block*:

assumes $j < b$
shows $\mathcal{B}s\ !\ j = (\lambda\ k.\ \mathcal{V}s\ !\ k)\ \{i \in \{..\ < v\} . (col\ N\ j)\ \$\ i = 1\}$
proof (*intro subset-antisym subsetI*)
fix x **assume** *assm1*: $x \in \mathcal{B}s\ !\ j$
then have $x \in \mathcal{V}$ **using** *wellformed assms valid-blocks-index* **by** *blast*
then obtain i **where** $vs: \mathcal{V}s\ !\ i = x$ **and** $i < v$
using *valid-points-index-cons* **by** *auto*
then have *inset*: $i \in \{..\ < v\}$
by *fastforce*
then have $col\ N\ j\ \$\ i = 1$ **using** *assm1 N-col-def assms vs*
using $\langle i < v \rangle$ **by** *presburger*
then have $i \in \{i.\ i \in \{..\ < v\} \wedge col\ N\ j\ \$\ i = 1\}$
using *inset* **by** *blast*
then show $x \in (!)\ \mathcal{V}s\ \{i.\ i \in \{..\ < v\} \wedge col\ N\ j\ \$\ i = 1\}$ **using** *vs* **by** *blast*
next
fix x **assume** *assm2*: $x \in ((\lambda\ k.\ \mathcal{V}s\ !\ k)\ \{i \in \{..\ < v\} . col\ N\ j\ \$\ i = 1\})$
then obtain k **where** $x = \mathcal{V}s\ !\ k$ **and** *inner*: $k \in \{i \in \{..\ < v\} . col\ N\ j\ \$\ i = 1\}$
by *blast*
then have *ilt*: $k < v$ **by** *auto*
then have $N\ \$\$ (k, j) = 1$ **using** *inner*
by (*metis (mono-tags) N-col-def assms matrix-point-in-block-iff matrix-point-not-in-block-zero mem-Collect-eq*)
then show $x \in \mathcal{B}s\ !\ j$ **using** *ilt*
using $\langle x = \mathcal{V}s\ !\ k \rangle$ *assms matrix-point-in-block-iff* **by** *blast*
qed

lemma *matrix-col-to-block-v2*: $j < b \implies \mathcal{B}s\ !\ j = (\lambda\ k.\ \mathcal{V}s\ !\ k)\ \{map\text{-col-to-block } (col\ N\ j)\}$
using *matrix-col-to-block map-col-to-block-def* **by** *fastforce*

lemma *matrix-col-in-blocks*: $j < b \implies (!)\ \mathcal{V}s\ \{map\text{-col-to-block } (col\ N\ j)\} \in\# \mathcal{B}$
using *matrix-col-to-block-v2* **by** (*metis (no-types, lifting) valid-blocks-index*)

lemma *inc-matrix-col-block*:

assumes $c \in set\ (cols\ N)$
shows $(\lambda\ x.\ \mathcal{V}s\ !\ x)\ \{map\text{-col-to-block } c\} \in\# \mathcal{B}$
proof –
obtain j **where** $c = col\ N\ j$ **and** $j < b$ **using** *assms cols-length cols-nth in-mset-conv-nth*
ordered-incidence-system-axioms set-mset-mset **by** (*metis dim-col-is-b*)

thus *?thesis*
using *matrix-col-in-blocks* **by** *blast*
qed

Incidence Matrix Row Definitions

lemma *N-row-def*: $j < b \implies i < v \implies (\text{row } N \ i) \ \$ \ j = (\text{if } (\mathcal{V}s \ ! \ i \in \mathcal{B}s \ ! \ j) \ \text{then } 1 \ \text{else } 0)$

by (*metis inc-mat-row-def points-list-length blocks-list-length*)

lemma *N-row-list-map-elem*: $j < b \implies i < v \implies$

$\text{row } N \ i \ \$ \ j = \text{map-vec } (\lambda \ bl . \ \text{if } ((\mathcal{V}s \ ! \ i) \in \ bl) \ \text{then } 1 \ \text{else } 0) \ (\text{vec-of-list } \mathcal{B}s) \ \$ \ j$

by (*metis inc-mat-row-list-map-elem points-list-length blocks-list-length*)

lemma *N-row-list-map*: $i < v \implies$

$\text{row } N \ i = \text{map-vec } (\lambda \ bl . \ \text{if } ((\mathcal{V}s \ ! \ i) \in \ bl) \ \text{then } 1 \ \text{else } 0) \ (\text{vec-of-list } \mathcal{B}s)$

by (*simp add: inc-mat-row-list-map points-list-length blocks-list-length*)

lemma *N-row-mset-blocks-img*: $i < v \implies$

$\text{vec-mset } (\text{row } N \ i) = \text{image-mset } (\lambda \ x . \ \text{if } ((\mathcal{V}s \ ! \ i) \in \ x) \ \text{then } 1 \ \text{else } 0) \ \mathcal{B}$

using *vec-mset-img-map N-row-list-map* **by** *metis*

Alternate Block representations

lemma *block-mat-cond-rep*:

assumes $j < \text{length } \mathcal{B}s$

shows $(\mathcal{B}s \ ! \ j) = \{\mathcal{V}s \ ! \ i \mid i. \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1\}$

proof –

have *cond*: $\bigwedge i. \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1 \longleftrightarrow i \in \{..< v\} \wedge (\text{col } N \ j) \ \$ \ i = 1$

using *assms points-list-length* **by** *auto*

have $(\mathcal{B}s \ ! \ j) = (\lambda \ k . \ \mathcal{V}s \ ! \ k) \ ' \ \{i \in \{..< v\} . (\text{col } N \ j) \ \$ \ i = 1\}$

using *matrix-col-to-block assms* **by** *simp*

also have $\dots = \{\mathcal{V}s \ ! \ i \mid i. \ i \in \{..< v\} \wedge (\text{col } N \ j) \ \$ \ i = 1\}$ **by** *auto*

finally show $(\mathcal{B}s \ ! \ j) = \{\mathcal{V}s \ ! \ i \mid i. \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1\}$

using *Collect-cong cond* **by** *auto*

qed

lemma *block-mat-cond-rep'*: $j < \text{length } \mathcal{B}s \implies (\mathcal{B}s \ ! \ j) = ((!) \ \mathcal{V}s) \ ' \ \{i . \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1\}$

by (*simp add: block-mat-cond-rep setcompr-eq-image*)

lemma *block-mat-cond-rev*:

assumes $j < \text{length } \mathcal{B}s$

shows $\{i . \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1\} = ((\text{List-Index.index}) \ \mathcal{V}s) \ ' \ (\mathcal{B}s \ ! \ j)$

proof (*intro Set.set-eqI iffI*)

fix i **assume** *a1*: $i \in \{i. \ i < \text{length } \mathcal{V}s \wedge N \ \$ \$ (i, j) = 1\}$

then have *ilt1*: $i < \text{length } \mathcal{V}s$ **and** *Ni1*: $N \ \$ \$ (i, j) = 1$ **by** *auto*

then obtain x **where** $\mathcal{V}s \ ! \ i = x$ **and** $x \in (\mathcal{B}s \ ! \ j)$

using *assms inc-matrix-point-in-block* **by** *blast*

then have $\text{List-Index.index } \mathcal{V}s \ x = i$ **using** *distinct index-nth-id ilt1* **by** *auto*

```

then show  $i \in \text{List-Index.index } \mathcal{V}s \text{ ' } \mathcal{B}s ! j$  by (metis  $\langle x \in \mathcal{B}s ! j \rangle \text{ imageI}$ )
next
fix  $i$  assume  $a2: i \in \text{List-Index.index } \mathcal{V}s \text{ ' } \mathcal{B}s ! j$ 
then obtain  $x$  where  $ieq: i = \text{List-Index.index } \mathcal{V}s \text{ } x$  and  $xin: x \in \mathcal{B}s ! j$ 
  by blast
then have  $ilt: i < \text{length } \mathcal{V}s$ 
  by (smt ( $z3$ ) assms index-first index-le-size nat-less-le nth-mem-mset points-list-length)

  valid-points-index-cons wf-invalid-point
then have  $N \text{ } \$\$ (i, j) = 1$  using  $xin \text{ inc-matrix-point-in-block-one}$ 
  by (metis  $ieq \text{ assms index-conv-size-if-notin less-irrefl-nat nth-index}$ )
then show  $i \in \{i. i < \text{length } \mathcal{V}s \wedge N \text{ } \$\$ (i, j) = 1\}$  using  $ilt$  by simp
qed

```

Incidence Matrix incidence system properties

lemma *incomplete-block-col*:

```

assumes  $j < b$ 
assumes incomplete-block ( $\mathcal{B}s ! j$ )
shows  $0 \in \$ (col \ N \ j)$ 

```

proof –

```

obtain  $x$  where  $x \in \mathcal{V}$  and  $x \notin (\mathcal{B}s ! j)$ 
  by (metis Diff-iff assms(2) incomplete-block-proper-subset psubset-imp-ex-mem)
then obtain  $i$  where  $\mathcal{V}s ! i = x$  and  $i < v$ 
  using valid-points-index-cons by blast
then have  $N \text{ } \$\$ (i, j) = 0$ 
  using  $\langle x \notin \mathcal{B}s ! j \rangle \text{ assms(1) matrix-point-not-in-block-zero}$  by blast
then have  $col \ N \ j \ \$ \ i = 0$ 
  using N-col-def  $\langle \mathcal{V}s ! i = x \rangle \langle i < v \rangle \langle x \notin \mathcal{B}s ! j \rangle \text{ assms(1)}$  by fastforce
thus ?thesis using vec-setI
  by (smt ( $z3$ )  $\langle i < v \rangle \text{ dim-col dim-row-is-v}$ )
qed

```

lemma *mat-rep-num-N-row*:

```

assumes  $i < v$ 
shows  $\text{mat-rep-num } N \ i = \mathcal{B} \text{ rep } (\mathcal{V}s ! i)$ 

```

proof –

```

have  $\text{count } (\text{image-mset } (\lambda x . \text{if } ((\mathcal{V}s ! i) \in x) \text{ then } 1 \text{ else } (0 :: \text{int}))) \ \mathcal{B} \ 1 =$ 
   $\text{size } (\text{filter-mset } (\lambda x . (\mathcal{V}s ! i) \in x) \ \mathcal{B})$ 
  using count-mset-split-image-filter[of  $\mathcal{B} \ 1 \ \lambda x . (0 :: \text{int}) \ \lambda x . (\mathcal{V}s ! i) \in x]$  by
simp
then have  $\text{count } (\text{image-mset } (\lambda x . \text{if } ((\mathcal{V}s ! i) \in x) \text{ then } 1 \text{ else } (0 :: \text{int}))) \ \mathcal{B} \ 1 =$ 
   $\mathcal{B} \text{ rep } (\mathcal{V}s ! i)$  by (simp add: point-rep-number-alt-def)
thus ?thesis using N-row-mset-blocks-img assms
  by (simp add: mat-rep-num-def)
qed

```

lemma *point-rep-mat-row-sum*: $i < v \implies \text{sum-vec } (\text{row } N \ i) = \mathcal{B} \text{ rep } (\mathcal{V}s ! i)$

```

using count-vec-sum-ones-alt mat-rep-num-N-row mat-row-elems mat-rep-num-def
by metis

```


lemma *mat-block-size-N-col*:
assumes $j < b$
shows *mat-block-size* $N j = \text{card } (\mathcal{B}s ! j)$
proof –
have *val-b*: $\mathcal{B}s ! j \in \# \mathcal{B}$ **using** *assms valid-blocks-index* **by** *auto*
have $\bigwedge x. x \in \# \text{mset-set } \mathcal{V} \implies (\lambda x. (0 :: \text{int})) x \neq 1$ **using** *zero-neq-one* **by** *simp*
then have *count* (*image-mset* ($\lambda x. \text{if } (x \in (\mathcal{B}s ! j)) \text{ then } 1 \text{ else } (0 :: \text{int}))$)
(*mset-set* \mathcal{V}) $1 =$
size (*filter-mset* ($\lambda x. x \in (\mathcal{B}s ! j)$) (*mset-set* \mathcal{V}))
using *count-mset-split-image-filter* [*of mset-set* \mathcal{V} 1 ($\lambda x. (0 :: \text{int}))$] $\lambda x. x \in$
 $\mathcal{B}s ! j]$
by *simp*
then have *count* (*image-mset* ($\lambda x. \text{if } (x \in (\mathcal{B}s ! j)) \text{ then } 1 \text{ else } (0 :: \text{int}))$)
(*mset-set* \mathcal{V}) $1 = \text{card } (\mathcal{B}s ! j)$
using *val-b block-size-alt* **by** (*simp add: finite-sets*)
thus *?thesis* **using** *N-col-mset-point-set-img* *assms mat-block-size-def* **by** *metis*
qed

lemma *block-size-mat-rep-sum*: $j < b \implies \text{sum-vec } (\text{col } N j) = \text{mat-block-size } N j$
using *count-vec-sum-ones-alt* *mat-block-size-N-col* *mat-block-size-def* **by** (*metis*
mat-col-elems)

lemma *mat-point-index-rep*:
assumes $I \subseteq \{..<v\}$
shows *mat-point-index* $N I = \mathcal{B}$ *index* ($(\lambda i. \mathcal{V}s ! i) \text{ ' } I$)
proof –
have $\bigwedge i. i \in I \implies \mathcal{V}s ! i \in \mathcal{V}$ **using** *assms valid-points-index* **by** *auto*
then have *eqP*: $\bigwedge j. j < \text{dim-col } N \implies ((\lambda i. \mathcal{V}s ! i) \text{ ' } I) \subseteq (\mathcal{B}s ! j) \iff (\forall i$
 $\in I. N \text{ \$\$ } (i, j) = 1)$
proof (*intro iffI subsetI, simp-all*)
show $\bigwedge j. j < \text{length } \mathcal{B}s \implies (\bigwedge i. i \in I \implies \mathcal{V}s ! i \in \mathcal{V}) \implies (!) \mathcal{V}s \text{ ' } I \subseteq \mathcal{B}s !$
 $j \implies$
 $\forall i \in I. N \text{ \$\$ } (i, j) = 1$
using *matrix-subset-implies-one* *assms* **by** *simp*
have $\bigwedge x. x \in (!) \mathcal{V}s \text{ ' } I \implies \exists i \in I. \mathcal{V}s ! i = x$
by *auto*
then show $\bigwedge j x. j < \text{length } \mathcal{B}s \implies \forall i \in I. N \text{ \$\$ } (i, j) = 1 \implies x \in (!) \mathcal{V}s \text{ ' } I$
 $\implies (\bigwedge i. i \in I \implies \mathcal{V}s ! i \in \mathcal{V}) \implies x \in \mathcal{B}s ! j$
using *assms matrix-one-implies-membership* **by** (*metis* *blocks-list-length*)
qed
have *card* $\{j. j < \text{dim-col } N \wedge (\forall i \in I. N \text{ \$\$ } (i, j) = 1)\} =$
card $\{j. j < \text{dim-col } N \wedge ((\lambda i. \mathcal{V}s ! i) \text{ ' } I) \subseteq \mathcal{B}s ! j\}$
using *eqP* **by** (*metis* (*mono-tags, lifting*))
also have $\dots = \text{size } \{\# b \in \# \mathcal{B}. ((\lambda i. \mathcal{V}s ! i) \text{ ' } I) \subseteq b \#\}$
using *filter-size-blocks-eq-card-indexes* **by** *auto*
also have $\dots = \text{points-index } \mathcal{B} ((\lambda i. \mathcal{V}s ! i) \text{ ' } I)$
by (*simp add: points-index-def*)

finally have $\text{card } \{j . j < \text{dim-col } N \wedge (\forall i \in I . N \text{ \$\$}(i, j) = 1)\} = \mathcal{B} \text{ index}$
 $((\lambda i . \mathcal{V}s ! i) ' I)$
by *blast*
thus *?thesis unfolding mat-point-index-def by simp*
qed

lemma *incidence-mat-two-index*: $i1 < v \implies i2 < v \implies$
 $\text{mat-point-index } N \{i1, i2\} = \mathcal{B} \text{ index } \{\mathcal{V}s ! i1, \mathcal{V}s ! i2\}$
using *mat-point-index-two-alt[of i1 N i2] mat-point-index-rep[of {i1, i2}]*
dim-row-is-v
by (*metis (no-types, lifting) empty-subsetI image-empty image-insert insert-subset lessThan-iff*)

lemma *ones-incidence-mat-block-size*:
assumes $j < b$
shows $((u_v \ v) \ v^* \ N) \$ j = \text{mat-block-size } N \ j$
proof –
have $\text{dim-vec } ((u_v \ v) \ v^* \ N) = b$ **by** (*simp*)
then have $((u_v \ v) \ v^* \ N) \$ j = (u_v \ v) \cdot \text{col } N \ j$ **using** *assms by simp*
also have $\dots = (\sum i \in \{0 .. < v\}. (u_v \ v) \$ i * (\text{col } N \ j) \$ i)$
by (*simp add: scalar-prod-def*)
also have $\dots = \text{sum-vec } (\text{col } N \ j)$ **using** *dim-row-is-v by (simp add: sum-vec-def)*
finally show *?thesis using block-size-mat-rep-sum assms by simp*
qed

lemma *mat-block-size-conv*: $j < \text{dim-col } N \implies \text{card } (\mathcal{B}s ! j) = \text{mat-block-size } N \ j$
by (*simp add: mat-block-size-N-col*)

lemma *mat-inter-num-conv*:
assumes $j1 < \text{dim-col } N \ j2 < \text{dim-col } N$
shows $(\mathcal{B}s ! j1) |\cap| (\mathcal{B}s ! j2) = \text{mat-inter-num } N \ j1 \ j2$
proof –
have $\text{eq-sets} : \bigwedge P. (\lambda i . \mathcal{V}s ! i) ' \{i \in \{0..<v\}. P (\mathcal{V}s ! i)\} = \{x \in \mathcal{V} . P \ x\}$
by (*metis Compr-image-eq lessThan-atLeast0 points-set-index-img*)
have $\text{bin} : \mathcal{B}s ! j1 \in \# \mathcal{B} \ \mathcal{B}s ! j2 \in \# \mathcal{B}$ **using** *assms dim-col-is-b by simp-all*
have $(\mathcal{B}s ! j1) |\cap| (\mathcal{B}s ! j2) = \text{card } ((\mathcal{B}s ! j1) \cap (\mathcal{B}s ! j2))$
by (*simp add: intersection-number-def*)
also have $\dots = \text{card } \{x . x \in (\mathcal{B}s ! j1) \wedge x \in (\mathcal{B}s ! j2)\}$
by (*simp add: Int-def*)
also have $\dots = \text{card } \{x \in \mathcal{V}. x \in (\mathcal{B}s ! j1) \wedge x \in (\mathcal{B}s ! j2)\}$ **using** *wellformed bin*
by (*meson wf-invalid-point*)
also have $\dots = \text{card } ((\lambda i . \mathcal{V}s ! i) ' \{i \in \{0..<v\}. (\mathcal{V}s ! i) \in (\mathcal{B}s ! j1) \wedge (\mathcal{V}s ! i) \in (\mathcal{B}s ! j2)\})$
using *eq-sets[of $\lambda x. x \in (\mathcal{B}s ! j1) \wedge x \in (\mathcal{B}s ! j2)$] by simp*
also have $\dots = \text{card } (\{i \in \{0..<v\}. (\mathcal{V}s ! i) \in (\mathcal{B}s ! j1) \wedge (\mathcal{V}s ! i) \in (\mathcal{B}s ! j2)\})$
using *points-indexing-inj card-image*
by (*metis (no-types, lifting) lessThan-atLeast0 lessThan-iff mem-Collect-eq points-list-length*)
also have $\dots = \text{card } (\{i . i < v \wedge (\mathcal{V}s ! i) \in (\mathcal{B}s ! j1) \wedge (\mathcal{V}s ! i) \in (\mathcal{B}s ! j2)\})$

by *auto*
also have $\dots = \text{card} (\{i . i < v \wedge N \ \$\$ (i, j1) = 1 \wedge N \ \$\$ (i, j2) = 1\})$ **using**
assms
by (*metis* (*no-types*, *opaque-lifting*) *inc-mat-dim-col inc-matrix-point-in-block-iff*
points-list-length)
finally have $(\mathcal{B}s ! j1) \mid \cap \mid (\mathcal{B}s ! j2) = \text{card} \{i . i < \text{dim-row } N \wedge N \ \$\$ (i, j1) =$
 $1 \wedge N \ \$\$ (i, j2) = 1\}$
using *dim-row-is-v* **by** *presburger*
thus *?thesis* **using** *assms* **by** (*simp add: mat-inter-num-def*)
qed

lemma *non-empty-col-map-conv:*

assumes $j < \text{dim-col } N$
shows $\text{non-empty-col } N j \longleftrightarrow \mathcal{B}s ! j \neq \{\}$
proof (*intro iffI*)
assume *non-empty-col* $N j$
then obtain i **where** *ilt:* $i < \text{dim-row } N$ **and** $(\text{col } N j) \$ i \neq 0$
using *non-empty-col-obtains assms* **by** *blast*
then have $(\text{col } N j) \$ i = 1$
using *assms*
by (*metis* *N-col-def-indiv(1)* *N-col-def-indiv(2)* *dim-col-is-b dim-row-is-v*)
then have $\mathcal{V}s ! i \in \mathcal{B}s ! j$
by (*smt* (*verit*, *best*) *assms ilt inc-mat-col-def dim-col-is-b inc-mat-dim-col*
inc-mat-dim-row)
thus $\mathcal{B}s ! j \neq \{\}$ **by** *blast*
next
assume $a: \mathcal{B}s ! j \neq \{\}$
have $\mathcal{B}s ! j \in \# \mathcal{B}$ **using** *assms dim-col-is-b* **by** *simp*
then obtain x **where** $x \in \mathcal{B}s ! j$ **and** $x \in \mathcal{V}$ **using** *wellformed a* **by** *auto*
then obtain i **where** $\mathcal{V}s ! i \in \mathcal{B}s ! j$ **and** $i < \text{dim-row } N$ **using** *dim-row-is-v*
using *valid-points-index-cons* **by** *auto*
then have $N \ \$\$ (i, j) = 1$
using *assms* **by** (*meson inc-mat-of-index*)
then show *non-empty-col* $N j$ **using** *non-empty-col-alt-def*
using $\langle i < \text{dim-row } N \rangle$ *assms* **by** *fastforce*
qed

lemma *scalar-prod-inc-vec-inter-num:*

assumes $j1 < b$ $j2 < b$
shows $(\text{col } N j1) \cdot (\text{col } N j2) = (\mathcal{B}s ! j1) \mid \cap \mid (\mathcal{B}s ! j2)$
using *scalar-prod-inc-vec-mat-inter-num assms N-carrier-mat*
by (*simp add: mat-inter-num-conv*)

lemma *scalar-prod-block-size-lift-01:*

assumes $i < b$
shows $((\text{col } (\text{lift-01-mat } N) i) \cdot (\text{col } (\text{lift-01-mat } N) i)) = (\text{of-nat } (\text{card } (\mathcal{B}s ! i)))$
 $:: ('b :: \{\text{ring-1}\})$
proof –
interpret *z1: zero-one-matrix-ring-1 (lift-01-mat N)*

by (*intro-locales*) (*simp add: lift-mat-is-0-1*)
show *?thesis using* *assms z1.scalar-prod-inc-vec-block-size-mat preserve-mat-block-size*
mat-block-size-N-col lift-01-mat-def
 by (*metis inc-mat-dim-col lift-01-mat-simp(2) of-inj-on-01-hom.inj-on-01-hom-axioms*
size-mset)
qed

lemma *scalar-prod-inter-num-lift-01:*
 assumes $j1 < b$ $j2 < b$
 shows $((\text{col } (\text{lift-01-mat } N) \ j1) \cdot (\text{col } (\text{lift-01-mat } N) \ j2)) = (\text{of-nat } ((\mathcal{B}s \ ! \ j1) \ |\cap| \ (\mathcal{B}s \ ! \ j2))) \ :: \ ('b \ :: \ \{\text{ring-1}\}))$
proof –
 interpret *z1: zero-one-matrix-ring-1 (lift-01-mat N)*
 by (*intro-locales*) (*simp add: lift-mat-is-0-1*)
show *?thesis using* *assms z1.scalar-prod-inc-vec-mat-inter-num preserve-mat-inter-num*

mat-inter-num-conv lift-01-mat-def blocks-list-length inc-mat-dim-col
 by (*metis lift-01-mat-simp(2) of-inj-on-01-hom.inj-on-01-hom-axioms*)
qed

The System complement's incidence matrix flips 0's and 1's

lemma *map-block-complement-entry:* $j < b \implies (\text{map block-complement } \mathcal{B}s) \ ! \ j = \text{block-complement } (\mathcal{B}s \ ! \ j)$
 using *blocks-list-length* by (*metis nth-map*)

lemma *complement-mat-entries:*
 assumes $i < v$ and $j < b$
 shows $(\mathcal{V}s \ ! \ i \notin \mathcal{B}s \ ! \ j) \longleftrightarrow (\mathcal{V}s \ ! \ i \in (\text{map block-complement } \mathcal{B}s) \ ! \ j)$
 using *assms block-complement-def map-block-complement-entry valid-points-index*
 by *simp*

lemma *length-blocks-complement:* $\text{length } (\text{map block-complement } \mathcal{B}s) = b$
 by *auto*

lemma *ordered-complement:* *ordered-incidence-system* $\mathcal{V}s$ (*map block-complement* $\mathcal{B}s$)

proof –
 interpret *inc: finite-incidence-system* \mathcal{V} *complement-blocks*
 by (*simp add: complement-finite*)
 have *map inc.block-complement* $\mathcal{B}s \in \text{permutations-of-multiset complement-blocks}$
 using *complement-image* by (*simp add: permutations-of-multiset-def*)
 then show *?thesis using* *ordered-incidence-sysI[of* \mathcal{V} *complement-blocks* $\mathcal{V}s$ (*map*
block-complement $\mathcal{B}s$)]
 by (*simp add: inc.finite-incidence-system-axioms points-indexing*)
qed

interpretation *ordered-comp:* *ordered-incidence-system* $\mathcal{V}s$ (*map block-complement* $\mathcal{B}s$)

using *ordered-complement* **by** *simp*

lemma *complement-mat-entries-val*:
assumes $i < v$ **and** $j < b$
shows *ordered-comp.N* $\$ \$ (i, j) = (if \mathcal{V}s ! i \in \mathcal{B}s ! j \text{ then } 0 \text{ else } 1)$
proof –
have *cond*: $(\mathcal{V}s ! i \notin \mathcal{B}s ! j) \longleftrightarrow (\mathcal{V}s ! i \in (map \text{block-complement } \mathcal{B}s) ! j)$
using *complement-mat-entries* **assms** **by** *simp*
then have *ordered-comp.N* $\$ \$ (i, j) = (if (\mathcal{V}s ! i \in (map \text{block-complement } \mathcal{B}s)$
 $! j) \text{ then } 1 \text{ else } 0)$
using *assms ordered-comp.matrix-point-in-block-one ordered-comp.matrix-point-not-in-block-iff*

by *force*
then show *?thesis* **using** *cond* **by** *simp*
qed

lemma *ordered-complement-mat*: *ordered-comp.N* = *mat v b* $(\lambda (i,j) . if (\mathcal{V}s ! i) \in (\mathcal{B}s ! j) \text{ then } 0 \text{ else } 1)$
using *complement-mat-entries-val* **by** (*intro eq-matI, simp-all*)

lemma *ordered-complement-mat-map*: *ordered-comp.N* = *map-mat* $(\lambda x. if x = 1 \text{ then } 0 \text{ else } 1) N$
apply (*intro eq-matI, simp-all*)
using *ordered-incidence-system.matrix-point-in-block-iff ordered-incidence-system-axioms*

complement-mat-entries-val **by** (*metis blocks-list-length*)

end

Establishing connection between incidence system and ordered incidence system locale

lemma (*in incidence-system*) *alt-ordering-sysI*: $Vs \in \text{permutations-of-set } \mathcal{V} \implies Bs \in \text{permutations-of-multiset } \mathcal{B} \implies \text{ordered-incidence-system } Vs Bs$
by (*unfold-locales*) (*simp-all add: permutations-of-multisetD permutations-of-setD wellformed*)

lemma (*in finite-incidence-system*) *exists-ordering-sysI*: $\exists Vs Bs . Vs \in \text{permutations-of-set } \mathcal{V} \wedge Bs \in \text{permutations-of-multiset } \mathcal{B} \wedge \text{ordered-incidence-system } Vs Bs$
proof –
obtain Vs **where** $Vs \in \text{permutations-of-set } \mathcal{V}$
by (*meson all-not-in-conv finite-sets permutations-of-set-empty-iff*)
obtain Bs **where** $Bs \in \text{permutations-of-multiset } \mathcal{B}$
by (*meson all-not-in-conv permutations-of-multiset-not-empty*)
then show *?thesis* **using** *alt-ordering-sysI* $\langle Vs \in \text{permutations-of-set } \mathcal{V} \rangle$ **by** *blast*

qed

lemma *inc-sys-orderedI*:
assumes *incidence-system* $V B$ **and** *distinct* Vs **and** *set* $Vs = V$ **and** *mset* $Bs = B$
shows *ordered-incidence-system* $Vs Bs$
proof –
interpret *inc*: *incidence-system* $V B$ **using** *assms* **by** *simp*
show *?thesis* **proof** (*unfold-locales*)
show $\bigwedge b. b \in \# \text{ mset } Bs \implies b \subseteq \text{ set } Vs$ **using** *inc.wellformed* *assms* **by** *simp*
show *distinct* Vs **using** *assms(2)* *permutations-of-setD(2)* **by** *auto*
qed
qed

Generalise the idea of an incidence matrix to an unordered context

definition *is-incidence-matrix* :: ' c ' :: $\{\text{ring-1}\}$ *mat* \implies ' a *set* \implies ' a *set multiset* \implies *bool* **where**
is-incidence-matrix $N V B \longleftrightarrow$
 $(\exists Vs Bs. (Vs \in \text{permutations-of-set } V \wedge Bs \in \text{permutations-of-multiset } B \wedge N = (\text{inc-mat-of } Vs Bs)))$

lemma (**in** *incidence-system*) *is-incidence-mat-alt*: *is-incidence-matrix* $N \mathcal{V} \mathcal{B} \longleftrightarrow$

$(\exists Vs Bs. (\text{set } Vs = \mathcal{V} \wedge \text{mset } Bs = \mathcal{B} \wedge \text{ordered-incidence-system } Vs Bs \wedge N = (\text{inc-mat-of } Vs Bs)))$

proof (*intro iffI*, *simp add: is-incidence-matrix-def*)

assume $\exists Vs. Vs \in \text{permutations-of-set } \mathcal{V} \wedge (\exists Bs. Bs \in \text{permutations-of-multiset } \mathcal{B} \wedge N = \text{inc-mat-of } Vs Bs)$

then obtain $Vs Bs$ **where** $Vs \in \text{permutations-of-set } \mathcal{V} \wedge Bs \in \text{permutations-of-multiset } \mathcal{B} \wedge N = \text{inc-mat-of } Vs Bs$

by *auto*

then show $\exists Vs. \text{set } Vs = \mathcal{V} \wedge (\exists Bs. \text{mset } Bs = \mathcal{B} \wedge \text{ordered-incidence-system } Vs Bs \wedge N = \text{inc-mat-of } Vs Bs)$

using *incidence-system.alt-ordering-sysI* *incidence-system-axioms* *permutations-of-multisetD* *permutations-of-setD(1)*

by *blast*

next

assume $\exists Vs Bs. \text{set } Vs = \mathcal{V} \wedge \text{mset } Bs = \mathcal{B} \wedge \text{ordered-incidence-system } Vs Bs \wedge N = \text{inc-mat-of } Vs Bs$

then obtain $Vs Bs$ **where** $s: \text{set } Vs = \mathcal{V}$ **and** $ms: \text{mset } Bs = \mathcal{B}$ **and** *ordered-incidence-system* $Vs Bs$

and $n: N = \text{inc-mat-of } Vs Bs$ **by** *auto*

then interpret *ois*: *ordered-incidence-system* $Vs Bs$ **by** *simp*

have $vs: Vs \in \text{permutations-of-set } \mathcal{V}$

using *ois.points-indexing* s **by** *blast*

have $Bs \in \text{permutations-of-multiset } \mathcal{B}$ **using** *ois.blocks-indexing* ms **by** *blast*

then show *is-incidence-matrix* $N \mathcal{V} \mathcal{B}$ **using** n vs

using *is-incidence-matrix-def* **by** *blast*

qed

lemma (in *ordered-incidence-system*) *is-incidence-mat-true: is-incidence-matrix* N
 $\mathcal{V} \mathcal{B} = \text{True}$
using *blocks-indexing is-incidence-matrix-def points-indexing* **by** *blast*

4.5 Incidence Matrices on Design Subtypes

locale *ordered-design* = *ordered-incidence-system* $\mathcal{V} s \mathcal{B} s$ + *design set* $\mathcal{V} s$ *mset* $\mathcal{B} s$
for $\mathcal{V} s$ **and** $\mathcal{B} s$
begin

lemma *incidence-mat-non-empty-blocks*:
assumes $j < b$
shows $1 \in \$ (col\ N\ j)$
proof –
obtain bl **where** *isbl: $\mathcal{B} s ! j = bl$* **by** *simp*
then have $bl \in \# \mathcal{B}$
using *assms valid-blocks-index* **by** *auto*
then obtain x **where** *inbl: $x \in bl$*
using *blocks-nempty* **by** *blast*
then obtain i **where** *isx: $\mathcal{V} s ! i = x$* **and** *vali: $i < v$*
using $\langle bl \in \# \mathcal{B} \rangle$ *valid-points-index-cons wf-invalid-point* **by** *blast*
then have $N \$ \$ (i, j) = 1$
using $\langle \mathcal{B} s ! j = bl \rangle$ $\langle x \in bl \rangle$ *assms matrix-point-in-block-one* **by** *blast*
thus *?thesis* **using** *vec-setI*
by (*smt (verit, ccfv-SIG) N-col-def isx vali isbl inbl assms dim-vec-col-N of-nat-less-imp-less*)
qed

lemma *all-cols-non-empty: $j < dim-col\ N \implies non-empty-col\ N\ j$*
using *blocks-nempty non-empty-col-map-conv dim-col-is-b* **by** *simp*
end

locale *ordered-simple-design* = *ordered-design* $\mathcal{V} s \mathcal{B} s$ + *simple-design* (*set* $\mathcal{V} s$) *mset* $\mathcal{B} s$ **for** $\mathcal{V} s \mathcal{B} s$
begin

lemma *block-list-distinct: distinct $\mathcal{B} s$*
using *block-mset-distinct* **by** *auto*

lemma *distinct-cols-N: distinct (cols N)*
proof –
have *inj-on* ($\lambda bl . inc-vec-of\ \mathcal{V} s\ bl$) (*set* $\mathcal{B} s$) **using** *inc-vec-eq-iff-blocks*
by (*simp add: inc-vec-eq-iff-blocks inj-on-def*)
then show *?thesis* **using** *distinct-map inc-mat-of-cols-inc-vecs block-list-distinct*
by (*simp add: distinct-map inc-mat-of-cols-inc-vecs*)
qed

lemma *simp-blocks-length-card: length $\mathcal{B} s = card (set\ \mathcal{B} s)$*
using *design-support-def simple-block-size-eq-card* **by** *fastforce*

lemma *blocks-index-inj-on*: $\text{inj-on } (\lambda i . \mathcal{B}s ! i) \{0..<\text{length } \mathcal{B}s\}$
by (*auto simp add: inj-on-def*) (*metis simp-blocks-length-card card-distinct nth-eq-iff-index-eq*)

lemma *x-in-block-set-img*: **assumes** $x \in \text{set } \mathcal{B}s$ **shows** $x \in (!) \mathcal{B}s ' \{0..<\text{length } \mathcal{B}s\}$

proof –

obtain i **where** $\mathcal{B}s ! i = x$ **and** $i < \text{length } \mathcal{B}s$ **using** *assms*

by (*meson in-set-conv-nth*)

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

qed

lemma *blocks-index-simp-bij-betw*: $\text{bij-betw } (\lambda i . \mathcal{B}s ! i) \{0..<\text{length } \mathcal{B}s\} (\text{set } \mathcal{B}s)$
using *blocks-index-inj-on x-in-block-set-img* **by** (*auto simp add: bij-betw-def*)

lemma *blocks-index-simp-unique*: $i1 < \text{length } \mathcal{B}s \implies i2 < \text{length } \mathcal{B}s \implies i1 \neq i2$
 $\implies \mathcal{B}s ! i1 \neq \mathcal{B}s ! i2$

using *block-list-distinct nth-eq-iff-index-eq* **by** *blast*

lemma *lift-01-distinct-cols-N*: $\text{distinct } (\text{cols } (\text{lift-01-mat } N))$
using *lift-01-mat-distinct-cols distinct-cols-N* **by** *simp*

end

locale *ordered-proper-design* = *ordered-design* $\mathcal{V}s$ $\mathcal{B}s$ + *proper-design set* $\mathcal{V}s$ *mset* $\mathcal{B}s$

for $\mathcal{V}s$ **and** $\mathcal{B}s$

begin

lemma *mat-is-proper*: *proper-inc-mat* N

using *design-blocks-nempty v-non-zero*

by (*auto simp add: proper-inc-mat-def*)

end

locale *ordered-constant-rep* = *ordered-proper-design* $\mathcal{V}s$ $\mathcal{B}s$ + *constant-rep-design* *set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ r

for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** r

begin

lemma *incidence-mat-rep-num*: $i < v \implies \text{mat-rep-num } N i = r$

using *mat-rep-num-N-row rep-number valid-points-index* **by** *simp*

lemma *incidence-mat-rep-num-sum*: $i < v \implies \text{sum-vec } (\text{row } N i) = r$

using *incidence-mat-rep-num mat-rep-num-N-row*

by (*simp add: point-rep-mat-row-sum*)

lemma *transpose-N-mult-diag*:

assumes $i = j$ **and** $i < v$ **and** $j < v$
shows $(N * N^T) \text{\$ \$ } (i, j) = r$
proof –
have $unsq: \bigwedge k . k < b \implies (N \text{\$ \$ } (i, k))^{\wedge 2} = N \text{\$ \$ } (i, k)$
using $assms(2)$ *matrix-elems-one-zero* **by** *fastforce*
then have $(N * N^T) \text{\$ \$ } (i, j) = (\sum k \in \{0..<b\} . N \text{\$ \$ } (i, k) * N \text{\$ \$ } (j, k))$
using $assms(2)$ $assms(3)$ *transpose-mat-mult-entries[of i N j]* **by** $(simp)$
also have $\dots = (\sum k \in \{0..<b\} . (N \text{\$ \$ } (i, k))^{\wedge 2})$ **using** $assms(1)$
by $(simp \text{ add: power2-eq-square})$
also have $\dots = (\sum k \in \{0..<b\} . N \text{\$ \$ } (i, k))$
by $(meson \text{ atLeastLessThan-iff sum.cong unsq})$
also have $\dots = (\sum k \in \{0..<b\} . (row \ N \ i) \ \$ \ k)$
using $assms(2)$ *dim-col-is-b dim-row-is-v* **by** *auto*
finally have $(N * N^T) \text{\$ \$ } (i, j) = sum-vec \ (row \ N \ i)$
by $(simp \text{ add: sum-vec-def})$
thus *?thesis* **using** *incidence-mat-rep-num-sum*
using $assms(2)$ **by** *presburger*
qed
end

locale *ordered-block-design* = *ordered-proper-design* $\mathcal{V}s \ \mathcal{B}s$ + *block-design set* $\mathcal{V}s$
mset $\mathcal{B}s \ k$
for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** k

begin

lemma *incidence-mat-block-size*: $j < b \implies mat\text{-block-size} \ N \ j = k$
using *mat-block-size-N-col uniform valid-blocks-index* **by** *fastforce*

lemma *incidence-mat-block-size-sum*: $j < b \implies sum\text{-vec} \ (col \ N \ j) = k$
using *incidence-mat-block-size block-size-mat-rep-sum* **by** *presburger*

lemma *ones-mult-incidence-mat-k-index*: $j < b \implies ((u_v \ v) \ v * \ N) \ \$ \ j = k$
using *ones-incidence-mat-block-size uniform incidence-mat-block-size* **by** *blast*

lemma *ones-mult-incidence-mat-k*: $((u_v \ v) \ v * \ N) = k \cdot_v \ (u_v \ b)$
using *ones-mult-incidence-mat-k-index dim-col-is-b* **by** $(intro \ eq\text{-vecI}) \ (simp\text{-all})$

end

locale *ordered-incomplete-design* = *ordered-block-design* $\mathcal{V}s \ \mathcal{B}s \ k$ + *incomplete-design*
 $\mathcal{V} \ \mathcal{B} \ k$
for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** k

begin

lemma *incidence-mat-incomplete*: $j < b \implies 0 \in \$ \ (col \ N \ j)$

using *valid-blocks-index incomplete-block-col incomplete-imp-incomp-block* **by**
blast

end

locale *ordered-t-wise-balance = ordered-proper-design* $\mathcal{V}s$ $\mathcal{B}s$ + *t-wise-balance set*
 $\mathcal{V}s$ *mset* $\mathcal{B}s$ ι Λ_t

for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** ι **and** Λ_t

begin

lemma *incidence-mat-des-index:*

assumes $I \subseteq \{0..<v\}$

assumes $\text{card } I = \iota$

shows *mat-point-index* N $I = \Lambda_t$

proof –

have *card:* $\text{card } (!) \mathcal{V}s \text{ ' } I = \iota$ **using** *assms points-indexing-inj*

by (*metis (mono-tags, lifting) card-image ex-nat-less-eq not-le points-list-length subset-iff*)

have $(!) \mathcal{V}s \text{ ' } I \subseteq \mathcal{V}$ **using** *assms*

by (*metis atLeastLessThan-iff image-subset-iff subsetD valid-points-index*)

then have \mathcal{B} *index* $(!) \mathcal{V}s \text{ ' } I = \Lambda_t$ **using** *balanced assms(2) card* **by** *simp*

thus *?thesis* **using** *mat-point-index-rep assms(1) lessThan-atLeast0* **by** *presburger*

qed

end

locale *ordered-pairwise-balance = ordered-t-wise-balance* $\mathcal{V}s$ $\mathcal{B}s$ 2 Λ + *pairwise-balance*
 $\text{set } \mathcal{V}s$ *mset* $\mathcal{B}s$ Λ

for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** Λ

begin

lemma *incidence-mat-des-two-index:*

assumes $i1 < v$

assumes $i2 < v$

assumes $i1 \neq i2$

shows *mat-point-index* N $\{i1, i2\} = \Lambda$

using *incidence-mat-des-index incidence-mat-two-index*

proof –

have $\mathcal{V}s ! i1 \neq \mathcal{V}s ! i2$ **using** *assms(3)*

by (*simp add: assms(1) assms(2) distinct nth-eq-iff-index-eq points-list-length*)

then have *pair:* $\text{card } \{\mathcal{V}s ! i1, \mathcal{V}s ! i2\} = 2$ **using** *card-2-iff* **by** *blast*

have $\{\mathcal{V}s ! i1, \mathcal{V}s ! i2\} \subseteq \mathcal{V}$ **using** *assms*

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

then have \mathcal{B} *index* $\{\mathcal{V}s ! i1, \mathcal{V}s ! i2\} = \Lambda$ **using** *pair*

using *balanced* **by** *blast*

thus *?thesis* **using** *incidence-mat-two-index assms* **by** *simp*

qed

lemma *transpose-N-mult-off-diag*:
assumes $i \neq j$ **and** $i < v$ **and** $j < v$
shows $(N * N^T) \text{ \#\# } (i, j) = \Lambda$
proof –
have *rev*: $\bigwedge k. k \in \{0..<b\} \implies \neg (N \text{ \#\# } (i, k) = 1 \wedge N \text{ \#\# } (j, k) = 1) \longleftrightarrow N \text{ \#\# } (i, k) = 0 \vee N \text{ \#\# } (j, k) = 0$
using *assms matrix-elems-one-zero* **by** *auto*
then have *split*: $\{0..<b\} = \{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 1 \wedge N \text{ \#\# } (j, k) = 1\} \cup \{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 0 \vee N \text{ \#\# } (j, k) = 0\}$
by *blast*
have *zero*: $\bigwedge k. k \in \{0..<b\} \implies N \text{ \#\# } (i, k) = 0 \vee N \text{ \#\# } (j, k) = 0 \implies N \text{ \#\# } (i, k) * N \text{ \#\# } (j, k) = 0$
by *simp*
have *djnt*: $\{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 1 \wedge N \text{ \#\# } (j, k) = 1\} \cap \{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 0 \vee N \text{ \#\# } (j, k) = 0\} = \{\}$ **using** *rev* **by** *auto*
have *fin1*: *finite* $\{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 1 \wedge N \text{ \#\# } (j, k) = 1\}$ **by** *simp*
have *fin2*: *finite* $\{k \in \{0..<b\}. N \text{ \#\# } (i, k) = 0 \vee N \text{ \#\# } (j, k) = 0\}$ **by** *simp*
have $(N * N^T) \text{ \#\# } (i, j) = (\sum k \in \{0..<b\}. N \text{ \#\# } (i, k) * N \text{ \#\# } (j, k))$
using *assms(2) assms(3) transpose-mat-mult-entries[of i N j]* **by** *(simp)*
also have $\dots = (\sum k \in \{k' \in \{0..<b\}. N \text{ \#\# } (i, k') = 1 \wedge N \text{ \#\# } (j, k') = 1\} \cup \{k' \in \{0..<b\}. N \text{ \#\# } (i, k') = 0 \vee N \text{ \#\# } (j, k') = 0\}). N \text{ \#\# } (i, k) * N \text{ \#\# } (j, k))$
using *split* **by** *metis*
also have $\dots = (\sum k \in \{k' \in \{0..<b\}. N \text{ \#\# } (i, k') = 1 \wedge N \text{ \#\# } (j, k') = 1\}. N \text{ \#\# } (i, k) * N \text{ \#\# } (j, k)) + (\sum k \in \{k' \in \{0..<b\}. N \text{ \#\# } (i, k') = 0 \vee N \text{ \#\# } (j, k') = 0\}. N \text{ \#\# } (i, k) * N \text{ \#\# } (j, k))$
using *fin1 fin2 djnt sum.union-disjoint* **by** *blast*
also have $\dots = \text{card } \{k' \in \{0..<b\}. N \text{ \#\# } (i, k') = 1 \wedge N \text{ \#\# } (j, k') = 1\}$
by *(simp add: zero)*
also have $\dots = \text{mat-point-index } N \{i, j\}$
using *assms mat-point-index-two-alt[of i N j]* **by** *simp*
finally show *?thesis* **using** *incidence-mat-des-two-index assms* **by** *simp*
qed

end

context *pairwise-balance*

begin

lemma *ordered-pbdI*:

assumes $\mathcal{B} = \text{mset } \mathcal{B}s$ **and** $\mathcal{V} = \text{set } \mathcal{V}s$ **and** *distinct* $\mathcal{V}s$

shows *ordered-pairwise-balance* $\mathcal{V}s \ \mathcal{B}s \ \Lambda$

proof –

interpret *ois*: *ordered-incidence-system* $\mathcal{V}s \ \mathcal{B}s$

using *ordered-incidence-sysII assms finite-incidence-system-axioms* **by** *blast*

show *?thesis* **using** *b-non-zero blocks-nempty assms t-lt-order balanced*

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

qed
end

locale *ordered-regular-pairwise-balance* = *ordered-pairwise-balance* $\mathcal{V}s$ $\mathcal{B}s$ Λ +
regular-pairwise-balance set $\mathcal{V}s$ *mset* $\mathcal{B}s$ Λ r **for** $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** Λ **and** r

sublocale *ordered-regular-pairwise-balance* \subseteq *ordered-constant-rep*
by *unfold-locales*

context *ordered-regular-pairwise-balance*
begin

Stinson's Theorem 1.15. Stinson [8] gives an iff condition for incidence matrices of regular pairwise balanced designs. The other direction is proven in the *zero-one-matrix* context

lemma *rpbd-incidence-matrix-cond*: $N * (N^T) = \Lambda \cdot_m (J_m \ v) + (r - \Lambda) \cdot_m (1_m \ v)$

proof (*intro eq-matI*)

fix $i \ j$

assume *ilt*: $i < \text{dim-row } (int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v)$

and *jlt*: $j < \text{dim-col } (int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v)$

then have $(int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) =$

$(int \ \Lambda \cdot_m J_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) + (int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j)$

by *simp*

then have *split*: $(int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) =$

$(int \ \Lambda \cdot_m J_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) + (r - \Lambda) * ((1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j))$

using *ilt jlt by simp*

have *lhs*: $(int \ \Lambda \cdot_m J_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) = \Lambda$ **using** *ilt jlt by simp*

show $(N * N^T) \ \mathbb{S}\mathbb{S} \ (i, j) = (int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j)$

proof (*cases i = j*)

case *True*

then have *rhs*: $(int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) = (r - \Lambda)$ **using** *ilt by fastforce*

have $(int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) = \Lambda + (r - \Lambda)$

using *True jlt by auto*

then have $(int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) = r$

using *reg-index-1t-rep by (simp add: nat-diff-split)*

then show *?thesis using lhs split rhs True transpose-N-mult-diag ilt jlt by*

simp

next

case *False*

then have $(1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j) = 0$ **using** *ilt jlt by simp*

then have $(r - \Lambda) * ((1_m \ v) \ \mathbb{S}\mathbb{S} \ (i, j)) = 0$ **using** *ilt jlt*

by (*simp add: <1_m v > $\mathbb{S}\mathbb{S} \ (i, j) = 0$*)

then show *?thesis using lhs transpose-N-mult-off-diag ilt jlt False by simp*

qed

next

show $\text{dim-row } (N * N^T) = \text{dim-row } (int \ \Lambda \cdot_m J_m \ v + int \ (r - \Lambda) \cdot_m 1_m \ v)$

using *transpose-N-mult-dim(1) by auto*

```

next
  show  $\dim\text{-col } (N * N^T) = \dim\text{-col } (\text{int } \Lambda \cdot_m J_m \vee + \text{int } (r - \Lambda) \cdot_m I_m \vee)$ 
    using transpose-N-mult-dim(1) by auto
qed
end

locale ordered-bibd = ordered-proper-design  $\mathcal{V}s \mathcal{B}s$  + bibd set  $\mathcal{V}s$  mset  $\mathcal{B}s$   $k \Lambda$ 
  for  $\mathcal{V}s$  and  $\mathcal{B}s$  and  $k$  and  $\Lambda$ 

sublocale ordered-bibd  $\subseteq$  ordered-incomplete-design
  by unfold-locales

sublocale ordered-bibd  $\subseteq$  ordered-constant-rep  $\mathcal{V}s \mathcal{B}s$   $r$ 
  by unfold-locales

sublocale ordered-bibd  $\subseteq$  ordered-pairwise-balance
  by unfold-locales

locale ordered-sym-bibd = ordered-bibd  $\mathcal{V}s \mathcal{B}s$   $k \Lambda$  + symmetric-bibd set  $\mathcal{V}s$  mset  $\mathcal{B}s$   $k \Lambda$ 
  for  $\mathcal{V}s$  and  $\mathcal{B}s$  and  $k$  and  $\Lambda$ 

sublocale ordered-sym-bibd  $\subseteq$  ordered-simple-design
  by (unfold-locales)

locale ordered-const-intersect-design = ordered-proper-design  $\mathcal{V}s \mathcal{B}s$  + const-intersect-design
set  $\mathcal{V}s$  mset  $\mathcal{B}s$   $m$ 
  for  $\mathcal{V}s \mathcal{B}s$   $m$ 

locale simp-ordered-const-intersect-design = ordered-const-intersect-design + ordered-simple-design
begin

lemma max-one-block-size-inter:
  assumes  $b \geq 2$ 
  assumes  $bl \in\# \mathcal{B}$ 
  assumes  $\text{card } bl = m$ 
  assumes  $bl2 \in\# \mathcal{B} - \{\#bl\# \}$ 
  shows  $m < \text{card } bl2$ 
proof –
  have sd: simple-design  $\mathcal{V} \mathcal{B}$ 
    by (simp add: simple-design-axioms)
  have bl2in:  $bl2 \in\# \mathcal{B}$  using assms(4)
    by (meson in-diffD)
  have blin:  $bl \in\# \{\#b \in\# \mathcal{B} . \text{card } b = m\# \}$  using assms(3) assms(2) by simp
  then have slt:  $\text{size } \{\#b \in\# \mathcal{B} . \text{card } b = m\# \} = 1$  using simple-const-inter-iff
sd assms(1)

```

by (metis count-empty count-eq-zero-iff less-one nat-less-le size-eq-0-iff-empty)
 then have size $\{\#b \in \#(\mathcal{B} - \{\#bl\#\}) . \text{card } b = m\} = 0$ using blin
 by (smt (verit) add-mset-eq-singleton-iff count-eq-zero-iff count-filter-mset
 filter-mset-add-mset insert-DiffM size-1-singleton-mset size-eq-0-iff-empty)
 then have ne: $\text{card } bl2 \neq m$ using assms(4)
 by (metis (mono-tags, lifting) filter-mset-empty-conv size-eq-0-iff-empty)
 thus ?thesis using inter-num-le-block-size assms bl2in nat-less-le by presburger
 qed

lemma block-size-inter-num-cases:

assumes $bl \in \# \mathcal{B}$
 assumes $b \geq 2$
 shows $m < \text{card } bl \vee (\text{card } bl = m \wedge (\forall bl' \in \#(\mathcal{B} - \{\#bl\#\}) . m < \text{card } bl'))$
 proof (cases $\text{card } bl = m$)
 case True
 have $(\bigwedge bl'. bl' \in \#(\mathcal{B} - \{\#bl\#\}) \implies m < \text{card } bl')$
 using max-one-block-size-inter True assms by simp
 then show ?thesis using True by simp
 next
 case False
 then have $m < \text{card } bl$ using assms inter-num-le-block-size nat-less-le by presburger
 then show ?thesis by simp
 qed

lemma indexed-const-intersect:

assumes $j1 < b$
 assumes $j2 < b$
 assumes $j1 \neq j2$
 shows $(\mathcal{B}s ! j1) \cap (\mathcal{B}s ! j2) = m$
 proof -
 obtain $bl1 \ bl2$ where $bl1 \in \# \mathcal{B}$ and $\mathcal{B}s ! j1 = bl1$ and $bl2 \in \# \mathcal{B} - \{\#bl1\#\}$
 and $\mathcal{B}s ! j2 = bl2$
 using obtains-two-diff-block-indices assms by fastforce
 thus ?thesis by (simp add: const-intersect)
 qed

lemma const-intersect-block-size-diff:

assumes $j' < b$ and $j < b$ and $j \neq j'$ and $\text{card } (\mathcal{B}s ! j') = m$ and $b \geq 2$
 shows $\text{card } (\mathcal{B}s ! j) - m > 0$
 proof -
 obtain $bl1 \ bl2$ where $bl1 \in \# \mathcal{B}$ and $\mathcal{B}s ! j' = bl1$ and $bl2 \in \# \mathcal{B} - \{\#bl1\#\}$
 and $\mathcal{B}s ! j = bl2$
 using assms(1) assms(2) assms(3) obtains-two-diff-block-indices by fastforce
 then have $m < \text{card } (bl2)$
 using max-one-block-size-inter assms(4) assms(5) by blast
 thus ?thesis
 by (simp add: $\langle \mathcal{B}s ! j = bl2 \rangle$)
 qed

```

lemma scalar-prod-inc-vec-const-inter:
  assumes  $j1 < b$   $j2 < b$   $j1 \neq j2$ 
  shows  $(\text{col } N \ j1) \cdot (\text{col } N \ j2) = m$ 
  using scalar-prod-inc-vec-inter-num indexed-const-intersect assms by simp
end

```

4.6 Zero One Matrix Incidence System Existence

We prove 0-1 matrices with certain properties imply the existence of an incidence system with particular properties. This leads to Stinson's theorem in the other direction [8]

```

context zero-one-matrix
begin

```

```

lemma mat-is-ordered-incidence-sys: ordered-incidence-system  $[0..<(\text{dim-row } M)]$ 
   $(\text{map } (\text{map-col-to-block}) (\text{cols } M))$ 
  apply (unfold-locales, simp-all)
  using map-col-to-block-wf atLeastLessThan-upt by blast

```

```

interpretation mat-ord-inc-sys: ordered-incidence-system  $[0..<(\text{dim-row } M)]$   $(\text{map}$ 
   $(\text{map-col-to-block}) (\text{cols } M))$ 
  by (simp add: mat-is-ordered-incidence-sys)

```

```

lemma mat-ord-inc-sys-N: mat-ord-inc-sys.N = lift-01-mat M
  by (intro eq-matI, simp-all add: inc-mat-of-def map-col-to-block-elim)
  (metis lift-01-mat-simp(3) lift-mat-01-index-iff(2) of-zero-neq-one-def)

```

```

lemma map-col-to-block-mat-rep-num:
  assumes  $x < \text{dim-row } M$ 
  shows  $(\{\# \text{ map-col-to-block } c . c \in \# \text{ mset } (\text{cols } M) \# \} \text{ rep } x) = \text{mat-rep-num } M$ 
   $x$ 

```

proof –

```

  have mat-rep-num M x = mat-rep-num (lift-01-mat M) x
    using preserve-mat-rep-num mat-ord-inc-sys-N
    by (metis assms lift-01-mat-def of-inj-on-01-hom.inj-on-01-hom-axioms)
  then have mat-rep-num M x = (mat-rep-num mat-ord-inc-sys.N x) using
  mat-ord-inc-sys-N by (simp)
  then have mat-rep-num M x = mset (map (map-col-to-block) (cols M)) rep x
    using assms atLeastLessThan-upt card-atLeastLessThan mat-ord-inc-sys.mat-rep-num-N-row

```

```

  mat-ord-inc-sys-point minus-nat.diff-0 by presburger
  thus ?thesis using ordered-to-mset-col-blocks
  by presburger
qed

```

end

context *zero-one-matrix-ring-1*
begin

lemma *transpose-cond-index-vals*:

assumes $M * (M^T) = \Lambda \cdot_m (J_m (\dim\text{-row } M)) + (r - \Lambda) \cdot_m (I_m (\dim\text{-row } M))$
assumes $i < \dim\text{-row } (M * (M^T))$
assumes $j < \dim\text{-col } (M * (M^T))$
shows $i = j \implies (M * (M^T)) \text{ \#\# } (i, j) = r$ $i \neq j \implies (M * (M^T)) \text{ \#\# } (i, j) = \Lambda$
using *assms* **by** *auto*

end

locale *zero-one-matrix-int* = *zero-one-matrix-ring-1* **for** $M :: \text{int mat}$
begin

Some useful conditions on the transpose product for matrix system properties

lemma *transpose-cond-diag-r*:

assumes $i < \dim\text{-row } (M * (M^T))$
assumes $\bigwedge j. i = j \implies (M * (M^T)) \text{ \#\# } (i, j) = r$
shows $\text{mat-rep-num } M \ i = r$

proof –

have *eqr*: $(M * M^T) \text{ \#\# } (i, i) = r$ **using** *assms(2)*
by *simp*
have *unsq*: $\bigwedge k. k < \dim\text{-col } M \implies (M \text{ \#\# } (i, k))^{\wedge 2} = M \text{ \#\# } (i, k)$
using *assms elems01* **by** *fastforce*
have *sum-vec* $(\text{row } M \ i) = (\sum k \in \{0..<(\dim\text{-col } M)\} . (\text{row } M \ i) \ \$ \ k)$
using *assms* **by** (*simp add: sum-vec-def*)
also have $\dots = (\sum k \in \{0..<(\dim\text{-col } M)\} . M \ \#\# \ (i, k))$
using *assms* **by** *auto*
also have $\dots = (\sum k \in \{0..<(\dim\text{-col } M)\} . M \ \#\# \ (i, k)^{\wedge 2})$
using *atLeastLessThan-iff sum.cong unsq* **by** *simp*
also have $\dots = (\sum k \in \{0..<(\dim\text{-col } M)\} . M \ \#\# \ (i, k) * M \ \#\# \ (i, k))$
using *assms* **by** (*simp add: power2-eq-square*)
also have $\dots = (M * M^T) \text{ \#\# } (i, i)$
using *assms transpose-mat-mult-entries*[of $i \ M \ i$] **by** *simp*
finally have $\text{sum-vec } (\text{row } M \ i) = r$ **using** *eqr* **by** *simp*
thus *?thesis* **using** *mat-rep-num-sum-alt*
by (*metis assms(1) elems01 index-mult-mat(2) of-nat-eq-iff*)

qed

lemma *transpose-cond-non-diag*:

assumes $i1 < \dim\text{-row } (M * (M^T))$
assumes $i2 < \dim\text{-row } (M * (M^T))$
assumes $i1 \neq i2$
assumes $\bigwedge j. j \neq i \implies i < \dim\text{-row } (M * (M^T)) \implies j < \dim\text{-row } (M * (M^T)) \implies (M * (M^T)) \text{ \#\# } (i, j) = \Lambda$
shows $\Lambda = \text{mat-point-index } M \ \{i1, i2\}$

proof –

have *ilt*: $i1 < \dim\text{-row } M \ i2 < \dim\text{-row } M$
using *assms(1) assms(2) by auto*
have *rev*: $\bigwedge k. k \in \{0..<\dim\text{-col } M\} \implies$
 $\neg (M \ \$\$ (i1, k) = 1 \wedge M \ \$\$ (i2, k) = 1) \longleftrightarrow M \ \$\$ (i1, k) = 0 \vee M \ \$\$ (i2,$
 $k) = 0$
using *assms elems01 by fastforce*
then have *split*: $\{0..<\dim\text{-col } M\} = \{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 1 \wedge$
 $M \ \$\$ (i2, k) = 1\} \cup$
 $\{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 0 \vee M \ \$\$ (i2, k) = 0\}$
by *blast*
have *zero*: $\bigwedge k. k \in \{0..<\dim\text{-col } M\} \implies M \ \$\$ (i1, k) = 0 \vee M \ \$\$ (i2, k) =$
 $0 \implies M \ \$\$ (i1, k) * M \ \$\$ (i2, k) = 0$
by *simp*
have *djnt*: $\{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 1 \wedge M \ \$\$ (i2, k) = 1\} \cap$
 $\{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 0 \vee M \ \$\$ (i2, k) = 0\} = \{\}$
using *rev by auto*
have *fin1*: *finite* $\{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 1 \wedge M \ \$\$ (i2, k) = 1\}$
by *simp*
have *fin2*: *finite* $\{k \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k) = 0 \vee M \ \$\$ (i2, k) = 0\}$
by *simp*
have *mat-point-index* $M \ \{i1, i2\} = \text{card } \{k' \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k')$
 $= 1 \wedge M \ \$\$ (i2, k') = 1\}$
using *mat-point-index-two-alt ilt assms(3) by auto*
then have *mat-point-index* $M \ \{i1, i2\} =$
 $(\sum k \in \{k' \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k') = 1 \wedge M \ \$\$ (i2, k') = 1\} . M \ \$\$$
 $(i1, k) * M \ \$\$ (i2, k)) +$
 $(\sum k \in \{k' \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k') = 0 \vee M \ \$\$ (i2, k') = 0\} . M \ \$\$$
 $(i1, k) * M \ \$\$ (i2, k))$
by (*simp add: zero*)
also have $\dots = (\sum k \in (\{k' \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k') = 1 \wedge M \ \$\$ (i2,$
 $k') = 1\} \cup$
 $\{k' \in \{0..<\dim\text{-col } M\}. M \ \$\$ (i1, k') = 0 \vee M \ \$\$ (i2, k') = 0\}) . M \ \$\$ (i1,$
 $k) * M \ \$\$ (i2, k))$
using *fin1 fin2 djnt sum.union-disjoint by (metis (no-types, lifting))*
also have $\dots = (\sum k \in \{0..<\dim\text{-col } M\} . M \ \$\$ (i1, k) * M \ \$\$ (i2, k))$
using *split by metis*
finally have *mat-point-index* $M \ \{i1, i2\} = (M * (M^T)) \ \$\$ (i1, i2)$
using *assms(1) assms(2) transpose-mat-mult-entries[of i1 M i2] by simp*
thus *?thesis using assms by presburger*
qed

lemma *trans-cond-implies-map-rep-num*:

assumes $M * (M^T) = \Lambda \cdot_m (J_m (\dim\text{-row } M)) + (r - \Lambda) \cdot_m (1_m (\dim\text{-row } M))$

assumes $x < \dim\text{-row } M$

shows (*image-mset map-col-to-block (mset (cols M))*) *rep* $x = r$

proof –

interpret *ois*: *ordered-incidence-system* $[0..<\dim\text{-row } M]$ *map map-col-to-block*
(cols M)

using *mat-is-ordered-incidence-sys* **by** *simp*
have *eq: ois.B rep x = sum-vec (row M x)* **using** *ois.point-rep-mat-row-sum*
by (*simp add: assms(2) inc-mat-of-map-rev*)
then have $\bigwedge j. x = j \implies (M * (M^T)) \$\$ (x, j) = r$ **using** *assms(1) trans-
pose-cond-index-vals*
by (*metis assms(2) index-mult-mat(2) index-mult-mat(3) index-transpose-mat(3)*)

thus *?thesis* **using** *eq transpose-cond-diag-r assms(2) index-mult-mat(2)*
by (*metis map-col-to-block-mat-rep-num*)
qed

lemma *trans-cond-implies-map-index:*

assumes $M * (M^T) = \Lambda \cdot_m (J_m (\dim\text{-row } M)) + (r - \Lambda) \cdot_m (1_m (\dim\text{-row } M))$
assumes $ps \subseteq \{0..<\dim\text{-row } M\}$
assumes $\text{card } ps = 2$
shows (*image-mset map-col-to-block (mset (cols M))*) $\text{index } ps = \Lambda$

proof –

interpret *ois: ordered-incidence-system* $[0..<\dim\text{-row } M]$ *map map-col-to-block* $(\text{cols } M)$

using *mat-is-ordered-incidence-sys* **by** *simp*

obtain $i1\ i2$ **where** $i1in: i1 < \dim\text{-row } M$ **and** $i2in: i2 < \dim\text{-row } M$ **and** $psis: ps = \{i1, i2\}$ **and** $neqi: i1 \neq i2$

using *assms(2) assms(3) card-2-iff insert-subset* **by** (*metis atLeastLessThan-iff*)

have $\text{cond}: \bigwedge j\ i. j \neq i \implies i < \dim\text{-row } (M * (M^T)) \implies j < \dim\text{-row } (M * (M^T)) \implies (M * (M^T)) \$\$ (i, j) = \Lambda$

using *assms(1)* **by** *simp*

then have (*image-mset map-col-to-block (mset (cols M))*) $\text{index } ps = \text{mat-point-index } M\ ps$

using *ois.incidence-mat-two-index psis i1in i2in* **by** (*simp add: neqi inc-mat-of-map-rev*)

thus *?thesis* **using** *cond transpose-cond-non-diag[of i1 i2 Λ] i1in i2in index-mult-mat(2)[of M M^T]*

neqi of-nat-eq-iff psis **by** *simp*

qed

Stinson Theorem 1.15 existence direction

lemma *rpbd-exists:*

assumes $\dim\text{-row } M \geq 2$ – Min two points

assumes $\dim\text{-col } M \geq 1$ – Min one block

assumes $\bigwedge j. j < \dim\text{-col } M \implies 1 \in \$ \text{col } M\ j$ – no empty blocks

assumes $M * (M^T) = \Lambda \cdot_m (J_m (\dim\text{-row } M)) + (r - \Lambda) \cdot_m (1_m (\dim\text{-row } M))$

shows *ordered-regular-pairwise-balance* $[0..<\dim\text{-row } M]$ (*map map-col-to-block (cols M)*) $\Lambda\ r$

proof –

interpret *ois: ordered-incidence-system* $[0..<\dim\text{-row } M]$ (*map map-col-to-block (cols M)*)

using *mat-is-ordered-incidence-sys* **by** *simp*

interpret *pdes: ordered-design* $[0..<\dim\text{-row } M]$ (*map map-col-to-block (cols M)*)

using *assms(2) mat-is-design assms(3)*

by (*simp add: ordered-design-def ois.ordered-incidence-system-axioms*)
show *?thesis* **using** *assms trans-cond-implies-map-index trans-cond-implies-map-rep-num*

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

lemma *vec-k-uniform-mat-block-size*:
assumes $((u_v \text{ (dim-row } M)) \cdot_v M) = k \cdot_v (u_v \text{ (dim-col } M))$
assumes $j < \text{dim-col } M$
shows *mat-block-size* $M j = k$
proof –
have *mat-block-size* $M j = \text{sum-vec (col } M j)$ **using** *assms(2)*
by (*simp add: elems01 mat-block-size-sum-alt*)
also have $\dots = ((u_v \text{ (dim-row } M)) \cdot_v M) \$ j$ **using** *assms(2)*
by (*simp add: sum-vec-def scalar-prod-def*)
finally show *?thesis* **using** *assms(1) assms(2)* **by** (*simp*)
qed

lemma *vec-k-impl-uniform-block-size*:
assumes $((u_v \text{ (dim-row } M)) \cdot_v M) = k \cdot_v (u_v \text{ (dim-col } M))$
assumes $bl \in \# (\text{image-mset map-col-to-block (mset (cols } M)))$
shows *card* $bl = k$
proof –
obtain j **where** *jlt*: $j < \text{dim-col } M$ **and** *bleq*: $bl = \text{map-col-to-block (col } M j)$
using *assms(2)* *obtain-block-index-map-block-set* **by** *blast*
then have *card* $(\text{map-col-to-block (col } M j)) = \text{mat-block-size } M j$
by (*simp add: map-col-to-block-size*)
thus *?thesis* **using** *vec-k-uniform-mat-block-size assms(1) bleq jlt* **by** *blast*
qed

lemma *bibd-exists*:
assumes $\text{dim-col } M \geq 1$ — Min one block
assumes $\bigwedge j. j < \text{dim-col } M \implies 1 \in \$ \text{col } M j$ — no empty blocks
assumes $M * (M^T) = \Lambda \cdot_m (J_m \text{ (dim-row } M)) + (r - \Lambda) \cdot_m (1_m \text{ (dim-row } M))$
assumes $((u_v \text{ (dim-row } M)) \cdot_v M) = k \cdot_v (u_v \text{ (dim-col } M))$
assumes $(r :: \text{nat}) \geq 0$
assumes $k \geq 2 \ k < \text{dim-row } M$
shows *ordered-bibd* $[0..<\text{dim-row } M] (\text{map map-col-to-block (cols } M)) k \ \Lambda$
proof –
interpret *ipbd*: *ordered-regular-pairwise-balance* $[0..<\text{dim-row } M] (\text{map map-col-to-block (cols } M)) \ \Lambda \ r$
using *rpbd-exists assms* **by** *simp*
show *?thesis* **using** *vec-k-impl-uniform-block-size* **by** (*unfold-locales, simp-all add: assms*)
qed

end

4.7 Isomorphisms and Incidence Matrices

If two incidence systems have the same incidence matrix, they are isomorphic. Similarly if two incidence systems are isomorphic there exists an ordering such that they have the same incidence matrix

locale *two-ordered-sys* = *D1: ordered-incidence-system* $\mathcal{V}s$ $\mathcal{B}s$ + *D2: ordered-incidence-system* $\mathcal{V}s'$ $\mathcal{B}s'$

for $\mathcal{V}s$ and $\mathcal{B}s$ and $\mathcal{V}s'$ and $\mathcal{B}s'$

begin

lemma *equal-inc-mat-isomorphism*:

assumes $D1.N = D2.N$

shows *incidence-system-isomorphism* $D1.\mathcal{V}$ $D1.\mathcal{B}$ $D2.\mathcal{V}$ $D2.\mathcal{B}$ $(\lambda x . \mathcal{V}s' ! (List-Index.index \mathcal{V}s x))$

proof (*unfold-locales*)

show *bij-betw* $(\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x)$ $D1.\mathcal{V}$ $D2.\mathcal{V}$

proof –

have *comp*: $(\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x) = (\lambda i . \mathcal{V}s' ! i) \circ (\lambda y . List-Index.index \mathcal{V}s y)$

by (*simp add: comp-def*)

have *leq*: $length \mathcal{V}s = length \mathcal{V}s'$

using *assms* $D1.dim-row-is-v$ $D1.points-list-length$ $D2.dim-row-is-v$ $D2.points-list-length$

by force

have *bij1*: *bij-betw* $(\lambda i . \mathcal{V}s' ! i)$ $\{..<length \mathcal{V}s\}$ $(set \mathcal{V}s')$ using *leq*

by (*simp add: bij-betw-nth D2.distinct*)

have *bij-betw* $(List-Index.index \mathcal{V}s)$ $(set \mathcal{V}s)$ $\{..<length \mathcal{V}s\}$ using $D1.distinct$

by (*simp add: bij-betw-index lessThan-atLeast0*)

thus *?thesis* using *bij-betw-trans comp bij1* **by simp**

qed

next

have *len*: $length (map ((^)(\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x)) \mathcal{B}s) = length \mathcal{B}s'$

using *length-map assms* $D1.dim-col-is-b$ **by force**

have *mat-eq*: $\bigwedge i j . D1.N \ \$\$ (i, j) = D2.N \ \$\$ (i, j)$ using *assms*

by *simp*

have *vslen*: $length \mathcal{V}s = length \mathcal{V}s'$ using *assms*

using $D1.dim-row-is-v$ $D1.points-list-length$ $D2.dim-row-is-v$ $D2.points-list-length$

by force

have $\bigwedge j . j < length \mathcal{B}s' \implies (map ((^)(\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x)) \mathcal{B}s) ! j = \mathcal{B}s' ! j$

proof –

fix j assume $a: j < length \mathcal{B}s'$

then have $(map ((^)(\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x)) \mathcal{B}s) ! j = (\lambda x . \mathcal{V}s' ! List-Index.index \mathcal{V}s x) \text{ ` } (\mathcal{B}s ! j)$

by (*metis* $D1.blocks-list-length$ $D1.dim-col-is-b$ $D2.blocks-list-length$ $D2.dim-col-is-b$ *assms nth-map*)

also have $\dots = (\lambda i . \mathcal{V}s' ! i) \text{ ` } ((\lambda x . List-Index.index \mathcal{V}s x) \text{ ` } (\mathcal{B}s ! j))$

by *blast*

also have $\dots = ((\lambda i . \mathcal{V}s' ! i) \text{ ` } \{i . i < length \mathcal{V}s \wedge D1.N \ \$\$ (i, j) = 1\})$

using *D1.block-mat-cond-rev a assms*
by (*metis (no-types, lifting) D1.blocks-list-length D1.dim-col-is-b D2.blocks-list-length D2.dim-col-is-b*)
also have ... = $((\lambda i . \mathcal{V}s' ! i) \text{ ' } \{i . i < \text{length } \mathcal{V}s' \wedge D2.N \text{ \&\& } (i, j) = 1\})$
using *vslen mat-eq by simp*
finally have $(\text{map } ((\lambda x . \mathcal{V}s' ! \text{List-Index.index } \mathcal{V}s x)) \mathcal{B}s) ! j = (\mathcal{B}s' ! j)$
using *D2.block-mat-cond-rep' a by presburger*
then show $(\text{map } ((\lambda x . \mathcal{V}s' ! \text{List-Index.index } \mathcal{V}s x)) \mathcal{B}s) ! j = (\mathcal{B}s' ! j)$ **by**
simp
qed
then have $\text{map } ((\lambda x . \mathcal{V}s' ! \text{List-Index.index } \mathcal{V}s x)) \mathcal{B}s = \mathcal{B}s'$
using *len nth-equalityI[of (map ((\lambda x . \mathcal{V}s' ! \text{List-Index.index } \mathcal{V}s x)) \mathcal{B}s) \mathcal{B}s']*
by *simp*
then show $\text{image-mset } ((\lambda x . \mathcal{V}s' ! \text{List-Index.index } \mathcal{V}s x)) D1.\mathcal{B} = D2.\mathcal{B}$
using *mset-map by auto*
qed

lemma *equal-inc-mat-isomorphism-ex: D1.N = D2.N \implies $\exists \pi . \text{incidence-system-isomorphism } D1.\mathcal{V} D1.\mathcal{B} D2.\mathcal{V} D2.\mathcal{B} \pi$*
using *equal-inc-mat-isomorphism by auto*

lemma *equal-inc-mat-isomorphism-obtain:*
assumes $D1.N = D2.N$
obtains π **where** *incidence-system-isomorphism D1.V D1.B D2.V D2.B π*
using *equal-inc-mat-isomorphism assms by auto*

end

context *incidence-system-isomorphism*
begin

lemma *exists-eq-inc-mats:*
assumes *finite V finite V'*
obtains N **where** *is-incidence-matrix N V B and is-incidence-matrix N V' B'*

proof –

obtain Vs **where** *vsis: Vs \in permutations-of-set V using assms*
by (*meson all-not-in-conv permutations-of-set-empty-iff*)
obtain Bs **where** *bsis: Bs \in permutations-of-multiset B*
by (*meson all-not-in-conv permutations-of-multiset-not-empty*)
have *inj: inj-on π V using bij*
by (*simp add: bij-betw-imp-inj-on*)
then have *mapvs: map π Vs \in permutations-of-set V' using permutations-of-set-image-inj*
using $\langle Vs \in \text{permutations-of-set } \mathcal{V} \rangle \text{ iso-points-map}$ **by** *blast*
have *permutations-of-multiset (image-mset ((\lambda \pi) B) = map ((\lambda \pi) B) ' permutations-of-multiset B*
using *block-img permutations-of-multiset-image by blast*
then have *mapbs: map ((\lambda \pi) B) Bs \in permutations-of-multiset B' using bsis*
block-img by blast
define $N :: 'c :: \{\text{ring-1}\} \text{ mat}$ **where** $N \equiv \text{inc-mat-of } Vs Bs$

```

have is-incidence-matrix  $N \mathcal{V} \mathcal{B}$ 
  using N-def bsis is-incidence-matrix-def vsis by blast
have  $\bigwedge bl . bl \in (\text{set } Bs) \implies bl \subseteq (\text{set } Vs)$ 
by (meson bsis in-multiset-in-set ordered-incidence-system.wf-list source.alt-ordering-sysI
vsis)
then have  $N = \text{inc-mat-of } (\text{map } \pi \text{ } Vs) (\text{map } ((\cdot) \pi) \text{ } Bs)$ 
  using inc-mat-of-bij-betw inj
  by (metis N-def permutations-of-setD(1) vsis)
then have is-incidence-matrix  $N \mathcal{V}' \mathcal{B}'$ 
  using mapbs mapvs is-incidence-matrix-def by blast
thus ?thesis
  using  $\langle \text{is-incidence-matrix } N \mathcal{V} \mathcal{B} \rangle$  that by auto
qed

end

end

```

5 Dual Systems

The concept of a dual incidence system [3] is an important property in design theory. It enables us to reason on the existence of several different types of design constructs through dual properties [8]

```

theory Dual-Systems imports Incidence-Matrices
begin

```

5.1 Dual Blocks

A dual design of $(\mathcal{V}, \mathcal{B})$, is the design where each block in \mathcal{B} represents a point x , and a block in a dual design is a set of blocks which x is in from the original design. It is important to note that if a block repeats in \mathcal{B} , each instance of the block is a distinct point. As such the definition below uses each block's list index as its identifier. The list of points would simply be the indices $0..<\text{length } Bs$

```

definition dual-blocks :: 'a set  $\Rightarrow$  'a set list  $\Rightarrow$  nat set multiset where
dual-blocks  $\mathcal{V} \mathcal{B}s \equiv \{\# \{y . y < \text{length } \mathcal{B}s \wedge x \in \mathcal{B}s ! y\} . x \in \# (\text{mset-set } \mathcal{V})\# \}$ 

```

```

lemma dual-blocks-wf:  $b \in \# \text{dual-blocks } \mathcal{V} \mathcal{B}s \implies b \subseteq \{0..<\text{length } \mathcal{B}s\}$ 
  by (auto simp add: dual-blocks-def)

```

```

context ordered-incidence-system
begin

```

```

definition dual-blocks-ordered :: nat set list  $(\mathcal{B}s^*)$  where
dual-blocks-ordered  $\equiv \text{map } (\lambda x . \{y . y < \text{length } \mathcal{B}s \wedge x \in \mathcal{B}s ! y\}) \mathcal{V}s$ 

```

```

lemma dual-blocks-ordered-eq: dual-blocks  $\mathcal{V} \mathcal{B}s = \text{mset } (\mathcal{B}s^*)$ 

```

by (auto simp add: distinct dual-blocks-def dual-blocks-ordered-def mset-set-set)

lemma *dual-blocks-len*: $\text{length } \mathcal{B}s^* = \text{length } \mathcal{V}s$
 by (simp add: dual-blocks-ordered-def)

A dual system is an incidence system

sublocale *dual-sys*: *finite-incidence-system* $\{0..<\text{length } \mathcal{B}s\}$ *dual-blocks* $\mathcal{V} \mathcal{B}s$
 using *dual-blocks-wf* by (unfold-locales) (auto)

lemma *dual-is-ordered-inc-sys*: *ordered-incidence-system* $[0..<\text{length } \mathcal{B}s]$ $\mathcal{B}s^*$
 using *inc-sys-orderedI* *dual-blocks-ordered-eq*
 by (metis atLeastLessThan-upt distinct-upt *dual-sys.incidence-system-axioms*)

interpretation *ordered-dual-sys*: *ordered-incidence-system* $[0..<\text{length } \mathcal{B}s]$ $\mathcal{B}s^*$
 using *dual-is-ordered-inc-sys* by simp

5.2 Basic Dual Properties

lemma *ord-dual-blocks-b*: $\text{ordered-dual-sys.b} = v$
 using *dual-blocks-len* by (simp add: points-list-length)

lemma *dual-blocks-b*: $\text{dual-sys.b} = v$
 using *points-list-length*
 by (simp add: dual-blocks-len dual-blocks-ordered-eq)

lemma *dual-blocks-v*: $\text{dual-sys.v} = b$
 by fastforce

lemma *ord-dual-blocks-v*: $\text{ordered-dual-sys.v} = b$
 by fastforce

lemma *dual-point-block*: $i < v \implies \mathcal{B}s^* ! i = \{y. y < \text{length } \mathcal{B}s \wedge (\mathcal{V}s ! i) \in \mathcal{B}s ! y\}$
 by (simp add: dual-blocks-ordered-def points-list-length)

lemma *dual-incidence-iff*: $i < v \implies j < b \implies \mathcal{B}s ! j = bl \implies \mathcal{V}s ! i = x \implies (x \in bl \longleftrightarrow j \in \mathcal{B}s^* ! i)$
 using *dual-point-block* by (intro iffI)(simp-all)

lemma *dual-incidence-iff2*: $i < v \implies j < b \implies (\mathcal{V}s ! i \in \mathcal{B}s ! j \longleftrightarrow j \in \mathcal{B}s^* ! i)$
 using *dual-incidence-iff* by simp

lemma *dual-blocks-point-exists*: $bl \in \# \text{dual-blocks } \mathcal{V} \mathcal{B}s \implies \exists x. x \in \mathcal{V} \wedge bl = \{y. y < \text{length } \mathcal{B}s \wedge x \in \mathcal{B}s ! y\}$
 by (auto simp add: dual-blocks-def)

lemma *dual-blocks-ne-index-ne*: $j1 < \text{length } \mathcal{B}s^* \implies j2 < \text{length } \mathcal{B}s^* \implies \mathcal{B}s^* ! j1 \neq \mathcal{B}s^* ! j2 \implies j1 \neq j2$
 by auto

lemma *dual-blocks-list-index-img: image-mset* $(\lambda x . \mathcal{B}s* ! x)$ $(\text{mset-set } \{0..<\text{length } \mathcal{B}s*\}) = \text{mset } \mathcal{B}s*$
using *lessThan-atLeast0 ordered-dual-sys.blocks-list-length ordered-dual-sys.blocks-mset-image*
by *presburger*

lemma *dual-blocks-elem-iff:*
assumes $j < v$
shows $x \in (\mathcal{B}s* ! j) \longleftrightarrow \mathcal{V}s ! j \in \mathcal{B}s ! x \wedge x < b$
proof (*intro iffI conjI*)
show $x \in \mathcal{B}s* ! j \implies \mathcal{V}s ! j \in \mathcal{B}s ! x$
using *assms ordered-incidence-system.dual-point-block ordered-incidence-system-axioms*
by *fastforce*
show $x \in \mathcal{B}s* ! j \implies x < b$
using *assms dual-blocks-ordered-def dual-point-block by fastforce*
show $\mathcal{V}s ! j \in \mathcal{B}s ! x \wedge x < b \implies x \in \mathcal{B}s* ! j$
by (*metis (full-types) assms blocks-list-length dual-incidence-iff*)
qed

The incidence matrix of the dual of a design is just the transpose

lemma *dual-incidence-mat-eq-trans: ordered-dual-sys.N = N^T*
proof (*intro eq-matI*)
show *dimr: dim-row ordered-dual-sys.N = dim-row N<sup>T **using** *dual-blocks-v by (simp)*
show *dimc: dim-col ordered-dual-sys.N = dim-col N^T* **using** *ord-dual-blocks-b by (simp)*
show $\bigwedge i j. i < \text{dim-row } N^T \implies j < \text{dim-col } N^T \implies \text{ordered-dual-sys.N } \$\$ (i, j) = N^T \$\$ (i, j)$
proof –
fix $i j$ **assume** *ilt: i < dim-row N^T* **assume** *jlt: j < dim-col N^T*
then have *ilt2: i < length B* **using** *dimr*
using *blocks-list-length ord-dual-blocks-v ilt ordered-dual-sys.dim-row-is-v by linarith*
then have *ilt3: i < b* **by** *simp*
have *jlt2: j < v* **using** *jlt*
using *dim-row-is-v by fastforce*
have *ordered-dual-sys.N* $\$\$ (i, j) = (\text{if } ([0..<\text{length } \mathcal{B}s] ! i) \in (\mathcal{B}s* ! j) \text{ then } 1 \text{ else } 0)$
using *dimr dual-blocks-len ilt jlt inc-matrix-elems-one-zero*
by (*metis inc-mat-dim-row inc-matrix-point-in-block-iff index-transpose-mat(3)*)
then have *ordered-dual-sys.N* $\$\$ (i, j) = (\text{if } \mathcal{V}s ! j \in \mathcal{B}s ! i \text{ then } 1 \text{ else } 0)$
using *ilt3 jlt2 dual-incidence-iff2 by simp*
thus *ordered-dual-sys.N* $\$\$ (i, j) = N^T \$\$ (i, j)$
using *ilt3 jlt2 dim-row-is-v dim-col-is-b N-trans-index-val by simp*
qed
qed</sup>*

lemma *dual-incidence-mat-eq-trans-rev*: $(\text{ordered-dual-sys}.N)^T = N$
using *dual-incidence-mat-eq-trans* **by** *simp*

5.3 Incidence System Dual Properties

Many common design properties have a dual in the dual design which enables extensive reasoning. Using incidence matrices and the transpose property these are easy to prove. We leave examples of counting proofs (commented out), to demonstrate how incidence matrices can significantly simplify reasoning.

lemma *dual-blocks-nempty*:
assumes $(\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0)$
assumes $bl \in \# \text{ dual-blocks } \mathcal{V} \mathcal{B}s$
shows $bl \neq \{\}$
proof –
have $bl \in \# \{\# \{y . y < \text{length } \mathcal{B}s \wedge x \in \mathcal{B}s ! y\} . x \in \# (\text{mset-set } \mathcal{V})\# \}$
using *assms dual-blocks-def* **by** *metis*
then obtain x **where** $x \in \# (\text{mset-set } \mathcal{V})$ **and** $blval: bl = \{y . y < \text{length } \mathcal{B}s \wedge x \in \mathcal{B}s ! y\}$
by *blast*
then obtain bl' **where** $bl' \in \# \mathcal{B}$ **and** $xin: x \in bl'$ **using** *assms(1)*
using *point-in-block-rep-min-iff* **by** *auto*
then obtain y **where** $y < \text{length } \mathcal{B}s$ **and** $\mathcal{B}s ! y = bl'$
using *valid-blocks-index-cons* **by** *auto*
then have $y \in bl$
by *(simp add: xin blval)*
thus *?thesis* **by** *blast*
qed

lemma *dual-blocks-size-is-rep*: $j < \text{length } \mathcal{B}s^* \implies \text{card } (\mathcal{B}s^* ! j) = \mathcal{B} \text{ rep } (\mathcal{V}s ! j)$
using *dual-incidence-mat-eq-trans trans-mat-rep-block-size-sym(2)*
by *(metis dual-blocks-len dual-is-ordered-inc-sys inc-mat-dim-row mat-rep-num-N-row*

ordered-incidence-system.mat-block-size-N-col points-list-length size-mset)

lemma *dual-blocks-size-is-rep-obtain*:
assumes $bl \in \# \text{ dual-blocks } \mathcal{V} \mathcal{B}s$
obtains x **where** $x \in \mathcal{V}$ **and** $\text{card } bl = \mathcal{B} \text{ rep } x$
proof –
obtain j **where** $jlt1: j < \text{length } \mathcal{B}s^*$ **and** $bleq: \mathcal{B}s^* ! j = bl$
by *(metis assms dual-blocks-ordered-eq in-mset-conv-nth)*
then have $jlt: j < v$
by *(simp add: dual-blocks-len points-list-length)*
let $?x = \mathcal{V}s ! j$
have $xin: ?x \in \mathcal{V}$ **using** *jlt*
by *(simp add: valid-points-index)*

have $\text{card } \mathcal{B} \text{ rep } ?x$ **using** *dual-blocks-size-is-rep jlt1 bleq by auto*
thus *?thesis* **using** *xin that by auto*
qed

lemma *dual-blocks-rep-is-size:*

assumes $i < \text{length } \mathcal{B}s$

shows $(\text{mset } \mathcal{B}s^*) \text{ rep } i = \text{card } (\mathcal{B}s ! i)$

proof –

have $[0..<\text{length } \mathcal{B}s] ! i = i$ **using** *assms by simp*

then have $(\text{mset } \mathcal{B}s^*) \text{ rep } i = \text{mat-rep-num } \text{ordered-dual-sys.N } i$

using *ordered-dual-sys.mat-rep-num-N-row assms length-upt minus-nat.diff-0*
ordered-dual-sys.points-list-length by presburger

also have $\dots = \text{mat-block-size } (\text{ordered-dual-sys.N})^T i$ **using** *dual-incidence-mat-eq-trans*

trans-mat-rep-block-size-sym(2) **by** *(metis assms inc-mat-dim-col index-transpose-mat(2))*

finally show *?thesis* **using** *dual-incidence-mat-eq-trans-rev*

by *(metis assms blocks-list-length mat-block-size-N-col)*

qed

lemma *dual-blocks-inter-index:*

assumes $j1 < \text{length } \mathcal{B}s^* \ j2 < \text{length } \mathcal{B}s^*$

shows $(\mathcal{B}s^* ! j1) \mid \cap \mid (\mathcal{B}s^* ! j2) = \text{points-index } \mathcal{B} \{ \mathcal{V}s ! j1, \mathcal{V}s ! j2 \}$

proof –

have *assms2: $j1 < v \ j2 < v$* **using** *assms*

by *(simp-all add: dual-blocks-len points-list-length)*

have $(\mathcal{B}s^* ! j1) \mid \cap \mid (\mathcal{B}s^* ! j2) = \text{mat-inter-num } (\text{ordered-dual-sys.N}) \ j1 \ j2$

by *(simp add: assms(1) assms(2) ordered-dual-sys.mat-inter-num-conv)*

also have $\dots = \text{mat-point-index } N \ \{j1, j2\}$ **using** *dual-incidence-mat-eq-trans-rev*

trans-mat-point-index-inter-sym(2)

by *(metis assms inc-mat-dim-col)*

finally show *?thesis* **using** *assms2 incidence-mat-two-index*

by *presburger*

qed

lemma *dual-blocks-points-index-inter:*

assumes $i1 < b \ i2 < b$

shows $(\text{mset } \mathcal{B}s^*) \text{ index } \{i1, i2\} = (\mathcal{B}s ! i1) \mid \cap \mid (\mathcal{B}s ! i2)$

proof –

have $(\text{mset } \mathcal{B}s^*) \text{ index } \{i1, i2\} = \text{mat-point-index } (\text{ordered-dual-sys.N}) \ \{i1, i2\}$

using *assms(1) assms(2) blocks-list-length ord-dual-blocks-v ordered-dual-sys.dim-row-is-v*

ordered-dual-sys.incidence-mat-two-index ordered-dual-sys.mat-ord-inc-sys-point

by *presburger*

also have $\dots = \text{mat-inter-num } N \ i1 \ i2$ **using** *dual-incidence-mat-eq-trans trans-mat-point-index-inter-sym(1)*

by *(metis assms(1) assms(2) dual-incidence-mat-eq-trans-rev ord-dual-blocks-v ordered-dual-sys.dim-row-is-v)*

```

finally show ?thesis using mat-inter-num-conv
  using assms(1) assms(2) by auto
qed

```

```

end

```

5.4 Dual Properties for Design sub types

```

context ordered-design
begin

```

```

lemma dual-is-design:

```

```

  assumes ( $\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0$ ) — Required to ensure no blocks are
  empty

```

```

  shows design {0.. $\text{length } \mathcal{B}s$ } (dual-blocks  $\mathcal{V} \mathcal{B}s$ )

```

```

  using dual-blocks-nempty assms by (unfold-locales) (simp)

```

```

end

```

```

context ordered-proper-design

```

```

begin

```

```

lemma dual-sys-b-non-zero: dual-sys.b  $\neq 0$ 

```

```

  using v-non-zero dual-blocks-b by auto

```

```

lemma dual-is-proper-design:

```

```

  assumes ( $\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0$ ) — Required to ensure no blocks are
  empty

```

```

  shows proper-design {0.. $\text{length } \mathcal{B}s$ } (dual-blocks  $\mathcal{V} \mathcal{B}s$ )

```

```

  using dual-blocks-nempty dual-sys-b-non-zero assms by (unfold-locales) (simp-all)

```

```

end

```

```

context ordered-block-design

```

```

begin

```

```

lemma dual-blocks-const-rep:  $i \in \{0..\text{length } \mathcal{B}s\} \implies (\text{mset } \mathcal{B}s^*) \text{ rep } i = k$ 

```

```

  using dual-blocks-rep-is-size uniform by (metis atLeastLessThan-iff nth-mem-mset)

```

```

lemma dual-blocks-constant-rep-design:

```

```

  assumes ( $\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0$ )

```

```

  shows constant-rep-design {0.. $\text{length } \mathcal{B}s$ } (dual-blocks  $\mathcal{V} \mathcal{B}s$ ) k

```

```

proof —

```

```

  interpret des: proper-design {0.. $\text{length } \mathcal{B}s$ } (dual-blocks  $\mathcal{V} \mathcal{B}s$ )

```

```

  using dual-is-proper-design assms by simp

```

```

  show ?thesis using dual-blocks-const-rep dual-blocks-ordered-eq by (unfold-locales)
  (simp)

```

```

qed

```

end

context *ordered-constant-rep*
begin

lemma *dual-blocks-const-size*: $j < \text{length } \mathcal{B}s^* \implies \text{card } (\mathcal{B}s^* ! j) = r$
using *dual-blocks-rep-is-size dual-blocks-len dual-blocks-size-is-rep* **by** *fastforce*

lemma *dual-is-block-design*: *block-design* $\{0..<\text{length } \mathcal{B}s\}$ (*dual-blocks* \mathcal{V} $\mathcal{B}s$) r
proof –

have $r > 0$ **by** (*simp add: r-gzero*)
then have $(\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0)$ **using** *rep-number* **by** *simp*
then interpret *pdes: proper-design* $\{0..<\text{length } \mathcal{B}s\}$ (*dual-blocks* \mathcal{V} $\mathcal{B}s$)
using *dual-is-proper-design* **by** *simp*
have $\bigwedge bl. bl \in \# \text{ dual-blocks } \mathcal{V} \mathcal{B}s \implies \text{card } bl = r$
using *dual-blocks-const-size*
by (*metis dual-blocks-ordered-eq in-set-conv-nth set-mset-mset*)
thus *?thesis* **by** (*unfold-locales*) (*simp*)

qed

end

context *ordered-pairwise-balance*
begin

lemma *dual-blocks-const-intersect*:
assumes $j1 < \text{length } \mathcal{B}s^* \ j2 < \text{length } \mathcal{B}s^*$
assumes $j1 \neq j2$
shows $(\mathcal{B}s^* ! j1) \cap (\mathcal{B}s^* ! j2) = \Lambda$

proof –

have $\mathcal{V}s ! j1 \neq \mathcal{V}s ! j2$ **using** *assms(3)*
using *assms(1) assms(2) distinct dual-blocks-len nth-eq-iff-index-eq* **by** *auto*
then have $c: \text{card } \{\mathcal{V}s ! j1, \mathcal{V}s ! j2\} = 2$
using *card-2-iff* **by** *blast*
have $ss: \{\mathcal{V}s ! j1, \mathcal{V}s ! j2\} \subseteq \mathcal{V}$ **using** *assms points-list-length*
using *dual-blocks-len* **by** *auto*
have $(\mathcal{B}s^* ! j1) \cap (\mathcal{B}s^* ! j2) = \text{points-index } \mathcal{B} \{\mathcal{V}s ! j1, \mathcal{V}s ! j2\}$
using *dual-blocks-inter-index assms* **by** *simp*
thus *?thesis* **using** *ss c balanced*
by *blast*

qed

lemma *dual-is-const-intersect-des*:

assumes $\Lambda > 0$
shows *const-intersect-design* $\{0..<(\text{length } \mathcal{B}s)\}$ (*dual-blocks* \mathcal{V} $\mathcal{B}s$) Λ

proof –

have $(\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x \geq \Lambda)$ **using** *const-index-lt-rep* **by** *simp*

then have $(\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0)$ **using** *assms*
by (*metis gr-zeroI le-zero-eq*)
then interpret *pd: proper-design* $\{0..<(\text{length } \mathcal{B}s)\}$ $(\text{dual-blocks } \mathcal{V} \mathcal{B}s)$
using *dual-is-proper-design* **by** (*simp*)
show *?thesis* **proof** (*unfold-locales*)
fix *b1 b2*
assume *b1in: b1* $\in \#$ *dual-blocks* $\mathcal{V} \mathcal{B}s$
assume *b2in: b2* $\in \#$ *remove1-mset* *b1* $(\text{dual-blocks } \mathcal{V} \mathcal{B}s)$
obtain *j1* **where** *b1eq: b1* $= \mathcal{B}s^* ! j1$ **and** *j1lt: j1* $< \text{length } \mathcal{B}s^*$ **using** *b1in*
by (*metis dual-blocks-ordered-eq in-set-conv-nth set-mset-mset*)
obtain *j2* **where** *b2eq: b2* $= \mathcal{B}s^* ! j2$ **and** *j2lt: j2* $< \text{length } \mathcal{B}s^*$ **and** $j1 \neq j2$
using *b2in index-remove1-mset-ne*
by (*metis (mono-tags) b1eq dual-blocks-ordered-eq j1lt nth-mem set-mset-mset*)

then show $b1 \mid \cap \mid b2 = \Lambda$
using *dual-blocks-const-intersect b1eq b2eq j1lt j2lt* **by** *simp*
qed
qed

lemma *dual-is-simp-const-inter-des:*
assumes $\Lambda > 0$
assumes $\bigwedge bl. bl \in \# \mathcal{B} \implies \text{incomplete-block } bl$
shows *simple-const-intersect-design* $\{0..<(\text{length } \mathcal{B}s)\}$ $(\text{dual-blocks } \mathcal{V} \mathcal{B}s) \ \Lambda$
proof –
interpret *d: const-intersect-design* $\{0..<(\text{length } \mathcal{B}s)\}$ $(\text{dual-blocks } \mathcal{V} \mathcal{B}s) \ \Lambda$
using *assms dual-is-const-intersect-des* **by** *simp*
– Show that $m < \text{block size}$ for all blocks
have $\bigwedge x. x \in \mathcal{V} \implies \Lambda < \mathcal{B} \text{ rep } x$ **using** *assms incomplete-index-strict-lt-rep*
by *blast*
then have $\bigwedge bl. bl \in \# (\text{dual-blocks } \mathcal{V} \mathcal{B}s) \implies \Lambda < \text{card } bl$
by (*metis dual-blocks-size-is-rep-obtain*)
then interpret *s: simple-design* $\{0..<(\text{length } \mathcal{B}s)\}$ $(\text{dual-blocks } \mathcal{V} \mathcal{B}s)$
using *d.simple-const-inter-block-size* **by** *simp*
show *?thesis* **by** (*unfold-locales*)
qed
end

context *ordered-const-intersect-design*
begin

lemma *dual-is-balanced:*
assumes $ps \subseteq \{0..<\text{length } \mathcal{B}s\}$
assumes $\text{card } ps = 2$
shows $(\text{dual-blocks } \mathcal{V} \mathcal{B}s) \text{ index } ps = m$
proof –
obtain *i1 i2* **where** *psin: ps* $= \{i1, i2\}$ **and** *neq: i1* $\neq i2$ **using** *assms*
by (*meson card-2-iff*)
then have *lt: i1* $< b$ **using** *assms*

```

    by (metis atLeastLessThan-iff blocks-list-length insert-subset)
  have lt2: i2 < b using assms psin
    by (metis atLeastLessThan-iff blocks-list-length insert-subset)
  then have inter: (dual-blocks  $\vee$   $\mathcal{B}s$ ) index ps = ( $\mathcal{B}s ! i1$ )  $\cap$  ( $\mathcal{B}s ! i2$ ) using
dual-blocks-points-index-inter neq lt
    using dual-blocks-ordered-eq psin by presburger
  have inb1: ( $\mathcal{B}s ! i1$ )  $\in\#$   $\mathcal{B}$ 
    using lt by auto
  have inb2: ( $\mathcal{B}s ! i2$ )  $\in\#$  ( $\mathcal{B} - \{\#(\mathcal{B}s ! i1)\#$ ) using lt2 neq blocks-index-ne-belong
    by (metis blocks-list-length lt)
  thus ?thesis using const-intersect inb1 inb2 inter by blast
qed

```

lemma dual-is-pbd:

```

  assumes ( $\bigwedge x . x \in \mathcal{V} \implies \mathcal{B} \text{ rep } x > 0$ )
  assumes b  $\geq$  2
  shows pairwise-balance {0.. $\langle$ length  $\mathcal{B}s$ } (dual-blocks  $\vee$   $\mathcal{B}s$ ) m
proof -
  interpret pd: proper-design {0.. $\langle$ length  $\mathcal{B}s$ } (dual-blocks  $\vee$   $\mathcal{B}s$ )
    using dual-is-proper-design
    by (simp add: assms)
  show ?thesis proof (unfold-locales)
    show (1 :: nat)  $\leq$  2 by simp
    then show 2  $\leq$  dual-sys.v using assms(2)
      by fastforce
    show  $\bigwedge ps . ps \subseteq \{0.. $\langle$ length  $\mathcal{B}s$ \} \implies \text{card } ps = 2 \implies \text{dual-blocks } \vee \mathcal{B}s \text{ index}$ 
ps = m
      using dual-is-balanced by simp
    qed
  qed

```

end

context ordered-sym-bibd

begin

lemma dual-is-balanced:

```

  assumes ps  $\subseteq$  {0.. $\langle$ length  $\mathcal{B}s$ }
  assumes card ps = 2
  shows (dual-blocks  $\vee$   $\mathcal{B}s$ ) index ps =  $\Lambda$ 
proof -
  obtain i1 i2 where psin: ps = {i1, i2} and neq: i1  $\neq$  i2
    using assms by (meson card-2-iff)
  then have lt: i1 < b using assms
    by (metis atLeastLessThan-iff blocks-list-length insert-subset)
  have lt2: i2 < b using assms psin
    by (metis atLeastLessThan-iff blocks-list-length insert-subset)
  then have inter: (dual-blocks  $\vee$   $\mathcal{B}s$ ) index ps = ( $\mathcal{B}s ! i1$ )  $\cap$  ( $\mathcal{B}s ! i2$ )
    using dual-blocks-points-index-inter neq lt dual-blocks-ordered-eq psin by pres-

```

```

burger
  have inb1: ( $\mathcal{B}s ! i1$ )  $\in \# \mathcal{B}$ 
    using lt by auto
  have inb2: ( $\mathcal{B}s ! i2$ )  $\in \# (\mathcal{B} - \{\#(\mathcal{B}s ! i1)\# \})$  using lt2 neq blocks-index-simp-unique
    by (metis blocks-list-length in-remove1-mset-neq lt valid-blocks-index)
  thus ?thesis using sym-block-intersections-index inb1 inter by blast
qed

```

```

lemma dual-bibd: bibd  $\{0..<(\text{length } \mathcal{B}s)\}$  ( $\text{dual-blocks } \mathcal{V} \mathcal{B}s$ ) r  $\Lambda$ 
proof -
  interpret block: block-design  $\{0..<(\text{length } \mathcal{B}s)\}$  ( $\text{dual-blocks } \mathcal{V} \mathcal{B}s$ ) r
    using dual-is-block-design by simp
  show ?thesis proof (unfold-locales)
    show r < dual-sys.v
      using dual-blocks-v incomplete symmetric-condition-1 symmetric-condition-2
    by presburger
    show (1 :: nat)  $\leq 2$  by simp
    have v  $\geq 2$ 
      by (simp add: t-lt-order)
    then have b  $\geq 2$  using local.symmetric by auto
    then show 2  $\leq$  dual-sys.v by simp
    show  $\bigwedge ps. ps \subseteq \{0..<\text{length } \mathcal{B}s\} \implies \text{card } ps = 2 \implies \text{dual-blocks } \mathcal{V} \mathcal{B}s \text{ index } ps = \Lambda$ 
      using dual-is-balanced by simp
    show 2  $\leq$  r using r-ge-two by blast
  qed
qed

```

The dual of a BIBD must be symmetric

```

lemma dual-bibd-symmetric: symmetric-bibd  $\{0..<(\text{length } \mathcal{B}s)\}$  ( $\text{dual-blocks } \mathcal{V} \mathcal{B}s$ )
r  $\Lambda$ 
proof -
  interpret bibd: bibd  $\{0..<(\text{length } \mathcal{B}s)\}$  ( $\text{dual-blocks } \mathcal{V} \mathcal{B}s$ ) r  $\Lambda$ 
    using dual-bibd by simp
  show ?thesis using dual-blocks-b local.symmetric by (unfold-locales) (simp)
qed

end

```

5.5 Generalise Dual Concept

The above formalisation relies on one translation of a dual design. However, any design with an ordering of points and blocks such that the matrix is the transpose of the original is a dual. The definition below encapsulates this concept. Additionally, we prove an isomorphism exists between the generated dual from *dual-blocks* and any design satisfying the is dual definition

```

context ordered-incidence-system
begin

```

definition *is-dual*: 'b list \Rightarrow 'b set list \Rightarrow bool **where**
is-dual $Vs' Bs' \equiv$ ordered-incidence-system $Vs' Bs' \wedge$ (inc-mat-of $Vs' Bs' = N^T$)

lemma *is-dualI*:
assumes ordered-incidence-system $Vs' Bs'$
assumes (inc-mat-of $Vs' Bs' = N^T$)
shows *is-dual* $Vs' Bs'$
by (auto simp add: *is-dual-def* *assms*)

lemma *is-dualD1*:
assumes *is-dual* $Vs' Bs'$
shows (inc-mat-of $Vs' Bs' = N^T$)
using *is-dual-def* *assms*
by auto

lemma *is-dualD2*:
assumes *is-dual* $Vs' Bs'$
shows ordered-incidence-system $Vs' Bs'$
using *is-dual-def* *assms*
by auto

lemma *generated-is-dual*: *is-dual* $[0..<(\text{length } \mathcal{B}s)] \mathcal{B}s^*$
proof –
interpret *osys*: ordered-incidence-system $[0..<(\text{length } \mathcal{B}s)] \mathcal{B}s^*$ **using** *dual-is-ordered-inc-sys*
by *simp*
show ?thesis **using** *is-dual-def*
by (*simp* add: *is-dual-def* *dual-incidence-mat-eq-trans* *osys.ordered-incidence-system-axioms*)

qed

lemma *is-dual-isomorphism-generated*:
assumes *is-dual* $Vs' Bs'$
shows $\exists \pi$. incidence-system-isomorphism (set Vs') (mset Bs') ($\{0..<(\text{length } \mathcal{B}s)\}$) (dual-blocks $\mathcal{V} \mathcal{B}s$) π
proof –
interpret *os2*: ordered-incidence-system $([0..<(\text{length } \mathcal{B}s)]) (\mathcal{B}s^*)$
by (*simp* add: *dual-is-ordered-inc-sys*)
interpret *os1*: ordered-incidence-system $Vs' Bs'$ **using** *assms*
by (*simp* add: *is-dualD2*)
interpret *tos*: two-ordered-sys $Vs' Bs' ([0..<(\text{length } \mathcal{B}s)]) (\mathcal{B}s^*)$
using *assms* ordered-incidence-system-axioms two-ordered-sys.intro
by (*simp* add: *is-dualD2* two-ordered-sys.intro *dual-is-ordered-inc-sys*)
have *os2V*: *os2.V* = $\{0..<(\text{length } \mathcal{B}s)\}$
by auto
have *os2B*: *os2.B* = dual-blocks $\mathcal{V} \mathcal{B}s$
by (*simp* add: *dual-blocks-ordered-eq*)
have *os1.N* = inc-mat-of $Vs' Bs'$ **by** *simp*
then have *os2.N* = *os1.N*

using *assms is-dualD1 dual-incidence-mat-eq-trans* **by** *fastforce*
thus *?thesis using tos.equal-inc-mat-isomorphism-ex os2V os2B* **by** *auto*
qed

interpretation *ordered-dual-sys: ordered-incidence-system [0..<length Bs] Bs**
using *dual-is-ordered-inc-sys* **by** *simp*

Original system is dual of the dual

lemma *is-dual-rev: ordered-dual-sys.is-dual \mathcal{V} s Bs*

by (*simp add: dual-incidence-mat-eq-trans-rev ordered-dual-sys.is-dualI ordered-incidence-system-axioms*)

end

end

6 Rank Argument - General

General lemmas to enable reasoning using the rank argument. This is described by Godsil [5] and Bukh [2], both of whom present it as a foundational technique

theory *Rank-Argument-General* **imports** *Dual-Systems Jordan-Normal-Form.Determinant Jordan-Normal-Form.DL-Rank Jordan-Normal-Form.Ring-Hom-Matrix BenOr-Kozen-Reif.More-Matrix*
begin

6.1 Row/Column Operations

Extensions to the existing elementary operations are made to enable reasoning on multiple operations at once, similar to mathematical literature

lemma *index-mat-addrow-basic [simp]:*

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{addrow } a \ k \ l \ A \ \$\$ \ (i,j) = (\text{if } k = i \text{ then } (a * (A \ \$\$ \ (l,j)) + (A \ \$\$ \ (i,j))) \text{ else } A \ \$\$ \ (i,j))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{addrow } a \ i \ l \ A \ \$\$ \ (i,j) = (a * (A \ \$\$ \ (l,j)) + (A \ \$\$ \ (i,j)))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies k \neq i \implies \text{addrow } a \ k \ l \ A \ \$\$ \ (i,j) = A \ \$\$ \ (i,j)$

$\dim\text{-row } (\text{addrow } a \ k \ l \ A) = \dim\text{-row } A \ \dim\text{-col } (\text{addrow } a \ k \ l \ A) = \dim\text{-col } A$

unfolding *mat-addrow-def* **by** *auto*

Function to add a column to multiple other columns

fun *add-col-to-multiple* :: *'a :: semiring-1 \Rightarrow nat list \Rightarrow nat \Rightarrow 'a mat \Rightarrow 'a mat*
where

add-col-to-multiple a [] l A = A |

add-col-to-multiple a (k # ks) l A = (addcol a k l (add-col-to-multiple a ks l A))

Function to add a row to multiple other rows

fun *add-row-to-multiple* :: *'a :: semiring-1 \Rightarrow nat list \Rightarrow nat \Rightarrow 'a mat \Rightarrow 'a mat*
where

add-row-to-multiple $a \ [] \ l \ A = A \ |$
add-row-to-multiple $a \ (k \ \# \ ks) \ l \ A = (\text{addrow } a \ k \ l \ (\text{add-row-to-multiple } a \ ks \ l \ A))$

Function to add multiple rows to a single row

fun *add-multiple-rows* :: 'a :: semiring-1 \Rightarrow nat \Rightarrow nat list \Rightarrow 'a mat \Rightarrow 'a mat
where

add-multiple-rows $a \ k \ [] \ A = A \ |$
add-multiple-rows $a \ k \ (l \ \# \ ls) \ A = (\text{addrow } a \ k \ l \ (\text{add-multiple-rows } a \ k \ ls \ A))$

Function to add multiple columns to a single col

fun *add-multiple-cols* :: 'a :: semiring-1 \Rightarrow nat \Rightarrow nat list \Rightarrow 'a mat \Rightarrow 'a mat
where

add-multiple-cols $a \ k \ [] \ A = A \ |$
add-multiple-cols $a \ k \ (l \ \# \ ls) \ A = (\text{addcol } a \ k \ l \ (\text{add-multiple-cols } a \ k \ ls \ A))$

Basic lemmas on dimension and indexing of resulting matrix from above functions

lemma *add-multiple-rows-dim* [simp]:
 $\text{dim-row } (\text{add-multiple-rows } a \ k \ ls \ A) = \text{dim-row } A$
 $\text{dim-col } (\text{add-multiple-rows } a \ k \ ls \ A) = \text{dim-col } A$
by (induct ls) simp-all

lemma *add-multiple-rows-index-unchanged* [simp]:
 $i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow k \neq i \Longrightarrow \text{add-multiple-rows } a \ k \ ls \ A \ \$\$$
 $(i,j) = A \ \$\(i,j)
by (induct ls) (simp-all)

lemma *add-multiple-rows-index-eq*:
assumes $i < \text{dim-row } A$ **and** $j < \text{dim-col } A$ **and** $i \notin \text{set } ls$ **and** $\bigwedge l. l \in \text{set } ls \Longrightarrow l < \text{dim-row } A$
shows $\text{add-multiple-rows } a \ i \ ls \ A \ \$\$(i,j) = (\sum l \leftarrow ls. a * A \ \$\$(l,j)) + A \ \$\$(i,j)$
using *assms* **proof** (induct ls)
case Nil
then show ?case **by** simp
next
case (Cons aa ls)
then have $ne: i \neq aa$
by auto
have $lt: aa < \text{dim-row } A$ **using** *assms*(1)
by (simp add: Cons.prem(4))
have $(\text{add-multiple-rows } a \ i \ (aa \ \# \ ls) \ A) \ \$\$(i,j) =$
 $(\text{addrow } a \ i \ aa \ (\text{add-multiple-rows } a \ i \ ls \ A)) \ \$\$(i,j)$
by simp
also have $\dots = a * \text{add-multiple-rows } a \ i \ ls \ A \ \$\$(aa,j) + (\text{add-multiple-rows } a$
 $i \ ls \ A) \ \$\(i,j)
using *assms*(1) *assms*(2) *index-mat-addrow-basic*(2)[of i (add-multiple-rows a i ls A) j a aa]
by simp
also have $\dots = a * A \ \$\$(aa,j) + (\text{add-multiple-rows } a \ i \ ls \ A) \ \$\$(i,j)$

using $lt\ ne$ **by** (*simp add: assms(2)*)
also have $\dots = a * A\ \$\$(aa, j) + (\sum l \leftarrow ls. a * A\ \$\$(l, j)) + A\ \$\$(i, j)$
using *Cons.hyps assms(1) assms(2) Cons.prem(3) Cons.prem(4)*
by (*metis (mono-tags, lifting) ab-semigroup-add-class.add-ac(1) list.set-intros(2)*)

finally show (*add-multiple-rows a i (aa # ls) A*) $\$ \$ (i, j) =$
 $(\sum l \leftarrow (aa \# ls). a * A\ \$\$(l, j)) + A\ \$\$(i, j)$
by *simp*
qed

lemma *add-multiple-rows-index-eq-bounds*:
assumes $i < \dim\text{-row } A$ **and** $j < \dim\text{-col } A$ **and** $i < low \vee i \geq up$ **and** $up \leq \dim\text{-row } A$
shows *add-multiple-rows a i [low..<up] A* $\$ \$ (i, j) = (\sum l = low..<up. a * A\ \$\$(l, j)) + A\ \$\$(i, j)$
proof –
have *notin: i ∉ set [low..<up]* **using** *assms(3)* **by** *auto*
have $\bigwedge l. l \in \text{set } [low..<up] \implies l < \dim\text{-row } A$ **using** *assms(4)* **by** *auto*
thus *?thesis* **using** *add-multiple-rows-index-eq[of i A j [low..<up]]*
*sum-set-upt-eq-sum-list[of λ l. a * A \\$\$(l, j) low up] notin assms(1) assms(2)*
by *simp*
qed

lemma *add-multiple-cols-dim [simp]*:
 $\dim\text{-row } (\text{add-multiple-cols } a\ k\ ls\ A) = \dim\text{-row } A$
 $\dim\text{-col } (\text{add-multiple-cols } a\ k\ ls\ A) = \dim\text{-col } A$
by (*induct ls simp-all*)

lemma *add-multiple-cols-index-unchanged [simp]*:
 $i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies k \neq j \implies \text{add-multiple-cols } a\ k\ ls\ A\ \$ \$ (i, j) = A\ \$ \$ (i, j)$
by (*induct ls (simp-all)*)

lemma *add-multiple-cols-index-eq*:
assumes $i < \dim\text{-row } A$ **and** $j < \dim\text{-col } A$ **and** $j \notin \text{set } ls$ **and** $\bigwedge l. l \in \text{set } ls \implies l < \dim\text{-col } A$
shows *add-multiple-cols a j ls A* $\$ \$ (i, j) = (\sum l \leftarrow ls. a * A\ \$\$(i, l)) + A\ \$\$(i, j)$
using *assms*
proof (*induct ls*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons aa ls*)
then have $ne: j \neq aa$
by *auto*
have $lt: aa < \dim\text{-col } A$ **using** *assms*
by (*simp add: Cons.prem(4)*)
have (*add-multiple-cols a j (aa # ls) A*) $\$ \$ (i, j) = (\text{addcol } a\ j\ aa\ (\text{add-multiple-cols } a\ j\ ls\ A))\ \$ \$ (i, j)$

by simp
also have ... = $a * \text{add-multiple-cols } a \text{ } j \text{ } ls \text{ } A \text{ } \$\$ (i, aa) + (\text{add-multiple-cols } a \text{ } j \text{ } ls \text{ } A) \text{ } \$\$ (i, j)$
using *assms index-mat-addcol* **by simp**
also have ... = $a * A \text{ } \$\$ (i, aa) + (\text{add-multiple-cols } a \text{ } j \text{ } ls \text{ } A) \text{ } \$\$ (i, j)$
using *lt ne* **by** (*simp add: assms(1)*)
also have ... = $a * A \text{ } \$\$ (i, aa) + (\sum l \leftarrow ls. a * A \text{ } \$\$ (i, l)) + A \text{ } \$\$ (i, j)$
using *Cons.hyps assms(1) assms(2) Cons.prem(3) Cons.prem(4)*
by (*metis (mono-tags, lifting) ab-semigroup-add-class.add-ac(1) list.set-intros(2)*)

finally show ?*case* **by simp**
qed

lemma *add-multiple-cols-index-eq-bounds*:

assumes $i < \text{dim-row } A$ **and** $j < \text{dim-col } A$ **and** $j < \text{low} \vee j \geq \text{up}$ **and** $\text{up} \leq \text{dim-col } A$
shows $\text{add-multiple-cols } a \text{ } j \text{ } [\text{low}..<\text{up}] \text{ } A \text{ } \$\$ (i, j) = (\sum l = \text{low}..<\text{up}. a * A \text{ } \$\$ (i, l)) + A \text{ } \$\$ (i, j)$
proof –
have *notin*: $j \notin \text{set } [\text{low}..<\text{up}]$ **using** *assms(3)* **by auto**
have $\bigwedge l. l \in \text{set } [\text{low}..<\text{up}] \implies l < \text{dim-col } A$ **using** *assms(4)* **by auto**
thus ?*thesis* **using** *add-multiple-cols-index-eq*[*of i A j [low..<up] a*]
sum-set-upt-eq-sum-list[*of λ l. a * A \\$\$(i, l) low up*] *notin assms(1) assms(2)*
by simp
qed

lemma *add-row-to-multiple-dim* [*simp*]:

$\text{dim-row } (\text{add-row-to-multiple } a \text{ } ks \text{ } l \text{ } A) = \text{dim-row } A$
 $\text{dim-col } (\text{add-row-to-multiple } a \text{ } ks \text{ } l \text{ } A) = \text{dim-col } A$
by (*induct ks*) *simp-all*

lemma *add-row-to-multiple-index-unchanged* [*simp*]:

$i < \text{dim-row } A \implies j < \text{dim-col } A \implies i \notin \text{set } ks \implies \text{add-row-to-multiple } a \text{ } ks \text{ } l \text{ } A \text{ } \$\$ (i, j) = A \text{ } \$\$ (i, j)$
by (*induct ks*) *simp-all*

lemma *add-row-to-multiple-index-unchanged-bound*:

$i < \text{dim-row } A \implies j < \text{dim-col } A \implies i < \text{low} \implies i \geq \text{up} \implies$
 $\text{add-row-to-multiple } a \text{ } [\text{low}..<\text{up}] \text{ } l \text{ } A \text{ } \$\$ (i, j) = A \text{ } \$\$ (i, j)$
by simp

lemma *add-row-to-multiple-index-change*:

assumes $i < \text{dim-row } A$ **and** $j < \text{dim-col } A$ **and** $i \in \text{set } ks$ **and** *distinct ks* **and** $l \notin \text{set } ks$
and $l < \text{dim-row } A$
shows $\text{add-row-to-multiple } a \text{ } ks \text{ } l \text{ } A \text{ } \$\$ (i, j) = (a * A \text{ } \$\$ (l, j)) + A \text{ } \$\$ (i, j)$
using *assms*
proof (*induct ks*)
case Nil

```

then show ?case by simp
next
case (Cons aa ls)
then have lnotin:  $l \notin \text{set } ls$  using assms by simp
then show ?case
proof (cases  $i = aa$ )
  case True
  then have inotin:  $i \notin \text{set } ls$  using assms
  using Cons.prem(4) by fastforce
  have add-row-to-multiple a (aa # ls) l A  $\$ \$ (i, j) =$ 
    (addrow a aa l (add-row-to-multiple a ls l A))  $\$ \$ (i, j)$  by simp
  also have ... = (a * ((add-row-to-multiple a ls l A)  $\$ \$ (l, j)$ ) +
    ((add-row-to-multiple a ls l A)  $\$ \$ (i, j)$ ))
  using True assms(1) assms(2) by auto
  also have ... = a * A  $\$ \$ (l, j)$  + ((add-row-to-multiple a ls l A)  $\$ \$ (i, j)$ )
  using assms lnotin by simp
  finally have add-row-to-multiple a (aa # ls) l A  $\$ \$ (i, j) = a * A \ \$ \$ (l, j) + A$ 
 $\$ \$ (i, j)$ 
  using inotin assms by simp
  then show ?thesis by simp
next
case False
then have iin:  $i \in \text{set } ls$  using assms
  by (meson Cons.prem(3) set-ConsD)
  have add-row-to-multiple a (aa # ls) l A  $\$ \$ (i, j) =$  (addrow a aa l (add-row-to-multiple
a ls l A))  $\$ \$ (i, j)$ 
  by simp
  also have ... = ((add-row-to-multiple a ls l A)  $\$ \$ (i, j)$ )
  using False assms by auto
  finally have add-row-to-multiple a (aa # ls) l A  $\$ \$ (i, j) = a * A \ \$ \$ (l, j) +$ 
A  $\$ \$ (i, j)$ 
  using Cons.hyps by (metis Cons.prem(4) assms(1) assms(2) assms(6)
distinct.simps(2) iin lnotin)
  then show ?thesis by simp
qed
qed

```

lemma *add-row-to-multiple-index-change-bounds:*

assumes $i < \text{dim-row } A$ **and** $j < \text{dim-col } A$ **and** $i \geq \text{low}$ **and** $i < \text{up}$ **and** $l < \text{low} \vee l \geq \text{up}$

and $l < \text{dim-row } A$

shows $\text{add-row-to-multiple } a \text{ [low..<up] } l A \ \$ \$ (i, j) = (a * A \ \$ \$ (l, j)) + A \ \$ \$ (i, j)$

proof –

have $d: \text{distinct [low..<up]}$ **by** *simp*

have $iin: i \in \text{set [low..<up]}$ **using** *assms* **by** *auto*

have $lnin: l \notin \text{set [low..<up]}$ **using** *assms* **by** *auto*

thus *?thesis*

using *add-row-to-multiple-index-change d iin assms* **by** *blast*

qed

lemma *add-col-to-multiple-dim* [simp]:
 $\dim\text{-row } (\text{add-col-to-multiple } a \text{ } ks \text{ } l \text{ } A) = \dim\text{-row } A$
 $\dim\text{-col } (\text{add-col-to-multiple } a \text{ } ks \text{ } l \text{ } A) = \dim\text{-col } A$
by (induct ks) simp-all

lemma *add-col-to-multiple-index-unchanged* [simp]:
 $i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies j \notin \text{set } ks \implies \text{add-col-to-multiple } a \text{ } ks \text{ } l \text{ } A$
 $\$(i,j) = A \(i,j)
by (induct ks) simp-all

lemma *add-col-to-multiple-index-unchanged-bound*:
 $i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies j < \text{low} \implies j \geq \text{up} \implies$
 $\text{add-col-to-multiple } a \text{ } [\text{low}..\text{<up}] \text{ } l \text{ } A \$(i,j) = A \$(i,j)$
by simp

lemma *add-col-to-multiple-index-change*:
assumes $i < \dim\text{-row } A$ **and** $j < \dim\text{-col } A$ **and** $j \in \text{set } ks$ **and** *distinct ks* **and**
 $l \notin \text{set } ks$
and $l < \dim\text{-col } A$
shows $\text{add-col-to-multiple } a \text{ } ks \text{ } l \text{ } A \$(i,j) = (a * A \$(i, l)) + A \(i,j)
using *assms*
proof (induct ks)
case Nil
then show ?case **by** simp
next
case (Cons aa ls)
then have *lnotin*: $l \notin \text{set } ls$ **using** *assms* **by** simp
then show ?case
proof (cases $j = aa$)
case True
then have *notin*: $j \notin \text{set } ls$ **using** *assms*
using Cons.prem(4) **by** fastforce
have $\text{add-col-to-multiple } a \text{ } (aa \# ls) \text{ } l \text{ } A \$(i, j) =$
 $(\text{addcol } a \text{ } aa \text{ } l \text{ } (\text{add-col-to-multiple } a \text{ } ls \text{ } l \text{ } A)) \(i, j) **by** simp
also have $\dots = (a * ((\text{add-col-to-multiple } a \text{ } ls \text{ } l \text{ } A) \$(i,l)) +$
 $((\text{add-col-to-multiple } a \text{ } ls \text{ } l \text{ } A) \$(i,j)))$
using True *assms*(1) *assms*(2) **by** auto
also have $\dots = a * A \$(i, l) + ((\text{add-col-to-multiple } a \text{ } ls \text{ } l \text{ } A) \$(i,j))$
using *assms* *lnotin* **by** simp
finally have $\text{add-col-to-multiple } a \text{ } (aa \# ls) \text{ } l \text{ } A \$(i, j) = a * A \$(i,l) + A$
 $\$(i, j)$
using *notin* *assms* **by** simp
then show ?thesis **by** simp
next
case False
then have *iin*: $j \in \text{set } ls$ **using** *assms*
by (meson Cons.prem(3) set-ConsD)

have $\text{add-col-to-multiple } a \text{ (} aa \# ls \text{) } l A \text{ } \$\$ (i, j) =$
 $(\text{addcol } a \text{ } aa \text{ } l \text{ (} \text{add-col-to-multiple } a \text{ } ls \text{ } l A \text{) } \$\$ (i, j) \text{ by simp}$
also have $\dots = ((\text{add-col-to-multiple } a \text{ } ls \text{ } l A) \$\$ (i, j))$
using *False assms* **by auto**
finally have $\text{add-col-to-multiple } a \text{ (} aa \# ls \text{) } l A \text{ } \$\$ (i, j) = a * A \$\$ (i, l) +$
 $A \$\$ (i, j)$
using *Cons.hyps* **by** (*metis Cons.prem(4) assms(1) assms(2) assms(6)*
distinct.simps(2) iin lnotin)
then show *?thesis* **by simp**
qed
qed

lemma *add-col-to-multiple-index-change-bounds:*
assumes $i < \text{dim-row } A$ **and** $j < \text{dim-col } A$ **and** $j \geq \text{low}$ **and** $j < \text{up}$ **and** $l <$
 $\text{low} \vee l \geq \text{up}$
and $l < \text{dim-col } A$
shows $\text{add-col-to-multiple } a \text{ [low..<up] } l A \text{ } \$\$ (i, j) = (a * A \$\$ (i, l)) + A \$\$ (i, j)$
proof –
have $d: \text{distinct [low..<up]}$ **by simp**
have $jin: j \in \text{set [low..<up]}$ **using** *assms* **by auto**
have $l \notin \text{set [low..<up]}$ **using** *assms* **by auto**
thus *?thesis*
using *add-col-to-multiple-index-change d jin assms* **by blast**
qed

Operations specifically on 1st row/column

lemma *add-first-row-to-multiple-index:*
assumes $i < \text{dim-row } M$ **and** $j < \text{dim-col } M$
shows $i = 0 \implies (\text{add-row-to-multiple } a \text{ [1..<dim-row } M] \text{ } 0 M) \$\$ (i, j) = M \$\$$
 (i, j)
and $i \neq 0 \implies (\text{add-row-to-multiple } a \text{ [1..<dim-row } M] \text{ } 0 M) \$\$ (i, j) = (a *$
 $M \$\$ (0, j)) + M \$\$ (i, j)$
using *assms add-row-to-multiple-index-change-bounds[of i M j 1 dim-row M 0 a]*
by (*simp, linarith*)

lemma *add-all-cols-to-first:*
assumes $i < \text{dim-row } (M)$
assumes $j < \text{dim-col } (M)$
shows $j \neq 0 \implies \text{add-multiple-cols } 1 \text{ } 0 \text{ [1..<dim-col } M] M \$\$ (i, j) = M \$\$ (i,$
 $j)$
and $j = 0 \implies \text{add-multiple-cols } 1 \text{ } 0 \text{ [1..<dim-col } M] M \$\$ (i, j) = (\sum l =$
 $1..< \text{dim-col } M. M \$\$ (i, l)) + M \$\$ (i, 0)$
using *assms add-multiple-cols-index-eq-bounds[of i M j 1 dim-col M 1] assms* **by**
(simp-all)

Lemmas on the determinant of the matrix under extended row/column operations

lemma *add-row-to-multiple-carrier:*
 $A \in \text{carrier-mat } n \text{ } n \implies \text{add-row-to-multiple } a \text{ } ks \text{ } l A \in \text{carrier-mat } n \text{ } n$

by (*metis add-row-to-multiple-dim(1) add-row-to-multiple-dim(2) carrier-matD(1) carrier-matD(2) carrier-matI*)

lemma *add-col-to-multiple-carrier*:

$A \in \text{carrier-mat } n \ n \implies \text{add-col-to-multiple } a \ ks \ l \ A \in \text{carrier-mat } n \ n$

by (*metis add-col-to-multiple-dim carrier-matD(1) carrier-matD(2) carrier-matI*)

lemma *add-multiple-rows-carrier*:

$A \in \text{carrier-mat } n \ n \implies \text{add-multiple-rows } a \ k \ ls \ A \in \text{carrier-mat } n \ n$

by (*metis add-multiple-rows-dim carrier-matD(1) carrier-matD(2) carrier-matI*)

lemma *add-multiple-cols-carrier*:

$A \in \text{carrier-mat } n \ n \implies \text{add-multiple-cols } a \ k \ ls \ A \in \text{carrier-mat } n \ n$

by (*metis add-multiple-cols-dim carrier-matD(1) carrier-matD(2) carrier-matI*)

lemma *add-row-to-multiple-det*:

assumes $l \notin \text{set } ks$ **and** $l < n$ **and** $A \in \text{carrier-mat } n \ n$

shows $\det (\text{add-row-to-multiple } a \ ks \ l \ A) = \det A$

using *assms*

proof (*induct ks*)

case *Nil*

then show *?case by simp*

next

case (*Cons aa ks*)

have *ne: aa \neq l*

using *Cons.prem(1) by auto*

have $\det (\text{add-row-to-multiple } a \ (aa \ \# \ ks) \ l \ A) = \det (\text{addrow } a \ aa \ l \ (\text{add-row-to-multiple } a \ ks \ l \ A))$

by *simp*

also have $\dots = \det (\text{add-row-to-multiple } a \ ks \ l \ A)$

by (*meson det-addrow add-row-to-multiple-carrier ne assms*)

finally have $\det (\text{add-row-to-multiple } a \ (aa \ \# \ ks) \ l \ A) = \det A$

using *Cons.hyps assms by (metis Cons.prem(1) list.set-intros(2))*

then show *?case by simp*

qed

lemma *add-col-to-multiple-det*:

assumes $l \notin \text{set } ks$ **and** $l < n$ **and** $A \in \text{carrier-mat } n \ n$

shows $\det (\text{add-col-to-multiple } a \ ks \ l \ A) = \det A$

using *assms*

proof (*induct ks*)

case *Nil*

then show *?case by simp*

next

case (*Cons aa ks*)

have *ne: aa \neq l*

using *Cons.prem(1) by auto*

have $\det (\text{add-col-to-multiple } a \ (aa \ \# \ ks) \ l \ A) = \det (\text{addcol } a \ aa \ l \ (\text{add-col-to-multiple } a \ ks \ l \ A))$

by *simp*
 also have ... = det (add-col-to-multiple a ks l A)
 by (meson det-addcol add-col-to-multiple-carrier ne assms)
 finally have det (add-col-to-multiple a (aa # ks) l A) = det A
 using Cons.hyps assms by (metis Cons.prem(1) list.set-intros(2))
 then show ?case by *simp*
 qed

lemma *add-multiple-cols-det*:
 assumes $k \notin \text{set } ls$ and $\bigwedge l. l \in \text{set } ls \implies l < n$ and $A \in \text{carrier-mat } n \ n$
 shows det (add-multiple-cols a k ls A) = det A
 using *assms*
proof (*induct* ls)
 case Nil
 then show ?case by *simp*
next
 case (Cons aa ls)
 have ne: $aa \neq k$
 using Cons.prem(1) by *auto*
 have det (add-multiple-cols a k (aa # ls) A) = det (addcol a k aa (add-multiple-cols a k ls A))
 by *simp*
 also have ... = det (add-multiple-cols a k ls A)
 using det-addcol add-multiple-cols-carrier ne *assms* by (metis Cons.prem(2) list.set-intros(1))
 finally have det (add-multiple-cols a k (aa # ls) A) = det A
 using Cons.hyps *assms* by (metis Cons.prem(1) Cons.prem(2) list.set-intros(2))

 then show ?case by *simp*
 qed

lemma *add-multiple-rows-det*:
 assumes $k \notin \text{set } ls$ and $\bigwedge l. l \in \text{set } ls \implies l < n$ and $A \in \text{carrier-mat } n \ n$
 shows det (add-multiple-rows a k ls A) = det A
 using *assms*
proof (*induct* ls)
 case Nil
 then show ?case by *simp*
next
 case (Cons aa ls)
 have ne: $aa \neq k$
 using Cons.prem(1) by *auto*
 have det (add-multiple-rows a k (aa # ls) A) = det (addrow a k aa (add-multiple-rows a k ls A))
 by *simp*
 also have ... = det (add-multiple-rows a k ls A)
 using det-addrow add-multiple-rows-carrier ne *assms* by (metis Cons.prem(2) list.set-intros(1))
 finally have det (add-multiple-rows a k (aa # ls) A) = det A

```

using Cons.hyps assms by (metis Cons.prem1 Cons.prem2 list.set-intros2)

then show ?case by simp
qed

```

6.2 Rank and Linear Independence

abbreviation $\text{rank } v \ M \equiv \text{vec-space.rank } v \ M$

Basic lemma: the rank of the multiplication of two matrices will be less than the minimum of the individual ranks of those matrices. This directly follows from an existing lemmas in the linear algebra library which show independently that the resulting matrices rank is less than either the right or left matrix rank in the product

```

lemma rank-mat-mult-lt-min-rank-factor:
  fixes  $A :: 'a::\{\text{conjugatable-ordered-field}\} \text{ mat}$ 
  assumes  $A \in \text{carrier-mat } n \ m$ 
  assumes  $B \in \text{carrier-mat } m \ nc$ 
  shows  $\text{rank } n \ (A * B) \leq \min (\text{rank } n \ A) (\text{rank } m \ B)$ 
proof –
  have  $1: \text{rank } n \ (A * B) \leq (\text{rank } n \ A)$ 
    using assms(1) assms(2) vec-space.rank-mat-mul-right by blast
  have  $\text{rank } n \ (A * B) \leq \text{rank } m \ B$ 
    by (meson assms(1) assms(2) rank-mat-mul-left)
  thus ?thesis using  $1$  by simp
qed

```

Rank Argument 1: Given two a $x \times y$ matrix M where MM^T has rank x , $x \leq y$

```

lemma rank-argument:
  fixes  $M :: ('c :: \{\text{conjugatable-ordered-field}\}) \text{ mat}$ 
  assumes  $M \in \text{carrier-mat } x \ y$ 
  assumes  $\text{vec-space.rank } x \ (M * M^T) = x$ 
  shows  $x \leq y$ 
proof –
  let  $?B = (M * M^T)$ 
  have Mt-car:  $M^T \in \text{carrier-mat } y \ x$  using assms by simp
  have b-car:  $?B \in \text{carrier-mat } x \ x$ 
    using transpose-carrier-mat assms by simp
  then have  $\text{rank } x \ ?B \leq \min (\text{rank } x \ M) (\text{rank } y \ M^T)$ 
    using rank-mat-mult-lt-min-rank-factor Mt-car b-car assms(1) by blast
  thus ?thesis using le-trans vec-space.rank-le-nc
    by (metis assms(1) assms(2) min.bounded-iff)
qed

```

Generalise the rank argument to use the determinant. If the determinant of the matrix is non-zero, than it's rank must be equal to x . This removes the need for someone to use facts on rank in their proofs.

lemma *rank-argument-det*:

```

fixes  $M :: ('c :: \{\text{conjugatable-ordered-field}\}) \text{ mat}$ 
assumes  $M \in \text{carrier-mat } x \ y$ 
assumes  $\det (M * M^T) \neq 0$ 
shows  $x \leq y$ 
proof –
  let  $?B = (M * M^T)$ 
  have  $Mt\text{-car}: M^T \in \text{carrier-mat } y \ x$  using assms by simp
  have  $b\text{-car}: ?B \in \text{carrier-mat } x \ x$ 
    using transpose-carrier-mat assms by simp
  then have  $b\text{-rank}: \text{vec-space.rank } x \ ?B = x$ 
    using vec-space.low-rank-det-zero assms(2) by blast
  then have  $\text{rank } x \ ?B \leq \min (\text{rank } x \ M) (\text{rank } y \ M^T)$ 
    using rank-mat-mult-lt-min-rank-factor Mt-car b-car assms(1) by blast
  thus  $?thesis$  using le-trans vec-space.rank-le-nc
    by (metis assms(1) b-rank min.bounded-iff)
qed

end

```

7 Linear Bound Argument - General

Lemmas to enable general reasoning using the linear bound argument for combinatorial proofs. Jukna [7] presents a good overview of the mathematical background this theory is based on and applications

theory *Linear-Bound-Argument* **imports** *Incidence-Matrices Jordan-Normal-Form.DL-Rank*

Jordan-Normal-Form.Ring-Hom-Matrix

begin

7.1 Vec Space Extensions

Simple extensions to the existing vector space locale on linear independence

context *vec-space*

begin

lemma *lin-indpt-set-card-lt-dim*:

fixes $A :: 'a \text{ vec set}$

assumes $A \subseteq \text{carrier-vec } n$

assumes *lin-indpt* A

shows $\text{card } A \leq \text{dim}$

using *assms(1) assms(2) fin-dim li-le-dim(2)* **by** *blast*

lemma *lin-indpt-dim-col-lt-dim*:

fixes $A :: 'a \text{ mat}$

assumes $A \in \text{carrier-mat } n \ nc$

assumes *distinct* (*cols* A)

assumes *lin-indpt* (*set* (*cols* A))

shows $nc \leq \text{dim}$

proof –

```

have b: card (set (cols A)) = dim-col A using cols-length assms(2)
by (simp add: distinct-card)
have set (cols A)  $\subseteq$  carrier-vec n using assms(1) cols-dim by blast
thus ?thesis using lin-indpt-set-card-lt-dim assms b by auto
qed

```

lemma *lin-comb-imp-lin-dep-fin*:

```

fixes A :: 'a vec set
assumes finite A
assumes A  $\subseteq$  carrier-vec n
assumes lincomb c A = 0v n
assumes  $\exists$  a. a  $\in$  A  $\wedge$  c a  $\neq$  0
shows lin-dep A
unfolding lin-dep-def using assms lincomb-as-lincomb-list-distinct sumlist-nth
by auto

```

While a trivial definition, this enables us to directly reference the definition outside of a locale context, as *lin-indpt* is an inherited definition

definition *lin-indpt-vs*:: 'a vec set \Rightarrow bool **where**
lin-indpt-vs A \longleftrightarrow *lin-indpt* A

lemma *lin-comb-sum-lin-indpt*:

```

fixes A :: 'a vec list
assumes set (A)  $\subseteq$  carrier-vec n
assumes distinct A
assumes  $\bigwedge$  f. lincomb-list ( $\lambda$ i. f (A ! i)) A = 0v n  $\implies$   $\forall v \in$  (set A). f v = 0
shows lin-indpt (set A)
by (rule finite-lin-indpt2, auto simp add: assms lincomb-as-lincomb-list-distinct)

```

lemma *lin-comb-mat-lin-indpt*:

```

fixes A :: 'a vec list
assumes set (A)  $\subseteq$  carrier-vec n
assumes distinct A
assumes  $\bigwedge$  f. mat-of-cols n A *_v vec (length A) ( $\lambda$ i. f (A ! i)) = 0v n  $\implies$   $\forall v \in$ 
(set A). f v = 0
shows lin-indpt (set A)
proof (rule lin-comb-sum-lin-indpt, auto simp add: assms)
fix f v
have  $\bigwedge$  v. v  $\in$  set A  $\implies$  dim-vec v = n
using assms by auto
then show lincomb-list ( $\lambda$ i. f (A ! i)) A = 0v n  $\implies$  v  $\in$  set A  $\implies$  f v = 0
using lincomb-list-as-mat-mult[of A ( $\lambda$ i. f (A ! i))] assms(3)[of f] by auto
qed

```

lemma *lin-comb-mat-lin-indpt-vs*:

```

fixes A :: 'a vec list
assumes set (A)  $\subseteq$  carrier-vec n
assumes distinct A
assumes  $\bigwedge$  f. mat-of-cols n A *_v vec (length A) ( $\lambda$ i. f (A ! i)) = 0v n  $\implies$   $\forall v \in$ 

```

```

(set A). f v = 0
  shows lin-indpt-vs (set A)
  using lin-comb-mat-lin-indpt lin-indpt-vs-def assms by auto

```

end

7.2 Linear Bound Argument Lemmas

Three general representations of the linear bound argument, requiring a direct fact of linear independence on the rows of the vector space over either a set A of vectors, list xs of vectors or a Matrix' columns

```

lemma lin-bound-arg-general-set:
  fixes A :: ('a :: {field}) vec set
  assumes A  $\subseteq$  carrier-vec nr
  assumes vec-space.lin-indpt-vs nr A
  shows card A  $\leq$  nr
  using vec-space.lin-indpt-set-card-lt-dim[of A nr] vec-space.lin-indpt-vs-def[of nr A]
  vec-space.dim-is-n assms by metis

```

```

lemma lin-bound-arg-general-list:
  fixes xs :: ('a :: {field}) vec list
  assumes distinct xs
  assumes (set xs)  $\subseteq$  carrier-vec nr
  assumes vec-space.lin-indpt-vs nr (set xs)
  shows length xs  $\leq$  nr
  using lin-bound-arg-general-set[of set xs nr] distinct-card assms
  by force

```

```

lemma lin-bound-arg-general:
  fixes A :: ('a :: {field}) mat
  assumes distinct (cols A)
  assumes A  $\in$  carrier-mat nr nc
  assumes vec-space.lin-indpt-vs nr (set (cols A))
  shows nc  $\leq$  nr

```

proof –

```

  have ss: set (cols A)  $\subseteq$  carrier-vec nr using assms cols-dim by blast
  have length (cols A) = nc
    using assms(2) cols-length by blast
  thus ?thesis using lin-bound-arg-general-list[of cols A nr] ss assms by simp

```

qed

The linear bound argument lemma on a matrix requiring the lower level assumption on a linear combination. This removes the need to directly refer to any aspect of the linear algebra libraries

```

lemma lin-bound-argument:
  fixes A :: ('a :: {field}) mat

```

```

assumes distinct (cols A)
assumes  $A \in \text{carrier-mat } nr \ nc$ 
assumes  $\bigwedge f. A *_v \text{vec } nc (\lambda i. f (\text{col } A \ i)) = 0_v \ nr \implies \forall v \in (\text{set } (\text{cols } A)). f \ v = 0$ 
shows  $nc \leq nr$ 
proof (intro lin-bound-arg-general[of A nr nc] vec-space.lin-comb-mat-lin-indpt-vs, simp-all add: assms)
show  $\text{set } (\text{cols } A) \subseteq \text{carrier-vec } nr$  using assms cols-dim by blast
next
have  $mA: \text{mat-of-cols } nr (\text{cols } A) = A$  using mat-of-cols-def assms by auto
have  $\bigwedge f. \text{vec } (\text{dim-col } A) (\lambda i. f (\text{cols } A \ ! \ i)) = \text{vec } nc (\lambda i. f (\text{col } A \ i))$ 
proof (intro eq-vecI, simp-all add: assms)
show  $\bigwedge f \ i. i < nc \implies \text{vec } (\text{dim-col } A) (\lambda i. f (\text{cols } A \ ! \ i)) \$ i = f (\text{col } A \ i)$ 
using assms(2) by force
show  $\text{dim-col } A = nc$  using assms by simp
qed
then show  $\bigwedge f. \text{mat-of-cols } nr (\text{cols } A) *_v \text{vec } (\text{dim-col } A) (\lambda i. f (\text{cols } A \ ! \ i)) = 0_v \ nr \implies \forall v \in \text{set } (\text{cols } A). f \ v = 0$ 
using mA assms(3) by metis
qed

```

A further extension to present the linear combination directly as a sum. This manipulation from vector product to a summation was found to commonly be repeated in proofs applying this rule

```

lemma lin-bound-argument2:
fixes  $A :: ('a :: \{\text{field}\}) \text{mat}$ 
assumes distinct (cols A)
assumes  $A \in \text{carrier-mat } nr \ nc$ 
assumes  $\bigwedge f. \text{vec } nr (\lambda i. \sum j \in \{0..<nc\} . f (\text{col } A \ j) * (\text{col } A \ j) \$ i) = 0_v \ nr \implies \forall v \in (\text{set } (\text{cols } A)). f \ v = 0$ 
shows  $nc \leq nr$ 
proof (intro lin-bound-argument[of A nr nc], simp add: assms, simp add: assms)
fix  $f$ 
have  $A *_v \text{vec } nc (\lambda i. f (\text{col } A \ i)) = \text{vec } (\text{dim-row } A) (\lambda i. \sum j \in \{0..<nc\} . (\text{row } A \ i \ \$ \ j) * f (\text{col } A \ j))$ 
by (auto simp add: mult-mat-vec-def scalar-prod-def assms(2))
also have  $\dots = \text{vec } (\text{dim-row } A) (\lambda i. \sum j \in \{0..<nc\} . f (\text{col } A \ j) * (\text{col } A \ j \ \$ \ i))$ 
using assms(2) by (intro eq-vecI, simp-all) (meson mult.commute)
finally show  $A *_v \text{vec } nc (\lambda i. f (\text{col } A \ i)) = 0_v \ nr \implies \forall v \in \text{set } (\text{cols } A). f \ v = 0$ 
using assms(3)[of f] assms(2) by fastforce
qed
end

```

8 Fisher's Inequality

This theory presents the proof of Fisher's Inequality [4] on BIBD's (i.e. uniform Fisher's) and the generalised nonuniform Fisher's Inequality

theory *Fishers-Inequality* **imports** *Rank-Argument-General-Linear-Bound-Argument*
begin

8.1 Uniform Fisher's Inequality

context *ordered-bibd*
begin

Row/Column transformation steps

Following design theory lecture notes from MATH3301 at the University of Queensland [6], a simple transformation to produce an upper triangular matrix using elementary row operations is to (1) Subtract the first row from each other row, and (2) add all columns to the first column

lemma *transform-N-step1-vals*:

defines *mdef*: $M \equiv (N * N^T)$

assumes $i < \text{dim-row } M$

assumes $j < \text{dim-col } M$

shows $i = 0 \implies j = 0 \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M] 0 M) \text{ $$}$
 $(i, j) = (\text{int } r) - \text{top left elem}$

and $i \neq 0 \implies j = 0 \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M] 0 M) \text{ $$}$
 $(i, j) = (\text{int } \Lambda) - (\text{int } r) - \text{first column ex. 1}$

and $i = 0 \implies j \neq 0 \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M] 0 M) \text{ $$}$
 $(i, j) = (\text{int } \Lambda) - \text{first row ex. 1}$

and $i \neq 0 \implies j \neq 0 \implies i = j \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M]$
 $0 M) \text{ $$ } (i, j) = (\text{int } r) - (\text{int } \Lambda) - \text{diagonal ex. 1}$

and $i \neq 0 \implies j \neq 0 \implies i \neq j \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M]$
 $0 M) \text{ $$ } (i, j) = 0 - \text{everything else}$

using *transpose-N-mult-diag v-non-zero assms*

proof (*simp*)

show $i = 0 \implies j \neq 0 \implies (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } M] 0 M) \text{ $$}$
 $(i, j) = (\text{int } \Lambda)$

using *transpose-N-mult-off-diag v-non-zero assms transpose-N-mult-dim(2)* **by**
force

next

assume $a: j = 0 \ i \neq 0$

then have *ail*: $((-1) * M \text{ $$}(0, j)) = -(\text{int } r)$

using *transpose-N-mult-diag v-non-zero mdef* **by** *auto*

then have *ijne*: $j \neq i$ **using** a **by** *simp*

then have *aij*: $M \text{ $$ } (i, j) = (\text{int } \Lambda)$ **using** *assms(2) mdef transpose-N-mult-off-diag*
a v-non-zero

by (*metis transpose-N-mult-dim(1)*)

then have *add-row-to-multiple* $(-1) [1..<\text{dim-row } M] 0 M \text{ $$ } (i, j) = (-1)*(\text{int } r) + (\text{int } \Lambda)$

using *ail add-first-row-to-multiple-index(2) assms(2) assms(3) a* **by** (*metis mult-minus1*)
then show (*add-row-to-multiple (-1) [1..<dim-row M] 0 M*) \$\$ (i, j) = (int Λ)
- (int r)
by *linarith*
next
assume *a: i \neq 0 j \neq 0*
have *ail: ((-1) * M [0, j]) = -(int Λ)*
using *assms transpose-N-mult-off-diag a v-non-zero transpose-N-mult-dim(1)*
by *auto*
then have *i = j \implies M [i, j] = (int r)*
using *assms transpose-N-mult-diag a v-non-zero* **by** (*metis transpose-N-mult-dim(1)*)

then show *i = j \implies (add-row-to-multiple (-1) [1..<dim-row M] 0 M) [i, j]*
= (int r) - (int Λ)
using *ail add-first-row-to-multiple-index(2) assms a* **by** (*metis uminus-add-conv-diff*)

then have *i \neq j \implies M [i, j] = (int Λ)* **using** *assms transpose-N-mult-off-diag*
a v-non-zero
by (*metis transpose-N-mult-dim(1) transpose-N-mult-dim(2)*)
then show *i \neq j \implies (add-row-to-multiple (-1) [1..<dim-row M] 0 M) [i, j]*
= 0
using *ail add-first-row-to-multiple-index(2) assms a* **by** (*metis add.commute*
add.right-inverse)
qed

lemma *transform-N-step2-vals:*

defines *mdef: M \equiv (add-row-to-multiple (-1) [1..<dim-row (N * N^T)] 0 (N * N^T))*
assumes *i < dim-row (M)*
assumes *j < dim-col (M)*
shows *i = 0 \implies j = 0 \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] =*
*(int r) + (int Λ) * (v - 1) - top left element*
and *i = 0 \implies j \neq 0 \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] =*
(int Λ) - top row
and *i \neq 0 \implies i = j \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] =*
(int r) - (int Λ) - Diagonal
and *i \neq 0 \implies i \neq j \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] = 0*
- Everything else
proof -
show *i = 0 \implies j \neq 0 \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] =*
(int Λ)
using *add-all-cols-to-first assms transform-N-step1-vals(3)* **by** *simp*
show *i \neq 0 \implies i = j \implies add-multiple-cols 1 0 [1..<dim-col M] M [i, j] =*
(int r) - (int Λ)
using *add-all-cols-to-first assms transform-N-step1-vals(4)* **by** *simp*
next
assume *a: i = 0 j = 0*
then have *size: card {1..<dim-col M} = v - 1* **using** *assms* **by** *simp*


```

have val:  $\bigwedge l . l \in \{1..<dim-col\ M\} \implies M \ \$\$ (i, l) = (int\ \Lambda)$ 
using mdef transform-N-step1-vals(3) by (simp add: a(1))
have add-multiple-cols 1 0 [1..<dim-col M] M  $\ \$\$ (i, j) = (\sum l \in \{1..<dim-col\ M\}. M \ \$\$ (i, l)) + M \ \$\$ (i, 0)$ 
using a assms add-all-cols-to-first by blast
also have ... =  $(\sum l \in \{1..<dim-col\ M\}. (int\ \Lambda)) + M \ \$\$ (i, 0)$  using val by simp
also have ... =  $(v - 1) * (int\ \Lambda) + M \ \$\$ (i, 0)$  using size by (metis sum-constant)

finally show add-multiple-cols 1 0 [1..<dim-col M] M  $\ \$\$ (i, j) = (int\ r) + (int\ \Lambda) * (v - 1)$ 
using transform-N-step1-vals(1) a(1) a(2) assms(1) assms(2) by simp
next
assume a:  $i \neq 0 \ i \neq j$ 
then show add-multiple-cols 1 0 [1..<dim-col M] M  $\ \$\$ (i, j) = 0$ 
proof (cases  $j \neq 0$ )
  case True
    then show ?thesis using add-all-cols-to-first assms a transform-N-step1-vals(5)
by simp
  next
    case False
      then have iin:  $i \in \{1..<dim-col\ M\}$  using a(1) assms by simp
      have cond:  $\bigwedge l . l \in \{1..<dim-col\ M\} \implies l < dim-col\ (N * N^T) \wedge l \neq 0$  using
      assms by simp
      then have val:  $\bigwedge l . l \in \{1..<dim-col\ M\} - \{i\} \implies M \ \$\$ (i, l) = 0$ 
        using assms(3) transform-N-step1-vals(5) a False assms(1)
      by (metis DiffE iin index-mult-mat(2) index-mult-mat(3) index-transpose-mat(3)
      insertI1)
      have val2:  $M \ \$\$ (i, i) = (int\ r) - (int\ \Lambda)$  using mdef transform-N-step1-vals(4)
      a False
        assms(1) transpose-N-mult-dim(1) transpose-N-mult-dim(2)
      by (metis cond iin)
      have val3:  $M \ \$\$ (i, 0) = (int\ \Lambda) - (int\ r)$ 
        using assms(3) transform-N-step1-vals(2) a False assms(1) assms(2)
      by (metis add-row-to-multiple-dim(1) transpose-N-mult-dim(2) v-non-zero)
      have add-multiple-cols 1 0 [1..<dim-col M] M  $\ \$\$ (i, j) = (\sum l \in \{1..<dim-col\ M\}. M \ \$\$ (i, l)) + M \ \$\$ (i, 0)$ 
        using assms add-all-cols-to-first False by blast
      also have ... =  $M \ \$\$ (i, i) + (\sum l \in \{1..<dim-col\ M\} - \{i\}. M \ \$\$ (i, l)) + M \ \$\$ (i, 0)$ 
        by (metis iin finite-atLeastLessThan sum.remove)
      finally show ?thesis using val val2 val3 by simp
qed
qed

```

Transformed matrix is upper triangular

lemma transform-upper-triangular:

```

defines mdef:  $M \equiv (add-row-to-multiple\ (-1)\ [1..<dim-row\ (N * N^T)]\ 0\ (N * N^T))$ 
shows upper-triangular (add-multiple-cols 1 0 [1..<dim-col M] M)

```

```

using transform-N-step2-vals(4) by (intro upper-triangularI) (simp add: assms)

  Find the determinant of the  $NN^T$  matrix using transformed matrix values

lemma determinant-inc-mat-square:  $\det(N * N^T) = (r + \Lambda * (v - 1)) * (r - \Lambda)^{\wedge(v - 1)}$ 
proof -
  — Show the matrix is now lower triangular, therefore the det is the product of the sum of diagonal
    have cm:  $(N * N^T) \in \text{carrier-mat } v \ v$ 
      using transpose-N-mult-dim(1) transpose-N-mult-dim(2) by blast
      define C where  $C \equiv (\text{add-row-to-multiple } (-1) [1..<\text{dim-row } (N * N^T)]) \ 0 \ (N * N^T)$ 
      have  $0 \notin \text{set } [1..<\text{dim-row } (N * N^T)]$  by simp
      then have detbc:  $\det(N * N^T) = \det C$ 
        using C-def add-row-to-multiple-det v-non-zero by (metis cm)
        define D where  $D \equiv \text{add-multiple-cols } 1 \ 0 \ [1..<\text{dim-col } C] \ C$ 
        have d00:  $D \ \$\$ (0, 0) = ((\text{int } r) + (\text{int } \Lambda) * (v - 1))$  using transform-N-step2-vals(1)
          by (simp add: C-def D-def v-non-zero)
        have ine0:  $\bigwedge i. i \in \{1..<\text{dim-row } D\} \implies i \neq 0$  by simp
        have  $\bigwedge i. i \in \{1..<\text{dim-row } D\} \implies i < \text{dim-row } (N * N^T)$  using D-def C-def
by simp
        then have diagnon0:  $\bigwedge i. i \in \{1..<v\} \implies D \ \$\$ (i, i) = (\text{int } r) - (\text{int } \Lambda)$ 
          using transform-N-step2-vals(3) ine0 D-def C-def by simp
        have alll:  $\bigwedge l. l \in \text{set } [1..<\text{dim-col } C] \implies l < v$  using C-def by simp
        have cmc:  $C \in \text{carrier-mat } v \ v$  using cm C-def
          by (simp add: add-row-to-multiple-carrier)
        have dimgt2:  $\text{dim-row } D \geq 2$ 
          using t-lt-order D-def C-def by (simp)
        then have fstterm:  $0 \in \{0 ..<\text{dim-row } D\}$  by (simp add: points-list-length)
        have  $0 \notin \text{set } [1..<\text{dim-col } C]$  by simp
        then have  $\det(N * N^T) = \det D$  using add-multiple-cols-det alll cmc D-def
C-def
          by (metis detbc)
        also have ... =  $\text{prod-list } (\text{diag-mat } D)$  using det-upper-triangular
          using transform-upper-triangular D-def C-def by fastforce
        also have ... =  $(\prod i = 0 ..<\text{dim-row } D. D \ \$\$ (i, i))$  using prod-list-diag-prod
by blast
        also have ... =  $(\prod i = 0 ..<v. D \ \$\$ (i, i))$  by (simp add: D-def C-def)
        finally have  $\det(N * N^T) = D \ \$\$ (0, 0) * (\prod i = 1 ..<v. D \ \$\$ (i, i))$ 
          using dimgt2 by (simp add: prod.atLeast-Suc-lessThan v-non-zero)
        then have  $\det(N * N^T) = (r + \Lambda * (v - 1)) * ((\text{int } r) - (\text{int } \Lambda))^{\wedge(v - 1)}$ 
          using d00 diagnon0 by simp
        then have  $\det(N * N^T) = (r + \Lambda * (v - 1)) * (r - \Lambda)^{\wedge(v - 1)}$ 
          using index-lt-replication
          by (metis (no-types, lifting) less-imp-le-nat of-nat-diff of-nat-mult of-nat-power)
        then show ?thesis by blast
qed

```

Fisher's Inequality using the rank argument. Note that to use the rank

argument we must first map N to a real matrix. It is useful to explicitly include the parameters which should be used in the application of the *rank-argument-det* lemma

theorem *Fishers-Inequality-BIBD*: $v \leq b$

proof (*intro rank-argument-det[of map-mat real-of-int N v b], simp-all*)

show $N \in \text{carrier-mat } v \text{ (length } \mathcal{B}s)$ **using** *blocks-list-length N-carrier-mat* **by** *simp*

let $?B = \text{map-mat (real-of-int) } (N * N^T)$

have *b-split*: $?B = \text{map-mat (real-of-int) } N * (\text{map-mat (real-of-int) } N)^T$

using *semiring-hom.mat-hom-mult of-int-hom.semiring-hom-axioms transpose-carrier-mat map-mat-transpose*

by (*metis (no-types, lifting) N-carrier-mat*)

have *db*: $\det ?B = (r + \Lambda * (v - 1)) * (r - \Lambda) \wedge^{(v - 1)}$

using *determinant-inc-mat-square* **by** *simp*

have *lhn0*: $(r + \Lambda * (v - 1)) > 0$

using *r-gzero* **by** *blast*

have $(r - \Lambda) > 0$

using *index-lt-replication zero-less-diff* **by** *blast*

then have *det-not-0*: $\det ?B \neq 0$ **using** *lhn0 db*

by (*metis gr-implies-not0 mult-is-0 of-nat-eq-0-iff power-not-zero*)

thus $\det (\text{of-int-hom.mat-hom } N * (\text{of-int-hom.mat-hom } N)^T) \neq (0::\text{real})$ **using** *b-split* **by** *simp*

qed

end

8.2 Generalised Fisher's Inequality

context *simp-ordered-const-intersect-design*

begin

Lemma to reason on sum coefficients

lemma *sum-split-coeffs-0*:

fixes $c :: \text{nat} \Rightarrow \text{real}$

assumes $b \geq 2$

assumes $m > 0$

assumes $j' < b$

assumes $0 = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * ((\text{card } (\mathcal{B}s ! j)) - (\text{int } m))) + m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2})$

shows $c j' = 0$

proof (*rule ccontr*)

assume *cine0*: $c j' \neq 0$

have *innerge*: $\bigwedge j . j < b \implies (c j)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j) - (\text{int } m)) \geq 0$

using *inter-num-le-block-size assms(1)* **by** *simp*

then have *lhsge*: $(\sum j \in \{0..<b\} . (c j)^{\wedge 2} * ((\text{card } (\mathcal{B}s ! j)) - (\text{int } m))) \geq 0$

using *sum-bounded-below[of {0..<b} 0 λ i. (c i)² * ((card (Bs ! i)) - (int m))]* **by** *simp*

have $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2}) \geq 0$ **by** *simp*

then have *rhs0*: $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2}) = 0$

```

    using assms(2) assms(4) lhsge by linarith
  then have  $(\sum j \in \{0..<b\} . (c j)^2 * ((card (\mathcal{B}s ! j)) - (int m))) = 0$ 
    using assms by linarith
  then have lhs0inner:  $\bigwedge j . j < b \implies (c j)^2 * (card (\mathcal{B}s ! j) - (int m)) = 0$ 
    using innerge sum-nonneg-eq-0-iff[of  $\{0..<b\} \lambda j . (c j)^2 * (card (\mathcal{B}s ! j) - (int m))$ ]
    by simp
  thus False proof (cases  $card (\mathcal{B}s ! j') = m$ )
    case True
      then have cj0:  $\bigwedge j . j \in \{0..<b\} - \{j'\} \implies (c j) = 0$ 
        using lhs0inner const-intersect-block-size-diff assms True by auto
      then have  $(\sum i \in \{0..<b\} . c i) \neq 0$ 
        using sum.remove[of  $\{0..<b\} j' \lambda i . c i$ ] assms(3) cine0 cj0 by simp
      then show ?thesis using rhs0 assms by simp
    next
      case False
        then have ne:  $(card (\mathcal{B}s ! j') - (int m)) \neq 0$ 
          by linarith
        then have  $(c j')^2 * (card (\mathcal{B}s ! j') - (int m)) \neq 0$  using cine0
          by auto
        then show ?thesis using lhs0inner assms(3) by auto
    qed
  qed

```

The general non-uniform version of fisher's inequality is also known as the "Block town problem". In this case we are working in a constant intersect design, hence the inequality is the opposite way around compared to the BIBD version. The theorem below is the more traditional set theoretic approach. This proof is based off one by Jukna [7]

```

theorem general-fishers-inequality:  $b \leq v$ 
proof (cases  $m = 0 \vee b = 1$ )
  case True
    then show ?thesis using empty-inter-implies-b-lt-v v-non-zero by linarith
  next
    case False
      then have mge:  $m > 0$  by simp
      then have bge:  $b \geq 2$  using b-positive False blocks-list-length by linarith
      define NR :: real mat where  $NR \equiv lift-01-mat N$ 
      show ?thesis
      proof (intro lin-bound-argument2[of NR])
        show distinct (cols NR) using lift-01-distinct-cols-N NR-def by simp
        show nrcm:  $NR \in carrier-mat v b$  using NR-def N-carrier-mat-01-lift by simp

        have scalar-prod-real1:  $\bigwedge i . i < b \implies ((col NR i) \cdot (col NR i)) = card (\mathcal{B}s ! i)$ 
          using scalar-prod-block-size-lift-01 NR-def by auto
        have scalar-prod-real2:  $\bigwedge i j . i < b \implies j < b \implies i \neq j \implies ((col NR i) \cdot (col NR j)) = m$ 
          using scalar-prod-inter-num-lift-01 NR-def indexed-const-intersect by auto
        show  $\bigwedge f . vec v (\lambda i . \sum j = 0..<b . f (col NR j) * (col NR j) \$ i) = 0_v v \implies$ 

```

$\forall v \in \text{set } (\text{cols } NR). f v = 0$
proof (*intro ball1*)
fix $f v$
assume $eq0$: $\text{vec } v (\lambda i. \sum j = 0..<b. f (\text{col } NR j) * \text{col } NR j \$ i) = 0_v v$
assume vin : $v \in \text{set } (\text{cols } NR)$
define c **where** $c \equiv (\lambda j. f (\text{col } NR j))$
obtain j' **where** $v\text{-def}$: $\text{col } NR j' = v$ **and** $jvlt$: $j' < \text{dim-col } NR$
using vin **by** (*metis cols-length cols-nth index-less-size-conv nth-index*)
have dim-col : $\bigwedge j. j \in \{0..<b\} \implies \text{dim-vec } (\text{col } NR j) = v$ **using** *nrcm* **by**
auto
— Summation reasoning to obtain conclusion on coefficients
have $0 = (\text{vec } v (\lambda i. \sum j = 0..<b. c j * (\text{col } NR j) \$ i)) \cdot (\text{vec } v (\lambda i. \sum j = 0..<b. c j * (\text{col } NR j) \$ i))$
using *vec-prod-zero eq0 c-def* **by** *simp*
also have $\dots = (\sum j1 \in \{0..<b\} . c j1 * c j1 * ((\text{col } NR j1) \cdot (\text{col } NR j1)))$
 $+ (\sum j1 \in \{0..<b\} .$
 $(\sum j2 \in (\{0..<b\} - \{j1\}) . c j1 * c j2 * ((\text{col } NR j1) \cdot (\text{col } NR j2))))$
using *scalar-prod-double-sum-fn-vec[of b col NR v c] dim-col* **by** *simp*
also have $\dots = (\sum j1 \in \{0..<b\} . (c j1) * (c j1) * (\text{card } (\mathcal{B}s ! j1))) + (\sum$
 $j1 \in \{0..<b\} .$
 $(\sum j2 \in (\{0..<b\} - \{j1\}) . c j1 * c j2 * ((\text{col } NR j1) \cdot (\text{col } NR j2))))$
using *scalar-prod-real1* **by** *simp*
also have $\dots = (\sum j1 \in \{0..<b\} . (c j1)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j1))) + (\sum j1 \in$
 $\{0..<b\} .$
 $(\sum j2 \in (\{0..<b\} - \{j1\}) . c j1 * c j2 * ((\text{col } NR j1) \cdot (\text{col } NR j2))))$
by (*metis power2-eq-square*)
also have $\dots = (\sum j1 \in \{0..<b\} . (c j1)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j1))) + (\sum j1 \in$
 $\{0..<b\} .$
 $(\sum j2 \in (\{0..<b\} - \{j1\}) . c j1 * c j2 * m))$ **using** *scalar-prod-real2* **by**
auto
also have $\dots = (\sum j1 \in \{0..<b\} . (c j1)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j1))) +$
 $m * (\sum j1 \in \{0..<b\} . (\sum j2 \in (\{0..<b\} - \{j1\}) . c j1 * c j2))$
using *double-sum-mult-hom[of m λ i j . c i * c j λ i. {0..<b} - {i} {0..<b}]*
by (*metis (no-types, lifting) mult-of-nat-commute sum.cong*)
also have $\dots = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j))) +$
 $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2} - (\sum j \in \{0..<b\} . c j * c j))$
using *double-sum-split-square-diff* **by** *auto*
also have $\dots = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j))) + (-m) * (\sum j$
 $\in \{0..<b\} . (c j)^{\wedge 2}) +$
 $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2})$ **by** (*simp add: algebra-simps power2-eq-square*)
also have $\dots = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j))) + (\sum j \in \{0..<b\}$
 $. (-m) * (c j)^{\wedge 2}) +$
 $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2})$ **by** (*simp add: sum-distrib-left*)
also have $\dots = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * (\text{card } (\mathcal{B}s ! j))) + (-m) * (c j)^{\wedge 2}$
 $+$
 $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2})$ **by** (*metis (no-types) sum.distrib*)
finally have *sum-rep*: $0 = (\sum j \in \{0..<b\} . (c j)^{\wedge 2} * ((\text{card } (\mathcal{B}s ! j)) - (\text{int}$
 $m))) +$
 $m * ((\sum j \in \{0..<b\} . c j)^{\wedge 2})$ **by** (*simp add: algebra-simps*)

```

    thus  $f v = 0$  using sum-split-coeffs-0[of  $j'$   $c$ ] mge bge jvlt nrcm c-def v-def
  by simp
    qed
  qed
  qed
end

```

Using the dual design concept, it is easy to translate the set theoretic general definition of Fisher's inequality to a more traditional design theoretic version on pairwise balanced designs. Two versions of this are given using different trivial (but crucial) conditions on design properties

```

context ordered-pairwise-balance
begin

```

```

corollary general-nonuniform-fishers: — only valid on incomplete designs

```

```

  assumes  $\Lambda > 0$ 

```

```

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

```

```

    — i.e. not a super trivial design with only complete blocks

```

```

  shows  $v \leq b$ 

```

```

proof —

```

```

  have  $mset (\mathcal{B}s^*) = \text{dual-blocks } \mathcal{V} \mathcal{B}s$  using dual-blocks-ordered-eq by simp

```

```

  then interpret des: simple-const-intersect-design set  $[0..<(\text{length } \mathcal{B}s)]$   $mset (\mathcal{B}s^*)$ 

```

```

 $\Lambda$ 

```

```

    using assms dual-is-simp-const-inter-des by simp

```

```

  interpret odes: simp-ordered-const-intersect-design  $[0..<\text{length } \mathcal{B}s]$   $\mathcal{B}s^* \Lambda$ 

```

```

    using distinct-upt des.wellformed by (unfold-locales) (blast)

```

```

  have  $\text{length } (\mathcal{B}s^*) \leq \text{length } [0..<\text{length } \mathcal{B}s]$  using odes.general-fishers-inequality

```

```

    using odes.blocks-list-length odes.points-list-length by presburger

```

```

  then have  $v \leq \text{length } \mathcal{B}s$ 

```

```

    by (simp add: dual-blocks-len points-list-length)

```

```

  then show ?thesis by auto

```

```

qed

```

```

corollary general-nonuniform-fishers-comp:

```

```

  assumes  $\Lambda > 0$ 

```

```

  assumes  $\text{count } \mathcal{B} \mathcal{V} < \Lambda$  — i.e. not a super trivial design with only complete blocks and single blocks

```

```

  shows  $v \leq b$ 

```

```

proof —

```

```

  define  $B$  where  $B = (\text{removeAll-mset } \mathcal{V} \mathcal{B})$ 

```

```

  have b-smaller: size  $B \leq b$  using B-def removeAll-size-lt by simp

```

```

  then have b-incomp:  $\bigwedge bl. bl \in \# B \implies \text{card } bl < v$ 

```

```

    using wellformed B-def by (simp add: psubsetI psubset-card-mono)

```

```

  have index-gt:  $(\Lambda - (\text{count } \mathcal{B} \mathcal{V})) > 0$  using assms by simp

```

```

  interpret pbd: pairwise-balance  $\mathcal{V} B$   $(\Lambda - (\text{count } \mathcal{B} \mathcal{V}))$ 

```

```

    using remove-all-complete-blocks-pbd B-def assms(2) by blast

```

```

  obtain  $Bs$  where  $m: mset Bs = B$ 

```

```

    using ex-mset by blast

```

```

interpret opbd: ordered-pairwise-balance  $\mathcal{V}$ s  $B$ s ( $\Lambda - (\text{count } \mathcal{B} \mathcal{V})$ )
  by (intro pb.ordered-pbdI) (simp-all add: m distinct)
have  $v \leq (\text{size } B)$  using b-incomp opbd.general-nonuniform-fishers
  using index-gt m by blast
then show ?thesis using b-smaller m by auto
qed

end
end

```

9 Matrices/Vectors mod x

This section formalises operations and properties mod some integer x on integer matrices and vectors. Much of this file was no longer needed for fishers once the generic idea of lifting a 0-1 matrix was introduced, however it is left as an example for future work on matrices mod n, beyond 0-1 matrices

```

theory Vector-Matrix-Mod imports Matrix-Vector-Extras
Berlekamp-Zassenhaus.Finite-Field Berlekamp-Zassenhaus.More-Missing-Multiset
begin

```

Simple abbreviations for main mapping functions

```

abbreviation to-int-mat :: 'a :: finite mod-ring mat  $\Rightarrow$  int mat where
  to-int-mat  $\equiv$  map-mat to-int-mod-ring

```

```

abbreviation to-int-vec :: 'a :: finite mod-ring vec  $\Rightarrow$  int vec where
  to-int-vec  $\equiv$  map-vec to-int-mod-ring

```

```

interpretation of-int-mod-ring-hom-sr: semiring-hom of-int-mod-ring

```

```

proof (unfold-locales)

```

```

  show  $\bigwedge x y. \text{of-int-mod-ring } (x + y) = \text{of-int-mod-ring } x + \text{of-int-mod-ring } y$ 
    by (transfer, presburger)

```

```

  show of-int-mod-ring 1 = 1 by (metis of-int-hom.hom-one of-int-of-int-mod-ring)

```

```

  show  $\bigwedge x y. \text{of-int-mod-ring } (x * y) = \text{of-int-mod-ring } x * \text{of-int-mod-ring } y$ 
    by (transfer, simp add: mod-mult-eq)

```

```

qed

```

NOTE: The context directly below is copied from Matrix Vector Extras, as for some reason they can't be used locally if not? Ideally remove in future if possible

```

context inj-zero-hom
begin

```

```

lemma vec-hom-zero-iff[simp]: (map-vec hom  $x = 0_v$   $n$ ) = ( $x = 0_v$   $n$ )

```

```

proof -

```

```

  {
    fix i

```

```

assume  $i < n$   $\dim\text{-vec } x = n$ 
hence  $\text{map-vec hom } x \$ i = 0 \iff x \$ i = 0$ 
using  $\text{index-map-vec}(1)[\text{of } i \ x]$  by  $\text{simp}$ 
} note  $\text{main} = \text{this}$ 
show  $\text{?thesis unfolding vec-eq-iff}$  by  $(\text{simp}, \text{insert main}, \text{auto})$ 
qed

```

```

lemma  $\text{mat-hom-inj}$ :  $\text{map-mat hom } A = \text{map-mat hom } B \implies A = B$ 
unfolding  $\text{mat-eq-iff}$  by  $\text{auto}$ 

```

```

lemma  $\text{vec-hom-inj}$ :  $\text{map-vec hom } v = \text{map-vec hom } w \implies v = w$ 
unfolding  $\text{vec-eq-iff}$  by  $\text{auto}$ 

```

```

lemma  $\text{vec-hom-set-distinct-iff}$ :
fixes  $xs :: 'a \text{ vec list}$ 
shows  $\text{distinct } xs \iff \text{distinct } (\text{map } (\text{map-vec hom}) \ xs)$ 
using  $\text{vec-hom-inj}$  by  $(\text{induct } xs) (\text{auto})$ 
end

```

9.1 Basic Mod Context

```

locale  $\text{mat-mod} = \text{fixes } m :: \text{int}$ 
assumes  $\text{non-triv-}m$ :  $m > 1$ 
begin

```

First define the mod operations on vectors

```

definition  $\text{vec-mod} :: \text{int vec} \Rightarrow \text{int vec}$  where
 $\text{vec-mod } v \equiv \text{map-vec } (\lambda x . x \text{ mod } m) \ v$ 

```

```

lemma  $\text{vec-mod-dim}[\text{simp}]$ :  $\text{dim-vec } (\text{vec-mod } v) = \text{dim-vec } v$ 
using  $\text{vec-mod-def}$  by  $\text{simp}$ 

```

```

lemma  $\text{vec-mod-index}[\text{simp}]$ :  $i < \text{dim-vec } v \implies (\text{vec-mod } v) \$ i = (v \$ i) \text{ mod } m$ 
using  $\text{vec-mod-def}$  by  $\text{simp}$ 

```

```

lemma  $\text{vec-mod-index-same}[\text{simp}]$ :  $i < \text{dim-vec } v \implies v \$ i < m \implies v \$ i \geq 0$ 
 $\implies (\text{vec-mod } v) \$ i = v \$ i$ 
by  $\text{simp}$ 

```

```

lemma  $\text{vec-setI2}$ :  $i < \text{dim-vec } v \implies v \$ i \in \text{set}_v \ v$ 
by  $(\text{simp add: vec-setI})$ 

```

```

lemma  $\text{vec-mod-eq}$ :  $\text{set}_v \ v \subseteq \{0..<m\} \implies \text{vec-mod } v = v$ 
apply  $(\text{intro eq-vecI}, \text{simp-all})$ 
using  $\text{vec-setI2 vec-mod-index-same}$  by  $(\text{metis atLeastLessThan-iff subset-iff}$ 
 $\text{zmod-trivial-iff})$ 

```

```

lemma  $\text{vec-mod-set-vec-same}$ :  $\text{set}_v \ v \subseteq \{0..<m\} \implies \text{set}_v \ (\text{vec-mod } v) = \text{set}_v \ v$ 

```


using *vec-mod-eq* **by** *auto*

Define the mod operation on matrices

definition *mat-mod* :: *int mat* \Rightarrow *int mat* **where**
mat-mod *M* \equiv *map-mat* ($\lambda x. x \bmod m$) *M*

lemma *mat-mod-dim*[*simp*]: *dim-row* (*mat-mod* *M*) = *dim-row* *M* *dim-col* (*mat-mod* *M*) = *dim-col* *M*
using *mat-mod-def* **by** *simp-all*

lemma *mat-mod-index* [*simp*]: $i < \text{dim-row } M \Rightarrow j < \text{dim-col } M \Rightarrow (\text{mat-mod } M) \text{ \$(\$) } (i, j) = (M \text{ \$(\$) } (i, j)) \bmod m$
by(*simp add: mat-mod-def*)

lemma *mat-mod-index-same*[*simp*]: $i < \text{dim-row } M \Rightarrow j < \text{dim-col } M \Rightarrow M \text{ \$(\$) } (i, j) < m \Rightarrow$
 $M \text{ \$(\$) } (i, j) \geq 0 \Rightarrow \text{mat-mod } M \text{ \$(\$) } (i, j) = M \text{ \$(\$) } (i, j)$
by (*simp add: mat-mod-def*)

lemma *elements-matI2*: $i < \text{dim-row } A \Rightarrow j < \text{dim-col } A \Rightarrow A \text{ \$(\$) } (i, j) \in \text{elements-mat } A$
by *auto*

lemma *mat-mod-elements-in*:
assumes $x \in \text{elements-mat } M$
shows $x \bmod m \in \text{elements-mat } (\text{mat-mod } M)$
proof –
obtain $i j$ **where** $M \text{ \$(\$) } (i, j) = x$ **and** *ilt*: $i < \text{dim-row } M$ **and** *jlt*: $j < \text{dim-col } M$ **using** *assms* **by** *auto*
then have $\text{mat-mod } M \text{ \$(\$) } (i, j) = x \bmod m$ **by** *simp*
thus *?thesis* **using** *ilt jlt*
by (*metis elements-matI2 mat-mod-dim(1) mat-mod-dim(2)*)
qed

lemma *mat-mod-elements-map*:
assumes $x \in \text{elements-mat } M$
shows $\text{elements-mat } (\text{mat-mod } M) = (\lambda x. x \bmod m) \text{ ` } (\text{elements-mat } M)$
proof (*auto simp add: mat-mod-elements-in*)
fix x **assume** $x \in \text{elements-mat } (\text{local.mat-mod } M)$
then obtain $i j$ **where** $(\text{mat-mod } M) \text{ \$(\$) } (i, j) = x$ **and** $i < \text{dim-row } (\text{mat-mod } M)$ **and** $j < \text{dim-col } (\text{mat-mod } M)$ **by** *auto*
then show $x \in (\lambda x. x \bmod m) \text{ ` } \text{elements-mat } M$
by *auto*
qed

lemma *mat-mod-eq-cond*:
assumes $\text{elements-mat } M \subseteq \{0..<m\}$
shows $\text{mat-mod } M = M$
proof (*intro eq-matI, simp-all*)

fix $i\ j$ **assume** $i < \dim\text{-row } M\ j < \dim\text{-col } M$
then have $M \text{ \$\$ } (i, j) \in \{0..<m\}$
using *assms elements-matI2* **by** *blast*
then show $M \text{ \$\$ } (i, j) \bmod m = M \text{ \$\$ } (i, j)$
by (*simp*)
qed

lemma *mat-mod-eq-elements-cond*: $\text{elements-mat } M \subseteq \{0..<m\} \implies \text{elements-mat } (\text{mat-mod } M) = \text{elements-mat } M$
using *mat-mod-eq-cond* **by** *auto*

lemma *mat-mod-vec-mod-row*: $i < \dim\text{-row } A \implies \text{row } (\text{mat-mod } A)\ i = \text{vec-mod } (\text{row } A\ i)$
unfolding *mat-mod-def vec-mod-def* **by** (*simp*)

lemma *mat-mod-vec-mod-col*: $j < \dim\text{-col } A \implies \text{col } (\text{mat-mod } A)\ j = \text{vec-mod } (\text{col } A\ j)$
unfolding *mat-mod-def vec-mod-def* **by** (*simp*)

lemma *count-vec-mod-eq*: $\text{set}_v\ v \subseteq \{0..<m\} \implies \text{count-vec } v\ x = \text{count-vec } (\text{vec-mod } v)\ x$
using *vec-mod-eq* **by** (*simp*)

lemma *elems-mat-setv-row-0m*: $i < \dim\text{-row } M \implies \text{elements-mat } M \subseteq \{0..<m\} \implies \text{set}_v\ (\text{row } M\ i) \subseteq \{0..<m\}$
by (*metis row-elems-subset-mat subset-trans*)

lemma *elems-mat-setv-col-0m*: $j < \dim\text{-col } M \implies \text{elements-mat } M \subseteq \{0..<m\} \implies \text{set}_v\ (\text{col } M\ j) \subseteq \{0..<m\}$
by (*metis col-elems-subset-mat subset-trans*)

lemma *mat-mod-count-row-eq*: $i < \dim\text{-row } M \implies \text{elements-mat } M \subseteq \{0..<m\} \implies \text{count-vec } (\text{row } (\text{mat-mod } M)\ i)\ x = \text{count-vec } (\text{row } M\ i)\ x$
using *count-vec-mod-eq mat-mod-vec-mod-row elems-mat-setv-row-0m* **by** *simp*

lemma *mat-mod-count-col-eq*: $j < \dim\text{-col } M \implies \text{elements-mat } M \subseteq \{0..<m\} \implies \text{count-vec } (\text{col } (\text{mat-mod } M)\ j)\ x = \text{count-vec } (\text{col } M\ j)\ x$
using *count-vec-mod-eq mat-mod-vec-mod-col elems-mat-setv-col-0m* **by** *simp*

lemma *mod-mat-one*: $\text{mat-mod } (1_m\ n) = (1_m\ n)$
by (*intro eq-matI, simp-all add: mat-mod-def non-triv-m*)

lemma *mod-mat-zero*: $\text{mat-mod } (0_m\ nr\ nc) = (0_m\ nr\ nc)$
by (*intro eq-matI, simp-all add: mat-mod-def non-triv-m*)

lemma *vec-mod-unit*: $\text{vec-mod } (\text{unit-vec } n\ i) = (\text{unit-vec } n\ i)$
by (*intro eq-vecI, simp-all add: unit-vec-def vec-mod-def non-triv-m*)

lemma *vec-mod-zero*: $vec-mod (0_v n) = (0_v n)$
by (*intro eq-vecI, simp-all add: non-triv-m*)

lemma *mat-mod-cond-iff*: $elements-mat M \subseteq \{0..<m\} \implies P M \longleftrightarrow P (mat-mod M)$
by (*simp add: mat-mod-eq-cond*)

end

9.2 Mod Type

The below locale takes lemmas from the Poly Mod Finite Field theory in the Berlekamp Zassenhaus AFP entry, however has removed any excess material on polynomials mod, and only included the general factors. Ideally, this could be used as the base locale for both in the future

locale *mod-type* =
fixes $m :: int$ **and** $ty :: 'a :: nontriv\ itself$
assumes $m: m = CARD('a)$
begin

lemma *m1*: $m > 1$ **using** *nontriv*[**where** $'a = 'a$] **by** (*auto simp:m*)

definition *M* :: $int \Rightarrow int$ **where** $M x = x mod m$

lemma *M-0*[*simp*]: $M 0 = 0$
by (*auto simp add: M-def*)

lemma *M-M*[*simp*]: $M (M x) = M x$
by (*auto simp add: M-def*)

lemma *M-plus*[*simp*]: $M (M x + y) = M (x + y)$ $M (x + M y) = M (x + y)$
by (*auto simp add: M-def mod-simps*)

lemma *M-minus*[*simp*]: $M (M x - y) = M (x - y)$ $M (x - M y) = M (x - y)$
by (*auto simp add: M-def mod-simps*)

lemma *M-times*[*simp*]: $M (M x * y) = M (x * y)$ $M (x * M y) = M (x * y)$
by (*auto simp add: M-def mod-simps*)

lemma *M-1*[*simp*]: $M 1 = 1$ **unfolding** *M-def*
using *m1* **by** *auto*

lemma *M-sum*: $M (sum (\lambda x. M (f x)) A) = M (sum f A)$

proof (*induct A rule: infinite-finite-induct*)

case (*insert x A*)

from *insert(1-2)* **have** $M (\sum x \in insert\ x\ A. M (f x)) = M (f x + M ((\sum x \in A. M (f x))))$ **by** *simp*

also have $M ((\sum x \in A. M (f x))) = M ((\sum x \in A. f x))$ **using** *insert* **by** *simp*

finally show *?case using insert by simp*
qed *auto*

definition *inv-M* :: *int* \Rightarrow *int* **where**
inv-M *x* = (*if* *x* + *x* \leq *m* *then* *x* *else* *x* - *m*)

lemma *M-inv-M-id*[*simp*]: *M* (*inv-M* *x*) = *M* *x*
unfolding *inv-M-def* *M-def* **by** *simp*

definition *M-Rel* :: *int* \Rightarrow '*a mod-ring* \Rightarrow *bool*
where *M-Rel* *x* *x'* \equiv (*M* *x* = *to-int-mod-ring* *x'*)

lemma *to-int-mod-ring-plus*: *to-int-mod-ring* ((*x* :: '*a mod-ring*) + *y*) = *M* (*to-int-mod-ring* *x* + *to-int-mod-ring* *y*)
unfolding *M-def* **using** *m* **by** (*transfer*, *auto*)

lemma *to-int-mod-ring-times*: *to-int-mod-ring* ((*x* :: '*a mod-ring*) * *y*) = *M* (*to-int-mod-ring* *x* * *to-int-mod-ring* *y*)
unfolding *M-def* **using** *m* **by** (*transfer*, *auto*)

lemma *eq-M-Rel*[*transfer-rule*]: (*M-Rel* \implies *M-Rel* \implies (=)) (λ *x* *y*. *M* *x* = *M* *y*) (=)
unfolding *M-Rel-def* *rel-fun-def* **by** *auto*

lemma *one-M-Rel*[*transfer-rule*]: *M-Rel* 1 1
unfolding *M-Rel-def* *M-def*
unfolding *m* **by** *auto*

lemma *zero-M-Rel*[*transfer-rule*]: *M-Rel* 0 0
unfolding *M-Rel-def* *M-def*
unfolding *m* **by** *auto*

lemma *M-to-int-mod-ring*: *M* (*to-int-mod-ring* (*x* :: '*a mod-ring*)) = *to-int-mod-ring* *x*
unfolding *M-def* **unfolding** *m* **by** (*transfer*, *auto*)

lemma *right-total-M-Rel*[*transfer-rule*]: *right-total* *M-Rel*
unfolding *right-total-def* *M-Rel-def* **using** *M-to-int-mod-ring* **by** *blast*

lemma *left-total-M-Rel*[*transfer-rule*]: *left-total* *M-Rel*
unfolding *left-total-def* *M-Rel-def*[*abs-def*]

proof

fix *x*

show \exists *x'* :: '*a mod-ring*. *M* *x* = *to-int-mod-ring* *x'* **unfolding** *M-def* **unfolding** *m*

by (*rule* *exI*[*of - of-int* *x*], *transfer*, *simp*)

qed

lemma *bi-total-M-Rel*[*transfer-rule*]: *bi-total M-Rel*
using *right-total-M-Rel left-total-M-Rel* **by** (*metis bi-totalI*)

lemma *to-int-mod-ring-of-int-M*: *to-int-mod-ring (of-int x :: 'a mod-ring) = M x*
unfolding *M-def*
unfolding *m* **by** *transfer auto*

lemma *UNIV-M-Rel*[*transfer-rule*]: *rel-set M-Rel {0..*m*} UNIV*
unfolding *rel-set-def M-Rel-def[abs-def] M-def*
by (*auto simp: M-def m, goal-cases, metis to-int-mod-ring-of-int-mod-ring, (transfer, auto)+*)

end

9.3 Mat mod type

Define a context to work on matrices and vectors of type *'a mod-ring*

locale *mat-mod-type = mat-mod + mod-type*
begin

lemma *to-int-mod-ring-plus*: *to-int-mod-ring ((x :: 'a mod-ring) + y) = (to-int-mod-ring x + to-int-mod-ring y) mod m*
using *m* **by** (*transfer, auto*)

lemma *to-int-mod-ring-times*: *to-int-mod-ring ((x :: 'a mod-ring) * y) = (to-int-mod-ring x * to-int-mod-ring y) mod m*
using *m* **by** (*transfer, auto*)

Set up transfer relation for matrices and vectors

definition *MM-Rel* :: *int mat ⇒ 'a mod-ring mat ⇒ bool*
where *MM-Rel f f' ≡ (mat-mod f = to-int-mat f')*

definition *MV-Rel* :: *int vec ⇒ 'a mod-ring vec ⇒ bool*
where *MV-Rel v v' ≡ (vec-mod v = to-int-vec v')*

lemma *to-int-mat-index[simp]*: *i < dim-row N ⇒ j < dim-col N ⇒ (to-int-mat N \$\$ (i, j)) = to-int-mod-ring (N \$\$ (i, j))*
by *simp*

lemma *to-int-vec-index[simp]*: *i < dim-vec v ⇒ (to-int-vec v \$i) = to-int-mod-ring (v \$i)*
by *simp*

lemma *eq-dim-row-MM-Rel*[*transfer-rule*]: (*MM-Rel ==> (=)*) *dim-row dim-row*
by (*metis (mono-tags) MM-Rel-def index-map-mat(2) mat-mod-dim(1) rel-funI*)

lemma *lt-dim-row-MM-Rel*[*transfer-rule*]: (*MM-Rel ==> (=) ==> (=)*) (λM *i. i < dim-row M*) (λM *i. i < dim-row M*)

using *eq-dim-row-MM-Rel* **unfolding** *MM-Rel-def rel-fun-def* **by** *auto*

lemma *eq-dim-col-MM-Rel[transfer-rule]*: (*MM-Rel* \implies (=)) *dim-col dim-col*
unfolding *MM-Rel-def rel-fun-def*
by (*metis index-map-mat(3) mat-mod-dim(2)*)

lemma *lt-dim-col-MM-Rel[transfer-rule]*: (*MM-Rel* \implies (=) \implies (=)) ($\lambda M j. j < \text{dim-col } M$) ($\lambda M j. j < \text{dim-col } M$)
using *eq-dim-col-MM-Rel* **unfolding** *MM-Rel-def rel-fun-def* **by** *auto*

lemma *eq-dim-vec-MV-Rel[transfer-rule]*: (*MV-Rel* \implies (=)) *dim-vec dim-vec*
unfolding *MV-Rel-def rel-fun-def* **using** *index-map-vec(2) vec-mod-dim* **by** *metis*

lemma *lt-dim-vec-MV-Rel[transfer-rule]*: (*MV-Rel* \implies (=) \implies (=)) ($\lambda v j. j < \text{dim-vec } v$) ($\lambda v j. j < \text{dim-vec } v$)
unfolding *MV-Rel-def rel-fun-def* **using** *index-map-vec(2) vec-mod-dim* **by** *metis*

lemma *eq-MM-Rel[transfer-rule]*: (*MM-Rel* \implies *MM-Rel* \implies (=)) ($\lambda f f'. \text{mat-mod } f = \text{mat-mod } f'$) (=)
unfolding *MM-Rel-def rel-fun-def* **using** *to-int-mod-ring-hom.mat-hom-inj* **by** *(auto)*

lemma *eq-MV-Rel[transfer-rule]*: (*MV-Rel* \implies *MV-Rel* \implies (=)) ($\lambda v v'. \text{vec-mod } v = \text{vec-mod } v'$) (=)
unfolding *MV-Rel-def rel-fun-def* **using** *to-int-mod-ring-hom.vec-hom-inj* **by** *auto*

lemma *index-MV-Rel[transfer-rule]*: (*MV-Rel* \implies (=) \implies *M-Rel*)
($\lambda v i. \text{if } i < \text{dim-vec } v \text{ then } v \$ i \text{ else } 0$) ($\lambda v i. \text{if } i < \text{dim-vec } v \text{ then } v \$ i \text{ else } 0$)
using *lt-dim-vec-MV-Rel* **unfolding** *MV-Rel-def M-Rel-def M-def rel-fun-def*
by (*simp, metis to-int-vec-index vec-mod-index*)

lemma *index-MM-Rel[transfer-rule]*: (*MM-Rel* \implies (=) \implies (=) \implies *M-Rel*)
($\lambda M i j. \text{if } (i < \text{dim-row } M \wedge j < \text{dim-col } M) \text{ then } M \$\$ (i, j) \text{ else } 0$)
($\lambda M i j. \text{if } (i < \text{dim-row } M \wedge j < \text{dim-col } M) \text{ then } M \$\$ (i, j) \text{ else } 0$)
using *lt-dim-row-MM-Rel lt-dim-col-MM-Rel* **unfolding** *M-Rel-def M-def rel-fun-def*
by (*simp, metis mat-mod-index to-int-mat-index MM-Rel-def*)

lemma *index-MM-Rel-explicit*:
assumes *MM-Rel* *N N'*
assumes $i < \text{dim-row } N$ $j < \text{dim-col } N$
shows ($N \$\$ (i, j) \bmod m = \text{to-int-mod-ring } (N' \$\$ (i, j))$)
proof –
have *eq*: ($\text{to-int-mat } N'$) $\$ \$ (i, j) = \text{to-int-mod-ring } (N' \$\$ (i, j))$
by (*metis MM-Rel-def assms(1) assms(2) assms(3) index-map-mat mat-mod.mat-mod-dim*)

```

mat-mod-axioms)
have mat-mod  $N = \text{to-int-mat } N'$  using assms by (simp add: MM-Rel-def)
then have (mat-mod  $N$ )  $\$\$ (i, j) = (\text{to-int-mat } N') \$\$ (i, j)$ 
  by simp
thus ?thesis using mat-mod-index eq
  using assms(2) assms(3) by auto
qed

lemma one-MV-Rel[transfer-rule]: MV-Rel (unit-vec  $n$   $i$ ) (unit-vec  $n$   $i$ )
  unfolding MV-Rel-def vec-mod-unit non-triv-m unit-vec-def
  by (intro eq-vecI, simp-all add: non-triv-m)

lemma one-MM-Rel[transfer-rule]: MM-Rel ( $1_m$   $n$ ) ( $1_m$   $n$ )
  unfolding MM-Rel-def mod-mat-one
  by (intro eq-matI, simp-all)

lemma zero-MM-Rel[transfer-rule]: MM-Rel ( $0_m$   $nr$   $nc$ ) ( $0_m$   $nr$   $nc$ )
  unfolding MM-Rel-def
  by (intro eq-matI, simp-all)

lemma zero-MV-Rel[transfer-rule]: MV-Rel ( $0_v$   $n$ ) ( $0_v$   $n$ )
  unfolding MV-Rel-def by (intro eq-vecI, simp-all)

lemma right-unique-MV-Rel[transfer-rule]: right-unique MV-Rel
  unfolding right-unique-def MV-Rel-def
  using to-int-mod-ring-hom.vec-hom-inj by auto

lemma right-unique-MM-Rel[transfer-rule]: right-unique MM-Rel
  unfolding right-unique-def MM-Rel-def
  using to-int-mod-ring-hom.mat-hom-inj by auto

lemma mod-to-int-mod-ring: (to-int-mod-ring ( $x :: 'a$  mod-ring)) mod  $m = \text{to-int-mod-ring}$ 
 $x$ 
  unfolding  $m$  by (transfer, auto)

lemma mat-mod-to-int-mat: mat-mod (to-int-mat ( $N :: 'a$  mod-ring mat)) =
to-int-mat  $N$ 
  using mod-to-int-mod-ring by (intro eq-matI, simp-all)

lemma vec-mod-to-int-vec: vec-mod (to-int-vec ( $v :: 'a$  mod-ring vec)) = to-int-vec
 $v$ 
  using mod-to-int-mod-ring by (intro eq-vecI, simp-all)

lemma right-total-MM-Rel[transfer-rule]: right-total MM-Rel
  unfolding right-total-def MM-Rel-def
proof
  fix  $M :: 'a$  mod-ring mat
  show  $\exists x. \text{mat-mod } x = \text{to-int-mat } M$ 
  by (intro exI[of - to-int-mat M], simp add: mat-mod-to-int-mat)

```

qed

lemma *right-total-MV-Rel*[transfer-rule]: *right-total MV-Rel*

unfolding *right-total-def MV-Rel-def*

proof

fix $v :: 'a \text{ mod-ring } \text{vec}$

show $\exists x. \text{vec-mod } x = \text{to-int-vec } v$

by (*intro exI*[of - *to-int-vec v*], *simp add: vec-mod-to-int-vec*)

qed

lemma *to-int-mod-ring-of-int-mod*: *to-int-mod-ring (of-int x :: 'a mod-ring) = x mod m*

unfolding m **by** *transfer auto*

lemma *vec-mod-v-representative*: *vec-mod v = to-int-vec (map-vec of-int v :: 'a mod-ring vec)*

unfolding *mat-mod-def* **by** (*auto simp: to-int-mod-ring-of-int-mod*)

lemma *mat-mod-N-representative*: *mat-mod N = to-int-mat (map-mat of-int N :: 'a mod-ring mat)*

unfolding *mat-mod-def* **by** (*auto simp: to-int-mod-ring-of-int-mod*)

lemma *left-total-MV-Rel*[transfer-rule]: *left-total MV-Rel*

unfolding *left-total-def MV-Rel-def*[*abs-def*] **using** *vec-mod-v-representative* **by** *blast*

lemma *left-total-MM-Rel*[transfer-rule]: *left-total MM-Rel*

unfolding *left-total-def MM-Rel-def*[*abs-def*] **using** *mat-mod-N-representative* **by** *blast*

lemma *bi-total-MV-Rel*[transfer-rule]: *bi-total MV-Rel*

using *right-total-MV-Rel left-total-MV-Rel* **by** (*metis bi-totalI*)

lemma *bi-total-MM-Rel*[transfer-rule]: *bi-total MM-Rel*

using *right-total-MM-Rel left-total-MM-Rel* **by** (*metis bi-totalI*)

lemma *domain-MV-rel*[transfer-domain-rule]: *Domainp MV-Rel = ($\lambda f. \text{True}$)*

proof

fix $v :: \text{int } \text{vec}$

show *Domainp MV-Rel v = True* **unfolding** *MV-Rel-def*[*abs-def*] *Domainp.simps*

by (*auto simp: vec-mod-v-representative*)

qed

lemma *domain-MM-rel*[transfer-domain-rule]: *Domainp MM-Rel = ($\lambda f. \text{True}$)*

proof

fix $N :: \text{int } \text{mat}$

show *Domainp MM-Rel N = True* **unfolding** *MM-Rel-def*[*abs-def*] *Domainp.simps*

by (*auto simp: mat-mod-N-representative*)

qed

lemma *mem-MV-Rel[transfer-rule]*:

(*MV-Rel* \implies *rel-set MV-Rel* \implies (=)) ($\lambda x Y. \exists y \in Y. \text{vec-mod } x = \text{vec-mod } y$) (\in)

proof (*intro rel-funI iffI*)

fix $x y X Y$ **assume** $xy: \text{MV-Rel } x y$ **and** $XY: \text{rel-set MV-Rel } X Y$

{ **assume** $\exists x' \in X. \text{vec-mod } x = \text{vec-mod } x'$

then obtain x' **where** $x'X: x' \in X$ **and** $xx': \text{vec-mod } x = \text{vec-mod } x'$ **by** *auto*

with xy **have** $x'y: \text{MV-Rel } x' y$ **by** (*auto simp: MV-Rel-def*)

from *rel-setD1[OF XY x'X]* **obtain** y' **where** $\text{MV-Rel } x' y'$ **and** $y' \in Y$ **by** *auto*

with $x'y$

show $y \in Y$ **using** *to-int-mod-ring-hom.vec-hom-inj* **by** (*auto simp: MV-Rel-def*)

}

assume $y \in Y$

from *rel-setD2[OF XY this]* **obtain** x' **where** $x'X: x' \in X$ **and** $x'y: \text{MV-Rel } x' y$ **by** *auto*

from $xy x'y$ **have** $\text{vec-mod } x = \text{vec-mod } x'$ **by** (*auto simp: MV-Rel-def*)

with $x'X$ **show** $\exists x' \in X. \text{vec-mod } x = \text{vec-mod } x'$ **by** *auto*

qed

lemma *mem-MM-Rel[transfer-rule]*:

(*MM-Rel* \implies *rel-set MM-Rel* \implies (=)) ($\lambda x Y. \exists y \in Y. \text{mat-mod } x = \text{mat-mod } y$) (\in)

proof (*intro rel-funI iffI*)

fix $x y X Y$ **assume** $xy: \text{MM-Rel } x y$ **and** $XY: \text{rel-set MM-Rel } X Y$

{ **assume** $\exists x' \in X. \text{mat-mod } x = \text{mat-mod } x'$

then obtain x' **where** $x'X: x' \in X$ **and** $xx': \text{mat-mod } x = \text{mat-mod } x'$ **by** *auto*

with xy **have** $x'y: \text{MM-Rel } x' y$ **by** (*auto simp: MM-Rel-def*)

from *rel-setD1[OF XY x'X]* **obtain** y' **where** $\text{MM-Rel } x' y'$ **and** $y' \in Y$ **by** *auto*

with $x'y$

show $y \in Y$ **using** *to-int-mod-ring-hom.mat-hom-inj* **by** (*auto simp: MM-Rel-def*)

}

assume $y \in Y$

from *rel-setD2[OF XY this]* **obtain** x' **where** $x'X: x' \in X$ **and** $x'y: \text{MM-Rel } x' y$ **by** *auto*

from $xy x'y$ **have** $\text{mat-mod } x = \text{mat-mod } x'$ **by** (*auto simp: MM-Rel-def*)

with $x'X$ **show** $\exists x' \in X. \text{mat-mod } x = \text{mat-mod } x'$ **by** *auto*

qed

lemma *conversep-MM-Rel-OO-MM-Rel [simp]*: $\text{MM-Rel}^{-1-1} \text{ OO } \text{MM-Rel} = (=)$

using *mat-mod-to-int-mat apply (intro ext, auto simp: OO-def MM-Rel-def)*

using *to-int-mod-ring-hom.mat-hom-inj* **by** *auto*

lemma *MM-Rel-OO-conversep-MM-Rel [simp]*: $\text{MM-Rel} \text{ OO } \text{MM-Rel}^{-1-1} = (\lambda M M'. \text{mat-mod } M = \text{mat-mod } M')$

by (*intro ext, auto simp: OO-def MM-Rel-def mat-mod-N-representative*)

lemma *conversep-MM-Rel-OO-eq-m* [simp]: $MM-Rel^{-1-1} OO (\lambda M M' . mat-mod M = mat-mod M') = MM-Rel^{-1-1}$

by (*intro ext, auto simp: OO-def MM-Rel-def*)

lemma *eq-m-OO-MM-Rel* [simp]: $(\lambda M M' . mat-mod M = mat-mod M') OO MM-Rel = MM-Rel$

by (*intro ext, auto simp: OO-def MM-Rel-def*)

lemma *eq-mset-MM-Rel* [transfer-rule]:

$(rel-mset MM-Rel ==> rel-mset MM-Rel ==> (=)) (rel-mset (\lambda M M' . mat-mod M = mat-mod M')) (=)$

proof (*intro rel-funI iffI*)

fix $A B X Y$

assume $AX: rel-mset MM-Rel A X$ **and** $BY: rel-mset MM-Rel B Y$

{

assume $AB: rel-mset (\lambda M M' . mat-mod M = mat-mod M') A B$

from AX **have** $rel-mset MM-Rel^{-1-1} X A$ **by** (*simp add: multiset.rel-flip*)

note $rel-mset-OO[OF this AB]$

note $rel-mset-OO[OF this BY]$

then show $X = Y$ **by** (*simp add: multiset.rel-eq*)

}

assume $X = Y$

with BY **have** $rel-mset MM-Rel^{-1-1} X B$ **by** (*simp add: multiset.rel-flip*)

from $rel-mset-OO[OF AX this]$

show $rel-mset (\lambda M M' . mat-mod M = mat-mod M') A B$ **by** *simp*

qed

lemma *vec-mset-MV-Rel*[transfer-rule]:

$(MV-Rel ==> (=)) (\lambda v . vec-mset (vec-mod v)) (\lambda v . image-mset (to-int-mod-ring) (vec-mset v))$

unfolding $MV-Rel-def rel-fun-def$

proof (*intro allI impI subset-antisym subsetI*)

fix $x :: int\ vec$ **fix** $y :: 'a\ mod-ring\ vec$

assume $assm: vec-mod x = to-int-vec y$

have $image-mset\ to-int-mod-ring\ (vec-mset\ y) = vec-mset\ (to-int-vec\ y)$

using $inj-zero-hom.vec-hom-mset\ to-int-mod-ring-hom.inj-zero-hom-axioms$ **by** *auto*

then show $vec-mset\ (vec-mod\ x) = image-mset\ to-int-mod-ring\ (vec-mset\ y)$

using $assm$ **by** *simp*

qed

lemma *vec-count-MV-Rel-direct*:

assumes $MV-Rel\ v1\ v2$

shows $count-vec\ v2\ i = count-vec\ (vec-mod\ v1)\ (to-int-mod-ring\ i)$

proof–

have $eq-vecs: to-int-vec\ v2 = vec-mod\ v1$ **using** $assms$ **unfolding** $MV-Rel-def$ **by** *simp*

have $count-vec\ v2\ i = count\ (vec-mset\ v2)\ i$ **by** *simp*

also have $1: \dots = count\ (image-mset\ to-int-mod-ring\ (vec-mset\ v2))\ (to-int-mod-ring\ i)$

i)
using *count-image-mset-inj* **by** (*metis to-int-mod-ring-hom.inj-f*)
also have $2: \dots = \text{count } (\text{vec-mset } (\text{vec-mod } v1)) \text{ (to-int-mod-ring } i)$ **using** *assms*
by (*simp add: eq-vecs inj-zero-hom.vec-hom-mset to-int-mod-ring-hom.inj-zero-hom-axioms*)

finally show $\text{count-vec } v2 \ i = \text{count-vec } (\text{vec-mod } v1) \text{ (to-int-mod-ring } i)$
by (*simp add: 1 2*)
qed

lemma *MM-Rel-MV-Rel-row*: $MM\text{-Rel } A \ B \implies i < \text{dim-row } A \implies MV\text{-Rel } (\text{row } A \ i) \ (\text{row } B \ i)$
unfolding *MM-Rel-def MV-Rel-def*
by (*metis index-map-mat(2) mat-mod-dim(1) mat-mod-vec-mod-row row-map-mat*)

lemma *MM-Rel-MV-Rel-col*: $MM\text{-Rel } A \ B \implies j < \text{dim-col } A \implies MV\text{-Rel } (\text{col } A \ j) \ (\text{col } B \ j)$
unfolding *MM-Rel-def MV-Rel-def*
using *index-map-mat(3) mat-mod-dim(2) mat-mod-vec-mod-col col-map-mat* **by** (*metis*)

end
end

10 Variations on Fisher's Inequality

theory *Fishers-Inequality-Variations* **imports** *Dual-Systems Rank-Argument-General Vector-Matrix-Mod Linear-Bound-Argument*
begin

10.1 Matrix mod properties

This is reasoning on properties specific to incidence matrices under *mat-mod*. Ultimately, this definition was not used in the final proof but it is left as is in case of future use

context *mat-mod*
begin

lemma *mat-mod-proper-iff*: $\text{proper-inc-mat } (\text{mat-mod } N) \iff \text{proper-inc-mat } N$
by (*simp add: proper-inc-mat-def*)

lemma *mat-mod-rep-num-eq*: $i < \text{dim-row } N \implies \text{elements-mat } N \subseteq \{0..<m\} \implies$
 $\text{mat-rep-num } (\text{mat-mod } N) \ i = \text{mat-rep-num } N \ i$
by (*simp add: mat-mod-count-row-eq mat-rep-num-def*)

lemma *mat-point-index-eq*: $\text{elements-mat } N \subseteq \{0..<m\} \implies$
 $\text{mat-point-index } (\text{mat-mod } N) \ I = \text{mat-point-index } N \ I$

```

by (simp add: mat-mod-eq-cond)

lemma mod-mat-inter-num-eq: elements-mat  $N \subseteq \{0..<m\} \implies$ 
  mat-inter-num (mat-mod  $N$ )  $j1\ j2 = \text{mat-inter-num } N\ j1\ j2$ 
by (simp add: mat-mod-eq-cond)

lemma mod-mat-block-size: elements-mat  $N \subseteq \{0..<m\} \implies \text{mat-block-size (mat-mod } N) j = \text{mat-block-size } N\ j$ 
by (simp add: mat-mod-eq-cond)

lemma mat-mod-non-empty-col-iff: elements-mat  $M \subseteq \{0..<m\} \implies$ 
  non-empty-col (mat-mod  $M$ )  $j \longleftrightarrow \text{non-empty-col } M\ j$ 
using mat-mod-eq-cond by auto
end

context mat-mod-type
begin

lemma mat-rep-num-MM-Rel:
  assumes MM-Rel  $A\ B$ 
  assumes  $i < \text{dim-row } A$ 
  shows mat-rep-num (mat-mod  $A$ )  $i = \text{mat-rep-num } B\ i$ 
  unfolding mat-rep-num-def using vec-count-MV-Rel-direct assms mat-mod-vec-mod-row
  row-map-mat
  by (metis MM-Rel-def MV-Rel-def index-map-mat(2) mat-mod-dim(1) to-int-mod-ring-hom.hom-one)

lemma mat-block-size-MM-Rel:
  assumes MM-Rel  $A\ B$ 
  assumes  $j < \text{dim-col } A$ 
  shows mat-block-size (mat-mod  $A$ )  $j = \text{mat-block-size } B\ j$ 
  unfolding mat-block-size-def using vec-count-MV-Rel-direct assms MM-Rel-MV-Rel-col
  by (metis mat-mod-vec-mod-col to-int-mod-ring-hom.hom-one)

lemma mat-inter-num-MM-Rel:
  assumes MM-Rel  $A\ B$ 
  assumes  $j1 < \text{dim-col } A\ j2 < \text{dim-col } B$ 
  shows mat-inter-num (mat-mod  $A$ )  $j1\ j2 = \text{mat-inter-num } B\ j1\ j2$ 
  unfolding mat-inter-num-def using assms index-map-mat mat-mod-dim(2)
  by (smt (z3) Collect-cong MM-Rel-def to-int-mod-ring-hom.hom-1 to-int-mod-ring-hom.hom-one)

  Lift 01 and mat mod equivalence on 0-1 matrices

lemma of-int-mod-ring-lift-01-eq:
  assumes zero-one-matrix  $N$ 
  shows map-mat (of-int-mod-ring)  $N = (\text{lift-01-mat})\ N$ 
  apply (auto simp add: mat-eq-iff[of map-mat (of-int-mod-ring)  $N$  lift-01-mat  $N$ ])
  using assms zero-one-matrix.M-not-one-simp by fastforce

```

```

lemma to-int-mod-ring-lift-01-eq:
  assumes zero-one-matrix N
  shows to-int-mat N = (lift-01-mat) N
  apply (auto simp add: mat-eq-iff[of to-int-mat N lift-01-mat N])
  using assms using zero-one-matrix.M-not-zero-simp by fastforce

end

```

10.2 The Odd-town Problem

The odd-town problem [1] is perhaps one of the most common introductory problems for applying the linear algebra bound method to a combinatorial problem. In mathematical literature, it is considered simpler than Fisher's Inequality, however presents some interesting challenges to formalisation. Most significantly, it considers the incidence matrix to have elements of types integers mod 2.

Initially, define a locale to represent the odd town context (a town with v people, and b groups) which must each be of odd size, but have an even intersection number with any other group

```

locale odd-town = ordered-design +
  assumes odd-groups: bl ∈# B ⇒ odd (card bl)
  and even-inters: bl1 ∈# B ⇒ bl2 ∈# (B - {#bl1#}) ⇒ even (bl1 |∩| bl2)
begin

```

```

lemma odd-town-no-repeat-clubs: distinct-mset B

```

```

proof (rule ccontr)

```

```

  assume  $\neg$  distinct-mset B

```

```

  then obtain a where ain: a ∈# B and countne: count B a ≠ 1

```

```

    by (auto simp add: distinct-mset-def)

```

```

  then have count B a > 1

```

```

    using nat-less-le by auto

```

```

  then have ain2: a ∈# (B - {#a#})

```

```

    by (simp add: in-diff-count)

```

```

  then have odd (a |∩| a) using odd-groups ain by simp

```

```

  thus False using even-inters ain ain2

```

```

    by blast

```

```

qed

```

```

lemma odd-blocks-mat-block-size: j < dim-col N ⇒ odd (mat-block-size N j)

```

```

  using mat-block-size-conv odd-groups

```

```

  by (metis dim-col-is-b valid-blocks-index)

```

```

lemma odd-block-size-mod-2:

```

```

  assumes CARD('b::prime-card) = 2

```

```

  assumes  $j < b$ 

```

```

  shows of-nat (card (Bs ! j)) = (1 :: 'b mod-ring)

```

```

proof –

```

```

  have cb2: CARD('b) = 2 using assms by simp

```

then have $odd (card (\mathcal{B}s ! j))$ **using** $\langle j < b \rangle$ *odd-groups* **by** *auto*
then show $of\text{-}nat (card (\mathcal{B}s ! j)) = (1 :: 'b \text{ mod-ring})$
by (*transfer' fixing: j Bs, simp add: cb2*) *presburger*
qed

lemma *valid-indices-block-min*: $j1 < dim\text{-}col\ N \implies j2 < dim\text{-}col\ N \implies j1 \neq j2$
 $\implies b \geq 2$
by *simp*

lemma *even-inter-mat-intersections*: $j1 < dim\text{-}col\ N \implies j2 < dim\text{-}col\ N \implies j1 \neq j2$
 $\implies even (mat\text{-}inter\text{-}num\ N\ j1\ j2)$
using *even-inters mat-inter-num-conv valid-indices-block-min*
by (*metis dim-col-is-b obtains-two-diff-block-indices*)

lemma *even-inter-mod-2*:
assumes $CARD('b::prime\text{-}card) = 2$
assumes $i < b$ **and** $jlt: j < b$ **and** $ne: i \neq j$
shows $of\text{-}nat ((\mathcal{B}s ! i) |\cap| (\mathcal{B}s ! j)) = (0 :: 'b \text{ mod-ring})$
proof –
have $cb2: CARD('b) = 2$ **using** *assms* **by** *simp*
have $even ((\mathcal{B}s ! i) |\cap| (\mathcal{B}s ! j))$ **using** *even-inters assms*
using *blocks-index-ne-belong blocks-list-length valid-blocks-index* **by** *presburger*
then show $of\text{-}nat ((\mathcal{B}s ! i) |\cap| (\mathcal{B}s ! j)) = (0 :: 'b \text{ mod-ring})$
by (*transfer' fixing: i j Bs, simp add: cb2*)
qed

end

The odd town locale must be simple by definition

sublocale *odd-town* \subseteq *ordered-simple-design*
using *odd-town-no-repeat-clubs* **by** (*unfold-locales*) (*meson distinct-mset-def*)

context *odd-town*
begin

The upper bound lemma (i.e. variation on Fisher's) for the odd town property using the linear bound argument. Notice it follows exactly the same pattern as the generalised version, however it's sum manipulation argument is significantly simpler (in line with the mathematical proofs)

lemma *upper-bound-clubs*:
assumes $CARD('b::prime\text{-}card) = 2$
shows $b \leq v$
proof –
have $cb2: CARD('b) = 2$ **using** *assms* **by** *simp*
then interpret $mmt: mat\text{-}mod\text{-}type\ 2\ TYPE('b::prime\text{-}card)$
using *assms* **by** (*unfold-locales*) (*simp-all*)
define $N2 :: 'b \text{ mod-ring mat}$ **where** $N2 \equiv lift\text{-}01\text{-}mat\ N$
show *?thesis* **proof** (*intro lin-bound-argument2[of N2]*)

```

show distinct (cols (N2)) using lift-01-distinct-cols-N N2-def by simp
show n2cm:N2 ∈ carrier-mat v b using N2-def N-carrier-mat-01-lift by simp
have scalar-prod-odd:  $\bigwedge i. i < b \implies ((\text{col } N2 \ i) \cdot (\text{col } N2 \ i)) = 1$ 
using scalar-prod-block-size-lift-01 N2-def odd-block-size-mod-2 assms by (metis
cb2)
have scalar-prod-even:  $\bigwedge i \ j. i < b \implies j < b \implies i \neq j \implies ((\text{col } N2 \ i) \cdot (\text{col } N2 \ j)) = 0$ 
using even-inter-mod-2 scalar-prod-inter-num-lift-01 N2-def assms by metis
show  $\bigwedge f. \text{vec } v \ (\lambda i. \sum j = 0..<b. f \ (\text{col } N2 \ j) * (\text{col } N2 \ j) \$ i) = 0_v \ v \implies$ 
 $\forall v \in \text{set} \ (\text{cols } N2). f \ v = 0$ 
proof (auto)
fix f v
assume eq0:  $\text{vec } v \ (\lambda i. \sum j = 0..<\text{length } \mathcal{B}s. f \ (\text{col } N2 \ j) * (\text{col } N2 \ j) \$ i) =$ 
 $0_v \ v$ 
assume vin:  $v \in \text{set} \ (\text{cols } N2)$ 
define c where  $c \equiv (\lambda j. f \ (\text{col } N2 \ j))$ 
have inner:  $\bigwedge j \ l. v \ \$ \ l * (c \ j * (\text{col } N2 \ j) \$ l) = c \ j * v \ \$ \ l * (\text{col } N2 \ j) \$ \ l$ 
using mult.commute by auto
obtain j' where v-def:  $\text{col } N2 \ j' = v$  and jvlt:  $j' < \text{dim-col } N2$ 
using vin by (metis cols-length cols-nth index-less-size-conv nth-index)
then have jvltb:  $j' < b$  using n2cm by simp
then have even0:  $\bigwedge j. j \in \{0..<b\} - \{j'\} \implies c \ j * (v \cdot (\text{col } N2 \ j)) = 0$ 
using scalar-prod-even v-def by fastforce
have vinc:  $v \in \text{carrier-vec } v$  using n2cm set-cols-carrier vin by blast
then have  $0 = v \cdot \text{vec } v \ (\lambda i. \sum j = 0..<b. c \ j * (\text{col } N2 \ j) \$ i)$ 
using eq0 c-def by auto
also have  $\dots = (\sum l = 0..<\text{dim-row } N2. v \ \$ \ l * (\sum j = 0..<\text{dim-col } N2. c \ j * (\text{col } N2 \ j) \$ l))$ 
unfolding scalar-prod-def using n2cm by auto
also have  $\dots = (\sum l = 0..<\text{dim-row } N2. (\sum j = 0..<\text{dim-col } N2. v \ \$ \ l * c \ j * (\text{col } N2 \ j) \$ l))$ 
by (simp add: sum-distrib-left)
also have  $\dots = (\sum j \in \{0..<\text{dim-col } N2\}. v \cdot (c \ j \cdot_v (\text{col } N2 \ j)))$ 
using sum.swap scalar-prod-def[of v] by simp
also have  $\dots = v \cdot (c \ j' \cdot_v v) + (\sum j \in \{0..<\text{dim-col } N2\} - \{j'\}. v \cdot (c \ j \cdot_v (\text{col } N2 \ j)))$ 
using jvlt sum.remove[of {0..<dim-col N2} j' λ j. v · (c j ·v (col N2 j))]
v-def by simp
also have  $\dots = v \cdot (c \ j' \cdot_v v) + (\sum j \in \{0..<b\} - \{j'\}. c \ j * (v \cdot (\text{col } N2 \ j)))$ 
using n2cm scalar-prod-smult-distrib col-dim v-def by force
also have  $\dots = v \cdot (c \ j' \cdot_v v)$ 
using even0 by (simp add: sum.neutral)
also have  $\dots = (c \ j') * (v \cdot v)$ 
using scalar-prod-smult-distrib by (simp add: v-def)
finally have  $0 = (c \ j')$  using v-def jvlt n2cm scalar-prod-odd by fastforce
then show  $f \ v = 0$  using c-def v-def by simp
qed
qed

```

qed

end

end

theory *Fishers-Inequality-Root*

imports

Set-Multiset-Extras

Matrix-Vector-Extras

Design-Extras

Incidence-Matrices

Dual-Systems

Rank-Argument-General

Linear-Bound-Argument

Fishers-Inequality

Vector-Matrix-Mod

Fishers-Inequality-Variations

begin

end

References

- [1] L. Babai and P. Frankl. *Linear Algebra Methods in Combinatorics*. 2.1 edition, 2020.
- [2] B. Bukh. Lecture notes in algebraic Methods in Combinatorics: Rank argument, 2014.
- [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] R. A. Fisher. An Examination of the Different Possible Solutions of a Problem in Incomplete Blocks. *Annals of Eugenics*, 10(1):52–75, 1940.
- [5] C. D. Godsil. Tools from Linear Algebra. In L. L. Graham RL, Grötschel M, editor, *Handbook of Combinatorics*, volume 2. Elsevier, Amsterdam.
- [6] S. Herke. Math3301 lecture notes in combinatorial design theory, July 2016.

- [7] S. Jukna. *Extremal Combinatorics*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [8] D. Stinson. *Combinatorial Designs: Constructions and Analysis*. Springer, 2004.