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

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

1 Micellaneous Multset/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* $\Longrightarrow$ *finite B* $\Longrightarrow$ *card A* = *card B* $\Longrightarrow$

card (*A* ∩ *B*) = *card A* $\Longrightarrow$ *A* = *B*
⟨*proof*⟩

lemma *insert-filter-set-true*: *P x* $\Longrightarrow$ {*a* ∈ (*insert x A*) . *P a*} = *insert x* {*a* ∈ *A* . *P a*}
⟨*proof*⟩

lemma *insert-filter-set-false*: ¬ *P x* $\Longrightarrow$ {*a* ∈ (*insert x A*) . *P a*} = {*a* ∈ *A* . *P a*}
⟨*proof*⟩

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* ∈# *A* **and** *y* ∈# *A* − {#*x*#}
⟨*proof*⟩

lemma *obtain-two-items-mset-filter*:
assumes *size* {#*a* ∈# *A* . *P a* #} > *1*
obtains *x y* **where** *x* ∈# *A* **and** *y* ∈# *A* − {#*x*#} **and** *P x* **and** *P y*
⟨*proof*⟩

lemma *size-count-mset-ss*:
assumes *finite B*
assumes (*set-mset A*) ⊆ *B*
shows *size A* = (∑ *x* ∈ *B* . *count A x*)
⟨*proof*⟩

lemma *mset-list-by-index*: *mset* (*xs*) = *image-mset* (λ *i* . (*xs* ! *i*)) (*mset-set* {..<*length xs*})
⟨*proof*⟩

lemma *count-mset-split-image-filter*:
assumes ⋀ *x*. *x* ∈#*A* $\Longrightarrow$ *a* ≠ *g x*
shows *count* (*image-mset* (λ*x*. *if P x then a else g x*) *A*) *a* = *size* (*filter-mset P A*)
⟨*proof*⟩

lemma *count-mset-split-image-filter-not*:
assumes $\bigwedge x.\ x \in\# A \Longrightarrow b \neq f\ x$
shows *count* (*image-mset* ($\lambda x.$ *if* $P\ x$ *then* $f\ x$ *else* b) A) b = *size* (*filter-mset* ($\lambda x.\ \neg\ P\ x$) A)
⟨*proof*⟩

lemma *removeAll-size-lt*: *size* (*removeAll-mset* $C\ M$) $\leq$ *size* M
⟨*proof*⟩

lemma *mset-image-eq-filter-eq*: A = *image-mset* $f\ B \Longrightarrow$
filter-mset $P\ A$ = (*image-mset* f (*filter-mset* ($\lambda\ x.\ P\ (f\ x)$) B))
⟨*proof*⟩

1.3 Permutation on Sets and Multisets

lemma *elem-permutation-of-set-empty-iff*: *finite* $A \Longrightarrow$ *xs* $\in$ *permutations-of-set* $A \Longrightarrow$
$xs = [] \longleftrightarrow A = \{\}$
⟨*proof*⟩

lemma *elem-permutation-of-mset-empty-iff*: *xs* $\in$ *permutations-of-multiset* $A \Longrightarrow$ $xs = [] \longleftrightarrow A = \{\#\}$
⟨*proof*⟩

1.4 Lists

Further lemmas on the relationship between lists and multisets

lemma *count-distinct-mset-list-index*: *i1* $<$ *length xs* $\Longrightarrow$ *i2* $<$ *length xs* $\Longrightarrow$ *i1* $\neq$ *i2* $\Longrightarrow$
distinct-mset (*mset xs*) $\Longrightarrow$ *xs* ! *i1* $\neq$ *xs* ! *i2*
⟨*proof*⟩

lemma *index-remove1-mset-ne*:
assumes $x \in\#$ (*mset xs*)
assumes $y \in\#$ *remove1-mset* x (*mset xs*)
assumes *xs* ! *j1* = x
assumes *j1* $<$ *length xs*
obtains *j2* **where** *xs* ! *j2* = y **and** *j2* $<$ *length xs* **and** *j1* $\neq$ *j2*
⟨*proof*⟩

lemma *count-list-mset*: *count-list xs* x = *count* (*mset xs*) x
⟨*proof*⟩

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*⟩

lemma *count-min-2-indices*: $i1 \neq i2 \Longrightarrow xs\ !\ i1 = x \Longrightarrow xs\ !\ i2 = x \Longrightarrow i1 < length\ xs \Longrightarrow$
$i2 < length\ xs \Longrightarrow count\ (mset\ xs)\ x \geq 2$
⟨*proof*⟩

lemma *obtain-set-list-item*:
assumes $x \in set\ xs$
obtains i **where** $i < length\ xs$ **and** $xs\ !\ i = x$
⟨*proof*⟩

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*: $(\sum\ l \in A\ .\ (\sum\ i \in B\ .\ (\sum\ j \in C\ .\ g\ l\ i\ j))) =$
$(\sum\ i \in B\ .\ (\sum\ j \in C\ .\ (\sum\ l \in A\ .\ g\ l\ i\ j)))$
⟨*proof*⟩

lemma *double-sum-mult-hom*:
fixes $k :: 'b :: \{comm\text{-}ring\text{-}1\}$
shows $(\sum\ i \in A\ .\ (\sum\ j \in g\ i\ .\ k * (f\ i\ j))) = k * (\sum\ i \in A\ .\ (\sum\ j \in g\ i\ .\ f\ i\ j))$
⟨*proof*⟩

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*⟩

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*⟩

end

context *comm-ring-1*
begin

lemma *double-sum-split-case-square*:
assumes *finite A*
shows $(\sum\ i \in A\ .\ f\ i\ )\hat{}2 = (\sum\ i \in A\ .\ (f\ i * f\ i)) + (\sum\ i \in A\ .\ (\sum\ j \in (A - \{i\})\ .\ f\ i * f\ j))$
⟨*proof*⟩

lemma *double-sum-split-square-diff*: *finite* $\{0..<x\} \Longrightarrow$

$(\sum\ i \in \{0..<x\}\ .\ (\sum\ j \in (\{0..<x\} - \{i\})\ .\ c\ i * c\ j)) =$
$(\sum\ i \in \{0..<x\}\ .\ c\ i)\hat{}2 - (\sum\ i \in \{0..<x\}\ .\ c\ i * c\ i)$
⟨*proof*⟩

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* ⇒ *'a* **where**
sum-vec v ≡ *sum* (λ *i* . *v* $ *i*) {*0..<dim-vec v*}

lemma *sum-vec-vNil*[*simp*]: *sum-vec vNil* = *0*
⟨*proof*⟩

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

lemma *sum-vec-list*: *sum-list* (*list-of-vec v*) = *sum-vec v*
⟨*proof*⟩

lemma *sum-vec-mset*: *sum-vec v* = $(\sum\ x \in\#\ (mset\ (list\text{-}of\text{-}vec\ v))\ .\ x)$
⟨*proof*⟩

lemma *dim-vec-vCons-ne-0*: *dim-vec* (*vCons a v*) > *0*
⟨*proof*⟩

lemma *sum-vec-vCons-lt*:
assumes $\bigwedge$ *i*. *i* < *dim-vec* (*vCons a v*) ⟹ (*vCons a v*) $ *i* ≤ (*n* ::*int*)
assumes *sum-vec v* ≤ *m*
shows *sum-vec* (*vCons a v*) ≤ *n* + *m*
⟨*proof*⟩

lemma *sum-vec-one-zero*:
assumes $\bigwedge$ *i*. *i* < *dim-vec* (*v* :: *int vec*) ⟹ *v* $ *i* ≤ (*1* ::*int*)
shows *sum-vec v* ≤ *dim-vec v*
⟨*proof*⟩

Definition to convert a vector to a multiset

definition *vec-mset*:: *'a vec* ⇒ *'a multiset* **where**
vec-mset v ≡ *image-mset* (*vec-index v*) (*mset-set* {..<*dim-vec v*})

lemma *vec-elem-exists-mset*: (∃ *i* ∈ {..<*dim-vec v*}. *v* $ *i* = *x*) ⟷ *x* ∈# *vec-mset v*
⟨*proof*⟩

lemma *mset-vec-same-size*: *dim-vec v* = *size* (*vec-mset v*)
⟨*proof*⟩

lemma *mset-vec-eq-mset-list*: *vec-mset v* = *mset* (*list-of-vec v*)
⟨*proof*⟩

lemma *vec-mset-img-map*: *image-mset f* (*mset* (*xs*)) = *vec-mset* (*map-vec f* (*vec-of-list xs*))
⟨*proof*⟩

lemma *vec-mset-vNil*: *vec-mset vNil* = {#}
⟨*proof*⟩

lemma *vec-mset-vCons*: *vec-mset* (*vCons x v*) = *add-mset x* (*vec-mset v*)
⟨*proof*⟩

lemma *vec-mset-set*: *vec-set v* = *set-mset* (*vec-mset v*)
⟨*proof*⟩

lemma *vCons-set-contains-in*: *a* ∈ *set*$_v$ *v* ⟹ *set*$_v$ (*vCons a v*) = *set*$_v$ *v*
⟨*proof*⟩

lemma *vCons-set-contains-add*: *a* ∉ *set*$_v$ *v* ⟹ *set*$_v$ (*vCons a v*) = *set*$_v$ *v* ∪ {*a*}
⟨*proof*⟩

lemma *setv-vec-mset-not-in-iff*: *a* ∉ *set*$_v$ *v* ⟷ *a* ∉# *vec-mset v*
⟨*proof*⟩

Abbreviation for counting occurrences of an element in a vector

abbreviation *count-vec v a* ≡ *count* (*vec-mset v*) *a*

lemma *vec-count-lt-dim*: *count-vec v a* ≤ *dim-vec v*
⟨*proof*⟩

lemma *count-vec-empty*: *dim-vec v* = *0* ⟹ *count-vec v a* = *0*
⟨*proof*⟩

lemma *count-vec-vNil*: *count-vec vNil a* = *0*
⟨*proof*⟩

lemma *count-vec-vCons*: *count-vec* (*vCons aa v*) *a* = (*if* (*aa* = *a*) *then count-vec*

$v\ a + 1$ *else count-vec* $v\ a$)
⟨*proof*⟩

lemma *elem-exists-count-min*: $\exists\ i \in \{..<dim\text{-}vec\ v\}.\ v\ \$\ i = x \Longrightarrow count\text{-}vec\ v\ x \geq 1$
⟨*proof*⟩

lemma *count-vec-count-mset*: *vec-mset* v = *image-mset* $f\ A \Longrightarrow$ *count-vec* $v\ a$ = *count* (*image-mset* $f\ A$) a
⟨*proof*⟩

lemma *count-vec-alt-list*: *count-vec* $v\ a$ = *length* (*filter* $(\lambda y.\ a = y)$ (*list-of-vec* v))
⟨*proof*⟩

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

lemma *count-vec-sum-ones*:
fixes $v :: 'a :: \{ring\text{-}1\}\ vec$
assumes $\bigwedge i.\ i < dim\text{-}vec\ v \Longrightarrow v\ \$\ i = 1 \vee v\ \$\ i = 0$
shows *of-nat* (*count-vec* $v\ 1$) = *sum-vec* v
⟨*proof*⟩

lemma *count-vec-two-elems*:
fixes $v :: 'a :: \{zero\text{-}neq\text{-}one\}\ vec$
assumes $\bigwedge i.\ i < dim\text{-}vec\ v \Longrightarrow v\ \$\ i = 1 \vee v\ \$\ i = 0$
shows *count-vec* $v\ 1$ + *count-vec* $v\ 0$ = *dim-vec* v
⟨*proof*⟩

lemma *count-vec-sum-zeros*:
fixes $v :: 'a :: \{ring\text{-}1\}\ vec$
assumes $\bigwedge i.\ i < dim\text{-}vec\ v \Longrightarrow v\ \$\ i = 1 \vee v\ \$\ i = 0$
shows *of-nat* (*count-vec* $v\ 0$) = *of-nat* (*dim-vec* v) − *sum-vec* v
⟨*proof*⟩

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

lemma *setv-not-in-count0-iff*: $a \notin set_v\ v \longleftrightarrow$ *count-vec* $v\ a = 0$
⟨*proof*⟩

lemma *sum-count-vec*:
assumes *finite* ($set_v\ v$)
shows $(\sum\ i \in set_v\ v.$ *count-vec* $v\ i) =$ *dim-vec* v
⟨*proof*⟩

lemma *sum-setv-subset-eq*:
assumes *finite A*
assumes $set_v\ v \subseteq A$
shows $(\sum\ i \in set_v\ v.\ count\text{-}vec\ v\ i) = (\sum\ i \in A.\ count\text{-}vec\ v\ i)$
⟨*proof*⟩

lemma *sum-count-vec-subset*: $finite\ A \Longrightarrow set_v\ v \subseteq A \Longrightarrow (\sum\ i \in A.\ count\text{-}vec\ v\ i) = dim\text{-}vec\ v$
⟨*proof*⟩

An abbreviation for checking if an element is in a vector

abbreviation *vec-contains* :: $'a \Rightarrow 'a\ vec \Rightarrow bool$ (**infix** $\in\$$ *50*)**where**
$a \in\$\ v \equiv a \in set_v\ v$

lemma *vec-set-mset-contains-iff*: $a \in\$\ v \longleftrightarrow a \in\#\ vec\text{-}mset\ v$
⟨*proof*⟩

lemma *vec-contains-count-gt1-iff*: $a \in\$\ v \longleftrightarrow count\text{-}vec\ v\ a \geq 1$
⟨*proof*⟩

lemma *vec-contains-obtains-index*:
assumes $a \in\$\ v$
obtains i **where** $i < dim\text{-}vec\ v$ **and** $v\ \$\ i = a$
⟨*proof*⟩

lemma *vec-count-eq-list-count*: $count\ (mset\ xs)\ a = count\text{-}vec\ (vec\text{-}of\text{-}list\ xs)\ a$
⟨*proof*⟩

lemma *vec-contains-col-elements-mat*:
assumes $j < dim\text{-}col\ M$
assumes $a \in\$\ col\ M\ j$
shows $a \in elements\text{-}mat\ M$
⟨*proof*⟩

lemma *vec-contains-row-elements-mat*:
assumes $i < dim\text{-}row\ M$
assumes $a \in\$\ row\ M\ i$
shows $a \in elements\text{-}mat\ M$
⟨*proof*⟩

lemma *vec-contains-img*: $a \in\$\ v \Longrightarrow f\ a \in\$\ (map\text{-}vec\ f\ v)$
⟨*proof*⟩

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

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

⟨*proof*⟩

lemma *all-ones-index* [*simp*]: $i < n \Longrightarrow u_v\ n\ \$\ i = 1$
⟨*proof*⟩

lemma *dim-vec-mult-vec-mat* [*simp*]: *dim-vec* $(v\ _v{*}\ A) =$ *dim-col* A
⟨*proof*⟩

lemma *all-ones-vec-smult*[*simp*]: $i < n \Longrightarrow ((k :: ('a :: \{one, zero, monoid\text{-}mult\}))\ \cdot_v\ (u_v\ n))\ \$\ i = k$
⟨*proof*⟩

Extra lemmas on existing vector operations

lemma *smult-scalar-prod-sum*:
fixes $x :: 'a :: \{comm\text{-}ring\text{-}1\}$
assumes $vx \in carrier\text{-}vec\ n$
assumes $vy \in carrier\text{-}vec\ n$
shows $(\sum\ i \in \{0..<n\}\ .((x \cdot_v vx)\ \$\ i) * ((y \cdot_v vy)\ \$\ i)) = x * y * (vx \cdot vy)$
⟨*proof*⟩

lemma *scalar-prod-double-sum-fn-vec*:
fixes $c :: nat \Rightarrow ('a :: \{comm\text{-}semiring\text{-}0\})$
fixes $f :: nat \Rightarrow 'a\ vec$
assumes $\bigwedge j\ .\ j < k \Longrightarrow dim\text{-}vec\ (f\ j) = n$
shows $(vec\ n\ (\lambda i.\ \sum j = 0..<k.\ c\ j * (f\ j)\ \$\ i)) \cdot (vec\ n\ (\lambda i.\ \sum j = 0..<k.\ c\ j * (f\ j)\ \$\ i)) =$
$(\sum\ j1 \in \{0..<k\}\ .\ c\ j1 * c\ j1 * ((f\ j1) \cdot (f\ j1))) +$
$(\sum\ j1 \in \{0..<k\}\ .\ (\sum\ j2 \in (\{0..< k\} - \{j1\})\ .\ c\ j1 * c\ j2 * ((f\ j1) \cdot (f\ j2))))$
⟨*proof*⟩

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

lemma *map-vec-compose*: *map-vec* f (*map-vec* $g\ v$) = *map-vec* $(f \circ g)\ v$
⟨*proof*⟩

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

lemma *all-ones-mat-index*[*simp*]: $i <$ *dim-row* $(J_m\ n) \Longrightarrow j <$ *dim-col* $(J_m\ n) \Longrightarrow J_m\ n\ \$\$\ (i, j) = 1$
⟨*proof*⟩

lemma *all-ones-mat-dim-row*[*simp*]: *dim-row* $(J_m\ n) = n$
⟨*proof*⟩

lemma *all-ones-mat-dim-col*[*simp*]: *dim-col* $(J_m\ n) = n$
⟨*proof*⟩

Basic lemmas on existing matrix operations

lemma *index-mult-vec-mat*[*simp*]: $j <$ *dim-col* $A \Longrightarrow (v\ _v{*}\ A)\ \$\ j = v \cdot$ *col* $A\ j$
⟨*proof*⟩

lemma *transpose-mat-mult-entries*: $i <$ *dim-row* $A \Longrightarrow j <$ *dim-row* $A \Longrightarrow$
$(A * A^T)\ \$\$\ (i, j) = (\sum k \in \{0..<(\text{dim-col } A)\}.\ (A\ \$\$\ (i, k)) * (A\ \$\$\ (j, k)))$
⟨*proof*⟩

lemma *transpose-mat-elems*: *elements-mat* $A =$ *elements-mat* A^T
⟨*proof*⟩

lemma *row-elems-subset-mat*: $i <$ *dim-row* $N \Longrightarrow$ *vec-set* (*row* $N\ i$) $\subseteq$ *elements-mat* N
⟨*proof*⟩

lemma *col-elems-subset-mat*: $i <$ *dim-col* $N \Longrightarrow$ *vec-set* (*col* $N\ i$) $\subseteq$ *elements-mat* N
⟨*proof*⟩

lemma *obtain-row-index*:
assumes $r \in$ *set* (*rows* M)
obtains i **where** *row* $M\ i = r$ **and** $i <$ *dim-row* M
⟨*proof*⟩

lemma *row-prop-cond*: $(\bigwedge i.\ i <$ *dim-row* $M \Longrightarrow P$ (*row* $M\ i$)) $\Longrightarrow r \in$ *set* (*rows* M) $\Longrightarrow P\ r$
⟨*proof*⟩

lemma *obtain-col-index*:
assumes $c \in$ *set* (*cols* M)
obtains j **where** *col* $M\ j = c$ **and** $j <$ *dim-col* M
⟨*proof*⟩

lemma *col-prop-cond*: $(\bigwedge j.\ j <$ *dim-col* $M \Longrightarrow P$ (*col* $M\ j$)) $\Longrightarrow c \in$ *set* (*cols* M) $\Longrightarrow P\ c$
⟨*proof*⟩

Lemmas on the *map-mat* definition

lemma *row-map-mat*[*simp*]:
assumes $i <$ *dim-row* A **shows** *row* (*map-mat* $f\ A$) $i =$ *map-vec* f (*row* $A\ i$)
⟨*proof*⟩

lemma *map-vec-mat-rows*: *map* (*map-vec* f) (*rows* M) = *rows* ((*map-mat* f) M)
⟨*proof*⟩

lemma *map-vec-mat-cols*: *map* (*map-vec f*) (*cols M*) = *cols* ((*map-mat f*) *M*)
⟨*proof*⟩

lemma *map-mat-compose*: *map-mat f* (*map-mat g A*) = *map-mat* (*f* ∘ *g*) *A*
⟨*proof*⟩

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

Reasoning on sets and multisets of matrix elements

lemma *set-cols-carrier*: *A* ∈ *carrier-mat m n* ⟹ *v* ∈ *set* (*cols A*) ⟹ *v* ∈ *carrier-vec m*
⟨*proof*⟩

lemma *mset-cols-index-map*: *image-mset* (λ *j*. *col M j*) (*mset-set* {*0..*< *dim-col M*}) = *mset* (*cols M*)
⟨*proof*⟩

lemma *mset-rows-index-map*: *image-mset* (λ *i*. *row M i*) (*mset-set* {*0..*< *dim-row M*}) = *mset* (*rows M*)
⟨*proof*⟩

lemma *index-to-col-card-size-prop*:
assumes *i* < *dim-row M*
assumes ⋀ *j*. *j* < *dim-col M* ⟹ *P j* ⟷ *Q* (*col M j*)
shows *card* {*j* . *j* < *dim-col M* ∧ *P j*} = *size* {#*c* ∈# (*mset* (*cols M*)) . *Q c* #}
⟨*proof*⟩

lemma *index-to-row-card-size-prop*:
assumes *j* < *dim-col M*
assumes ⋀ *i*. *i* < *dim-row M* ⟹ *P i* ⟷ *Q* (*row M i*)
shows *card* {*i* . *i* < *dim-row M* ∧ *P i*} = *size* {#*r* ∈# (*mset* (*rows M*)) . *Q r* #}
⟨*proof*⟩

lemma *setv-row-subset-mat-elems*:
assumes *v* ∈ *set* (*rows M*)
shows *set*$_v$ *v* ⊆ *elements-mat M*
⟨*proof*⟩

lemma *setv-col-subset-mat-elems*:
assumes *v* ∈ *set* (*cols M*)
shows *set*$_v$ *v* ⊆ *elements-mat M*
⟨*proof*⟩

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 *dim-vec v2* $\leq$ *dim-vec v1*
shows *hom* (*v1* $\cdot$ *v2*) = vec_h *v1* $\cdot$ vec_h *v2*
⟨*proof*⟩

end

lemma *map-vec-vCons*: *vCons* (*f a*) (*map-vec f v*) = *map-vec f* (*vCons a v*)
⟨*proof*⟩

context *inj-zero-hom*
begin

lemma *vec-hom-zero-iff*[*simp*]: (*map-vec hom x* = 0_v *n*) = (*x* = 0_v *n*)
⟨*proof*⟩

lemma *mat-hom-inj*: *map-mat hom A* = *map-mat hom B* ⟹ *A* = *B*
⟨*proof*⟩

lemma *vec-hom-inj*: *map-vec hom v* = *map-vec hom w* ⟹ *v* = *w*
⟨*proof*⟩

lemma *vec-hom-set-distinct-iff*:
fixes *xs* :: *'a vec list*
shows *distinct xs* ⟷ *distinct* (*map* (*map-vec hom*) *xs*)
⟨*proof*⟩

lemma *vec-hom-mset*: *image-mset hom* (*vec-mset v*) = *vec-mset* (*map-vec hom v*)
⟨*proof*⟩

lemma *vec-hom-set*: *hom* ‘ set_v *v* = set_v (*map-vec hom v*)
⟨*proof*⟩

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* ⇒ *'b* **assumes** *injectivity-lim*: ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *f x* = *f y* ⟹ *x* = *y*
begin
lemma *eq-iff*[*simp*]: *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *f x* = *f y* ⟷ *x* = *y* ⟨*proof*⟩
lemma *inj-on-f*: *inj-on f A* ⟨*proof*⟩

end

sublocale *injective* $\subseteq$ *injective-lim Univ*
⟨*proof*⟩

context *injective-lim*
begin

lemma *mat-hom-inj-lim*:
assumes *elements-mat* $M \subseteq A$ **and** *elements-mat* $N \subseteq A$
shows *map-mat* $f\ M$ = *map-mat* $f\ N \Longrightarrow M = N$
⟨*proof*⟩

lemma *vec-hom-inj-lim*: **assumes** $set_v\ v \subseteq A$ $set_v\ w \subseteq A$
shows *map-vec* $f\ v$ = *map-vec* $f\ w \Longrightarrow v = w$
⟨*proof*⟩

lemma *lim-inj-hom-count-vec*:
assumes $set_v\ v \subseteq A$
assumes $x \in A$
shows *count-vec* $v\ x$ = *count-vec* (*map-vec* $f\ v$) ($f\ x$)
⟨*proof*⟩

lemma *vec-hom-lim-set-distinct-iff*:
fixes *xs* :: *'a vec list*
assumes $\bigwedge v$. $v \in set\ (xs) \Longrightarrow set_v\ v \subseteq A$
shows *distinct xs* $\longleftrightarrow$ *distinct* (*map* (*map-vec* f) *xs*)
⟨*proof*⟩

lemma *mat-rows-hom-lim-distinct-iff*:
assumes *elements-mat* $M \subseteq A$
shows *distinct* (*rows* M) $\longleftrightarrow$ *distinct* (*map* (*map-vec* f) (*rows* M))
⟨*proof*⟩

lemma *mat-cols-hom-lim-distinct-iff*:
assumes *elements-mat* $M \subseteq A$
shows *distinct* (*cols* M) $\longleftrightarrow$ *distinct* (*map* (*map-vec* f) (*cols* M))
⟨*proof*⟩

end

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

lemma *inj-0-iff*: $x \in \{0, 1\} \Longrightarrow hom\ x = 0 \longleftrightarrow x = 0$
⟨*proof*⟩

lemma *inj-1-iff*: $x \in \{0, 1\} \Longrightarrow hom\ x = 1 \longleftrightarrow x = 1$

⟨*proof*⟩

end

context *zero-neq-one*
begin

definition *of-zero-neq-one* :: *'b* :: {*zero-neq-one*} ⇒ *'a* **where**
of-zero-neq-one x ≡ *if* (*x* = *0*) *then 0 else 1*

lemma *of-zero-neq-one-1* [*simp*]: *of-zero-neq-one 1* = *1*
⟨*proof*⟩

lemma *of-zero-neq-one-0* [*simp*]: *of-zero-neq-one 0* = *0*
⟨*proof*⟩

lemma *of-zero-neq-one-0-iff*[*iff*]: *of-zero-neq-one x* = *0* ⟷ *x* = *0*
⟨*proof*⟩

lemma *of-zero-neq-one-lim-eq*: *x* ∈ {*0*, *1*} ⟹ *y* ∈ {*0*, *1*} ⟹ *of-zero-neq-one x* = *of-zero-neq-one y* ⟷ *x* = *y*
⟨*proof*⟩

end

interpretation *of-zero-hom*: *zero-hom-0 of-zero-neq-one*
⟨*proof*⟩

interpretation *of-injective-lim*: *injective-lim* {*0*, *1*} *of-zero-neq-one*
⟨*proof*⟩

interpretation *of-inj-on-01-hom*: *inj-on-01-hom of-zero-neq-one*
⟨*proof*⟩

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* ⇒ *'c* :: {*zero-neq-one*} *vec* **where**
lift-01-vec v ≡ *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* ⟹ (*lift-01-vec v*) $ *i* = *of-zero-neq-one* (*v* $ *i*)
⟨*proof*⟩

lemma *lift-01-vec-count*:
assumes set_v *v* ⊆ {*0*, *1*}
assumes *x* ∈ {*0*, *1*}
shows *count-vec v x* = *count-vec* (*lift-01-vec v*) (*of-zero-neq-one x*)
⟨*proof*⟩

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

lift-01-mat M ≡ *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* ⟹ *j* < *dim-col M* ⟹ (*lift-01-mat M*) $$ (*i*, *j*) = *of-zero-neq-one* (*M* $$ (*i*, *j*))
⟨*proof*⟩

lemma *lift-01-mat-carrier*: *lift-01-mat M* ∈ *carrier-mat* (*dim-row M*) (*dim-col M*)
⟨*proof*⟩

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* |∩| *b2* ≤ *card b1 b1* |∩| *b2* ≤ *card b2*
⟨*proof*⟩

lemma *inter-eq-blocks-eq-card*: *card b1* = *card b2* ⟹ *finite b1* ⟹ *finite b2* ⟹ *b1* |∩| *b2* = *card b1*
⟹ *b1* = *b2*
⟨*proof*⟩

lemma *inter-num-of-eq-blocks*: *b1* = *b2* ⟹ *b1* |∩| *b2* = *card b1*
⟨*proof*⟩

lemma *intersect-num-same-eq-size*[*simp*]: *bl* |∩| *bl* = *card bl*
⟨*proof*⟩

lemma *index-lt-rep-general*: *x* ∈ *ps* ⟹ *B index ps* ≤ *B rep x*
⟨*proof*⟩

context *incidence-system*
begin

lemma *block-size-alt*:

assumes $bl \in\# \mathcal{B}$
shows *card* bl = *card* $\{x \in \mathcal{V} \,.\, x \in bl\}$
⟨*proof*⟩

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

lemma *point-in-block-rep-min-iff*:
assumes $x \in \mathcal{V}$
shows $\exists\ bl\ .\ bl \in\# \mathcal{B} \wedge x \in bl \longleftrightarrow (\mathcal{B}\ rep\ x > 0)$
⟨*proof*⟩

lemma *points-inter-num-rep*:
assumes $b1 \in\# \mathcal{B}$ **and** $b2 \in\# \mathcal{B} - \{\#b1\#\}$
shows *card* $\{v \in \mathcal{V}\ .\ v \in b1 \wedge v \in b2\}$ = $b1\ |\cap|\ b2$
⟨*proof*⟩

Extensions on design operation lemmas

lemma *del-block-b*:
$bl \in\# \mathcal{B} \Longrightarrow$ *size* (*del-block* bl) = b − *1*
$bl \notin\# \mathcal{B} \Longrightarrow$ *size* (*del-block* bl) = b
⟨*proof*⟩

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

end

Extensions to properties of design sub types

context *finite-incidence-system*
begin

lemma *complete-block-size-eq-points*: $bl \in\# \mathcal{B} \Longrightarrow$ *card* bl = v $\Longrightarrow$ $bl = \mathcal{V}$
⟨*proof*⟩

lemma *complete-block-all-subsets*: $bl \in\# \mathcal{B} \Longrightarrow$ *card* bl = v $\Longrightarrow$ $ps \subseteq \mathcal{V} \Longrightarrow$ $ps \subseteq bl$
⟨*proof*⟩

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

end

context *design*
begin

lemma *block-num-rep-bound*: $\mathrm{b} \leq (\sum\ x \in \mathcal{V}.\ \mathcal{B}\ rep\ x)$
⟨*proof*⟩

end

context *proper-design*
begin

lemma *del-block-proper*:
  assumes $\mathrm{b} > 1$
  shows *proper-design* $\mathcal{V}$ (*del-block bl*)
⟨*proof*⟩

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*⟩

lemma *block-mset-distinct*: *distinct-mset* $\mathcal{B}$ ⟨*proof*⟩

end

context *constant-rep-design*
begin

lemma *index-lt-const-rep*:
  assumes *ps* $\subseteq$ $\mathcal{V}$
  assumes *ps* $\neq$ {}
  shows $\mathcal{B}$ *index ps* $\leq \mathrm{r}$
⟨*proof*⟩

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** *card ps* = t **and** $x \in ps$
⟨*proof*⟩

lemma *const-index-lt-rep*:
assumes $x \in \mathcal{V}$
shows $\Lambda_t \leq \mathcal{B}$ *rep x*
⟨*proof*⟩

end

context *pairwise-balance*
begin

lemma *index-zero-iff*: $\Lambda = 0 \longleftrightarrow (\forall\ bl \in\# \mathcal{B}\ .\ card\ bl = 1)$
⟨*proof*⟩

lemma *count-complete-lt-balance*: *count* $\mathcal{B}\ \mathcal{V} \leq \Lambda$
⟨*proof*⟩

lemma *eq-index-rep-imp-complete*:
assumes $\Lambda = \mathcal{B}$ *rep x*
assumes $x \in \mathcal{V}$
assumes $bl \in\# \mathcal{B}$
assumes $x \in bl$
shows *card bl* = v
⟨*proof*⟩

lemma *incomplete-index-strict-lt-rep*:
assumes $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow$ *incomplete-block bl*
assumes $x \in \mathcal{V}$
assumes $\Lambda > 0$
shows $\Lambda < \mathcal{B}$ *rep x*
⟨*proof*⟩

Construct new PBD's from existing PBD's

lemma *remove-complete-block-pbd*:
assumes b ≥ 2
assumes $bl \in\# \mathcal{B}$
assumes *card bl* = v
shows *pairwise-balance* $\mathcal{V}$ (*del-block bl*) $(\Lambda - 1)$
⟨*proof*⟩

lemma *remove-complete-block-pbd-alt*:
assumes b ≥ 2
assumes $bl \in\# \mathcal{B}$
assumes $bl = \mathcal{V}$
shows *pairwise-balance* $\mathcal{V}$ (*del-block bl*) $(\Lambda - 1)$

⟨*proof*⟩

lemma *b-gt-index*:b ≥ Λ
⟨*proof*⟩

lemma *remove-complete-blocks-set-pbd*:
assumes $x < \Lambda$
assumes *size* $A = x$
assumes $A \subset\# \mathcal{B}$
assumes $\bigwedge a.\ a \in\# A \Longrightarrow a = \mathcal{V}$
shows *pairwise-balance* $\mathcal{V}$ $(\mathcal{B} - A)$ $(\Lambda - x)$
⟨*proof*⟩

lemma *remove-all-complete-blocks-pbd*:
assumes *count* $\mathcal{B}$ $\mathcal{V} < \Lambda$
shows *pairwise-balance* $\mathcal{V}$ (*removeAll-mset* $\mathcal{V}$ $\mathcal{B}$) $(\Lambda - (count\ \mathcal{B}\ \mathcal{V}))$ (**is** *pairwise-balance* $\mathcal{V}$ *?B ?*Λ)
⟨*proof*⟩

end

context *bibd*
begin
lemma *symmetric-bibdIII*: r = k ⟹ *symmetric-bibd* $\mathcal{V}$ $\mathcal{B}$ k Λ
⟨*proof*⟩
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*⟩

end

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

Const Intersect Design

This is the dual of a balanced design, and used extensively in the re-

maining formalisation

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

sublocale *symmetric-bibd* $\subseteq$ *const-intersect-design* $\mathcal{V}$ $\mathcal{B}$ Λ
  ⟨*proof*⟩

context *const-intersect-design*
begin

lemma *inter-num-le-block-size*:
  assumes *bl* $\in\#$ $\mathcal{B}$
  assumes b $\geq$ *2*
  shows m $\leq$ *card bl*
⟨*proof*⟩

lemma *const-inter-multiplicity-one*:
  assumes *bl* $\in\#$ $\mathcal{B}$
  assumes m $<$ *card bl*
  shows *multiplicity bl* = *1*
⟨*proof*⟩

lemma *mult-blocks-const-inter*:
  assumes *bl* $\in\#$ $\mathcal{B}$
  assumes *multiplicity bl* $>$ *1*
  assumes b $\geq$ *2*
  shows m = *card bl*
⟨*proof*⟩

lemma *simple-const-inter-block-size*: ($\bigwedge$ *bl*. *bl* $\in\#$ $\mathcal{B}$ $\Longrightarrow$ m $<$ *card bl*) $\Longrightarrow$ *simple-design* $\mathcal{V}$ $\mathcal{B}$
  ⟨*proof*⟩

lemma *simple-const-inter-iff*:
  assumes b $\geq$ *2*
  shows *size* {#*bl* $\in\#$ $\mathcal{B}$. *card bl* = m #} $\leq$ *1* $\longleftrightarrow$ *simple-design* $\mathcal{V}$ $\mathcal{B}$
⟨*proof*⟩

lemma *empty-inter-implies-rep-one*:
  assumes m = *0*
  assumes *x* $\in$ $\mathcal{V}$
  shows $\mathcal{B}$ *rep x* $\leq$ *1*
⟨*proof*⟩

lemma *empty-inter-implies-b-lt-v*:
  assumes m = *0*
  shows b $\leq$ v
⟨*proof*⟩

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 ith position if point i is in that block

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

lemma *inc-vec-one-zero-elems*: set_v (*inc-vec-of Vs bl*) ⊆ {*0*, *1*}
⟨*proof*⟩

lemma *finite-inc-vec-elems*: *finite* (set_v (*inc-vec-of Vs bl*))
⟨*proof*⟩

lemma *inc-vec-elems-max-two*: *card* (set_v (*inc-vec-of Vs bl*)) ≤ *2*
⟨*proof*⟩

lemma *inc-vec-dim*: *dim-vec* (*inc-vec-of Vs bl*) = *length Vs*
⟨*proof*⟩

lemma *inc-vec-index*: *i* < *length Vs* ⟹ *inc-vec-of Vs bl* $ *i* = (*if* (*Vs* ! *i*) ∈ *bl then 1 else 0*)
⟨*proof*⟩

lemma *inc-vec-index-one-iff*: *i* < *length Vs* ⟹ *inc-vec-of Vs bl* $ *i* = *1* ⟷ *Vs* ! *i* ∈ *bl*
⟨*proof*⟩

lemma *inc-vec-index-zero-iff*: *i* < *length Vs* ⟹ *inc-vec-of Vs bl* $ *i* = *0* ⟷ *Vs* ! *i* ∉ *bl*
⟨*proof*⟩

lemma *inc-vec-of-bij-betw*:
assumes *inj-on f* (*set Vs*)
assumes *bl* ⊆ (*set Vs*)
shows *inc-vec-of Vs bl* = *inc-vec-of* (*map f Vs*) (*f* ‘ *bl*)
⟨*proof*⟩

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* ⇒ *'a set list* ⇒ (*'b* :: {*ring-1*}) *mat* **where**
inc-mat-of Vs Bs ≡ *mat* (*length Vs*) (*length Bs*) (λ (*i*,*j*) . *if* (*Vs* ! *i*) ∈ (*Bs* ! *j*) *then 1 else 0*)

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

lemma *inc-mat-one-zero-elems*: *elements-mat* (*inc-mat-of Vs Bs*) ⊆ {*0*, *1*}
⟨*proof*⟩

lemma *fin-incidence-mat-elems*: *finite* (*elements-mat* (*inc-mat-of Vs Bs*))
⟨*proof*⟩

lemma *inc-matrix-elems-max-two*: *card* (*elements-mat* (*inc-mat-of Vs Bs*)) ≤ *2*
⟨*proof*⟩

lemma *inc-mat-of-index* [*simp*]: *i* < *dim-row* (*inc-mat-of Vs Bs*) ⟹ *j* < *dim-col* (*inc-mat-of Vs Bs*) ⟹
inc-mat-of Vs Bs $$ (*i*, *j*) = (*if* (*Vs* ! *i*) ∈ (*Bs* ! *j*) *then 1 else 0*)
⟨*proof*⟩

lemma *inc-mat-dim-row*: *dim-row* (*inc-mat-of Vs Bs*) = *length Vs*
⟨*proof*⟩

lemma *inc-mat-dim-vec-row*: *dim-vec* (*row* (*inc-mat-of Vs Bs*) *i*) = *length Bs*
⟨*proof*⟩

lemma *inc-mat-dim-col*: *dim-col* (*inc-mat-of Vs Bs*) = *length Bs*
⟨*proof*⟩

lemma *inc-mat-dim-vec-col*: *dim-vec* (*col* (*inc-mat-of Vs Bs*) *i*) = *length Vs*
⟨*proof*⟩

lemma *inc-matrix-point-in-block-one*: *i* < *length Vs* ⟹ *j* < *length Bs* ⟹ *Vs* ! *i* ∈ *Bs* ! *j*
⟹ (*inc-mat-of Vs Bs*) $$ (*i*, *j*) = *1*

⟨*proof*⟩

lemma *inc-matrix-point-not-in-block-zero*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow Vs\ !\ i \notin Bs\ !\ j \Longrightarrow$
$(inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 0$
⟨*proof*⟩

lemma *inc-matrix-point-in-block*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 1 \Longrightarrow Vs\ !\ i \in Bs\ !\ j$
⟨*proof*⟩

lemma *inc-matrix-point-not-in-block*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 0 \Longrightarrow Vs\ !\ i \notin Bs\ !\ j$
⟨*proof*⟩

lemma *inc-matrix-point-not-in-block-iff*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 0 \longleftrightarrow Vs\ !\ i \notin Bs\ !\ j$
⟨*proof*⟩

lemma *inc-matrix-point-in-block-iff*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 1 \longleftrightarrow Vs\ !\ i \in Bs\ !\ j$
⟨*proof*⟩

lemma *inc-matrix-subset-implies-one*:
assumes $I \subseteq \{..< length\ Vs\}$
assumes $j < length\ Bs$
assumes (!) $Vs\ `\ I \subseteq Bs\ !\ j$
assumes $i \in I$
shows $(inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 1$
⟨*proof*⟩

lemma *inc-matrix-one-implies-membership*: $I \subseteq \{..< length\ Vs\} \Longrightarrow j < length\ Bs \Longrightarrow$
$(\bigwedge i.\ i \in I \Longrightarrow (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 1) \Longrightarrow i \in I \Longrightarrow Vs\ !\ i \in Bs\ !\ j$
⟨*proof*⟩

lemma *inc-matrix-elems-one-zero*: $i < length\ Vs \Longrightarrow j < length\ Bs \Longrightarrow$
$(inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 0 \vee (inc\text{-}mat\text{-}of\ Vs\ Bs)\ \$\$\ (i, j) = 1$
⟨*proof*⟩

Reasoning on Rows/Columns of the incidence matrix

lemma *inc-mat-col-def*: $j < length\ Bs \Longrightarrow i < length\ Vs \Longrightarrow$
$(col\ (inc\text{-}mat\text{-}of\ Vs\ Bs)\ j)\ \$\ i = (if\ (Vs\ !\ i \in Bs\ !\ j)\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *inc-mat-col-list-map-elem*: $j < length\ Bs \Longrightarrow i < length\ Vs \Longrightarrow$
$col\ (inc\text{-}mat\text{-}of\ Vs\ Bs)\ j\ \$\ i = map\text{-}vec\ (\lambda\ x\ .\ if\ (x \in (Bs\ !\ j))\ then\ 1\ else\ 0)\ (vec\text{-}of\text{-}list\ Vs)\ \$\ i$

⟨*proof*⟩

lemma *inc-mat-col-list-map*: $j < \text{length } Bs \Longrightarrow$
$\text{col } (\text{inc-mat-of } Vs\ Bs)\ j = \text{map-vec } (\lambda\ x\ .\ \text{if } (x \in (Bs\ !\ j))\ \text{then } 1\ \text{else } 0)$
$(\text{vec-of-list } Vs)$
⟨*proof*⟩

lemma *inc-mat-row-def*: $j < \text{length } Bs \Longrightarrow i < \text{length } Vs \Longrightarrow$
$(\text{row } (\text{inc-mat-of } Vs\ Bs)\ i)\ \$\ j = (\text{if } (Vs\ !\ i \in Bs\ !\ j)\ \text{then } 1\ \text{else } 0)$
⟨*proof*⟩

lemma *inc-mat-row-list-map-elem*: $j < \text{length } Bs \Longrightarrow i < \text{length } Vs \Longrightarrow$
$\text{row } (\text{inc-mat-of } Vs\ Bs)\ i\ \$\ j = \text{map-vec } (\lambda\ bl\ .\ \text{if } ((Vs\ !\ i) \in bl)\ \text{then } 1\ \text{else } 0)$
$(\text{vec-of-list } Bs)\ \$\ j$
⟨*proof*⟩

lemma *inc-mat-row-list-map*: $i < \text{length } Vs \Longrightarrow$
$\text{row } (\text{inc-mat-of } Vs\ Bs)\ i = \text{map-vec } (\lambda\ bl\ .\ \text{if } ((Vs\ !\ i) \in bl)\ \text{then } 1\ \text{else } 0)$
$(\text{vec-of-list } Bs)$
⟨*proof*⟩

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

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

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*⟩

lemma *inc-mat-of-bij-betw*:
assumes *inj-on f* (*set Vs*)
assumes $\bigwedge bl\ .\ bl \in (\text{set } Bs) \Longrightarrow bl \subseteq (\text{set } Vs)$
shows *inc-mat-of Vs Bs* = *inc-mat-of* (*map f Vs*) (*map* ((`) *f*) *Bs*)
⟨*proof*⟩

Definitions for the incidence matrix representation of common incidence system properties

definition *non-empty-col* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *nat* ⇒ *bool* **where**
non-empty-col M j ≡ ∃ *k*. *k* ≠ *0* ∧ *k* ∈$ *col M j*

definition *proper-inc-mat* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *bool* **where**
proper-inc-mat M ≡ (*dim-row M* > *0* ∧ *dim-col M* > *0*)

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

definition *mat-rep-num* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *nat* ⇒ *nat* **where**
mat-rep-num M i ≡ *count-vec* (*row M i*) *1*

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

definition *mat-point-index* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *nat set* ⇒ *nat* **where** *mat-point-index M I* ≡ *card* {*j* . *j* < *dim-col M* ∧ (∀ *i* ∈ *I*. *M* $$ (*i*, *j*) = *1*)}

definition *mat-inter-num* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where** *mat-inter-num M j1 j2* ≡ *card* {*i* . *i* < *dim-row M* ∧ *M* $$ (*i*, *j1*) = *1* ∧ *M* $$ (*i*, *j2*) = *1*}

Matrix version of the block size property

definition *mat-block-size* :: ('*a* :: {*zero-neq-one*}) *mat* ⇒ *nat* ⇒ *nat* **where** *mat-block-size M j* ≡ *count-vec* (*col M j*) *1*

lemma *non-empty-col-obtains*:
assumes *non-empty-col M j*
obtains *i* **where** *i* < *dim-row M* **and** (*col M j*) $ *i* ≠ *0*
⟨*proof*⟩

lemma *non-empty-col-alt-def*:
assumes *j* < *dim-col M*
shows *non-empty-col M j* ⟷ (∃ *i*. *i* < *dim-row M* ∧ *M* $$ (*i*, *j*) ≠ *0*)
⟨*proof*⟩

lemma *proper-inc-mat-map*: *proper-inc-mat M* ⟹ *proper-inc-mat* (*map-mat f M*)
⟨*proof*⟩

lemma *mat-point-index-alt*: *mat-point-index M I* = *card* {*j* ∈ {*0*..<*dim-col M*} . (∀ *i* ∈ *I* . *M* $$(*i*, *j*) = *1*)}
⟨*proof*⟩

lemma *mat-block-size-sum-alt*:
fixes *M* :: '*a* :: {*ring-1*} *mat*
shows *elements-mat M* ⊆ {*0*, *1*} ⟹ *j* < *dim-col M* ⟹ *of-nat* (*mat-block-size M j*) = *sum-vec* (*col M j*)
⟨*proof*⟩

lemma *mat-rep-num-sum-alt*:
fixes *M* :: '*a* :: {*ring-1*} *mat*
shows *elements-mat M* ⊆ {*0*, *1*} ⟹ *i* < *dim-row M* ⟹ *of-nat* (*mat-rep-num M i*) = *sum-vec* (*row M i*)
⟨*proof*⟩

lemma *mat-point-index-two-alt*:
assumes *i1* < *dim-row M*
assumes *i2* < *dim-row M*
shows *mat-point-index M* {*i1*, *i2*} = *card* {*j* . *j* < *dim-col M* ∧ *M* $$(*i1*, *j*) = *1* ∧ *M* $$ (*i2*, *j*) = *1*}
⟨*proof*⟩

Transpose symmetries

lemma *trans-mat-rep-block-size-sym*: *j* < *dim-col M* ⟹ *mat-block-size M j* = *mat-rep-num* M^T *j*

$i < dim\text{-}row\ M \Longrightarrow mat\text{-}rep\text{-}num\ M\ i = mat\text{-}block\text{-}size\ M^T\ i$
⟨*proof*⟩

lemma *trans-mat-point-index-inter-sym*:
$i1 < dim\text{-}row\ M \Longrightarrow i2 < dim\text{-}row\ M \Longrightarrow mat\text{-}point\text{-}index\ M\ \{i1, i2\} = mat\text{-}inter\text{-}num\ M^T\ i1\ i2$
$j1 < dim\text{-}col\ M \Longrightarrow j2 < dim\text{-}col\ M \Longrightarrow mat\text{-}inter\text{-}num\ M\ j1\ j2 = mat\text{-}point\text{-}index\ M^T\ \{j1, j2\}$
⟨*proof*⟩

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*: *elements-mat M* ⊆ {*0*, *1*}
begin

Row and Column Properties of the Matrix

lemma *row-elems-ss01*:$i < dim\text{-}row\ M \Longrightarrow vec\text{-}set\ (row\ M\ i) \subseteq \{0, 1\}$
⟨*proof*⟩

lemma *col-elems-ss01*:
assumes $j < dim\text{-}col\ M$
shows $vec\text{-}set\ (col\ M\ j) \subseteq \{0, 1\}$
⟨*proof*⟩

lemma *col-nth-0-or-1-iff*:
assumes $j < dim\text{-}col\ M$
assumes $i < dim\text{-}row\ M$
shows $col\ M\ j\ \$\ i = 0 \longleftrightarrow col\ M\ j\ \$\ i \neq 1$
⟨*proof*⟩

lemma *row-nth-0-or-1-iff*:
assumes $j < dim\text{-}col\ M$
assumes $i < dim\text{-}row\ M$
shows $row\ M\ i\ \$\ j = 0 \longleftrightarrow row\ M\ i\ \$\ j \neq 1$
⟨*proof*⟩

lemma *transpose-entries*: *elements-mat* $(M^T) \subseteq \{0, 1\}$
⟨*proof*⟩

lemma *M-not-zero-simp*: $j < dim\text{-}col\ M \Longrightarrow i < dim\text{-}row\ M \Longrightarrow M\ \$\$\ (i, j) \neq 0 \Longrightarrow M\ \$\$\ (i, j) = 1$
⟨*proof*⟩

lemma *M-not-one-simp*: $j < \textit{dim-col}\ M \Longrightarrow i < \textit{dim-row}\ M \Longrightarrow M\ \$\$\ (i, j) \neq 1 \Longrightarrow M\ \$\$\ (i, j) = 0$
⟨*proof*⟩

Definition for mapping a column to a block

definition *map-col-to-block* :: *'a* :: {*zero-neq-one*} *vec* ⇒ *nat set* **where** *map-col-to-block c* ≡ { *i* ∈ {..<*dim-vec c*} . *c* $ *i* = *1*}

lemma *map-col-to-block-alt*: *map-col-to-block c* = {*i* . *i* < *dim-vec c* ∧ *c*$ *i* = *1*}
⟨*proof*⟩

lemma *map-col-to-block-elem*: *i* < *dim-vec c* ⟹ *i* ∈ *map-col-to-block c* ⟷ *c* $ *i* = *1*
⟨*proof*⟩

lemma *in-map-col-valid-index*: *i* ∈ *map-col-to-block c* ⟹ *i* < *dim-vec c*
⟨*proof*⟩

lemma *map-col-to-block-size*: *j* < *dim-col M* ⟹ *card* (*map-col-to-block* (*col M j*)) = *mat-block-size M j*
⟨*proof*⟩

lemma *in-map-col-valid-index-M*: *j* < *dim-col M* ⟹ *i* ∈ *map-col-to-block* (*col M j*) ⟹ *i* < *dim-row M*
⟨*proof*⟩

lemma *map-col-to-block-elem-not*: *c* ∈ *set* (*cols M*) ⟹ *i* < *dim-vec c* ⟹ *i* ∉ *map-col-to-block c* ⟷ *c* $ *i* = *0*
⟨*proof*⟩

lemma *obtain-block-index-map-block-set*:
assumes *bl* ∈# {# *map-col-to-block c* . *c* ∈# *mset* (*cols M*)#}
obtains *j* **where** *j* < *dim-col M* **and** *bl* = *map-col-to-block* (*col M j*)
⟨*proof*⟩

lemma *mat-ord-inc-sys-point*[*simp*]: *x* < *dim-row M* ⟹ [*0*..<(*dim-row M*)] ! *x* = *x*
⟨*proof*⟩

lemma *mat-ord-inc-sys-block*[*simp*]: *j* < *dim-col M* ⟹
(*map* (*map-col-to-block*) (*cols M*)) ! *j* = *map-col-to-block* (*col M j*)
⟨*proof*⟩

lemma *ordered-to-mset-col-blocks*:
{# *map-col-to-block c* . *c* ∈# *mset* (*cols M*)#} = *mset* (*map* (*map-col-to-block*) (*cols M*))
⟨*proof*⟩

Lemmas on incidence matrix properties

lemma *non-empty-col-01*:

assumes $j < \textit{dim-col}\ M$
shows $\textit{non-empty-col}\ M\ j \longleftrightarrow 1 \in\$\ \textit{col}\ M\ j$
⟨*proof*⟩

lemma *mat-rep-num-alt*:
assumes $i < \textit{dim-row}\ M$
shows $\textit{mat-rep-num}\ M\ i = \textit{card}\ \{j\ .\ j < \textit{dim-col}\ M \wedge M\ \$\$\ (i, j) = 1\}$
⟨*proof*⟩

lemma *mat-rep-num-alt-col*: $i < \textit{dim-row}\ M \Longrightarrow \textit{mat-rep-num}\ M\ i = \textit{size}\ \{\#c \in\#\ (\textit{mset}\ (\textit{cols}\ M))\ .\ c\ \$\ i = 1\#\}$
⟨*proof*⟩

A zero one matrix is an incidence system

lemma *map-col-to-block-wf*: $\bigwedge c.\ c \in \textit{set}\ (\textit{cols}\ M) \Longrightarrow \textit{map-col-to-block}\ c \subseteq \{0..<\textit{dim-row}\ M\}$
⟨*proof*⟩

lemma *one-implies-block-nempty*: $j < \textit{dim-col}\ M \Longrightarrow 1 \in\$\ (\textit{col}\ M\ j) \Longrightarrow \textit{map-col-to-block}\ (\textit{col}\ M\ j) \neq \{\}$
⟨*proof*⟩

interpretation *incidence-sys*: *incidence-system* $\{0..<\textit{dim-row}\ M\}$
$\{\#\ \textit{map-col-to-block}\ c\ .\ c \in\#\ \textit{mset}\ (\textit{cols}\ M)\#\}$
⟨*proof*⟩

interpretation *fin-incidence-sys*: *finite-incidence-system* $\{0..<\textit{dim-row}\ M\}$
$\{\#\ \textit{map-col-to-block}\ c\ .\ c \in\#\ \textit{mset}\ (\textit{cols}\ M)\#\}$
⟨*proof*⟩

lemma *block-nempty-implies-all-zeros*: $j < \textit{dim-col}\ M \Longrightarrow \textit{map-col-to-block}\ (\textit{col}\ M\ j) = \{\} \Longrightarrow$
$i < \textit{dim-row}\ M \Longrightarrow \textit{col}\ M\ j\ \$\ i = 0$
⟨*proof*⟩

lemma *block-nempty-implies-no-one*: $j < \textit{dim-col}\ M \Longrightarrow \textit{map-col-to-block}\ (\textit{col}\ M\ j) = \{\} \Longrightarrow \neg\ (1 \in\$\ (\textit{col}\ M\ j))$
⟨*proof*⟩

lemma *mat-is-design*:
assumes $\bigwedge j.\ j < \textit{dim-col}\ M \Longrightarrow 1 \in\$\ (\textit{col}\ M\ j)$
shows $\textit{design}\ \{0..<\textit{dim-row}\ M\}\ \{\#\ \textit{map-col-to-block}\ c\ .\ c \in\#\ \textit{mset}\ (\textit{cols}\ M)\#\}$
⟨*proof*⟩

lemma *mat-is-proper-design*:
assumes $\bigwedge j.\ j < \textit{dim-col}\ M \Longrightarrow 1 \in\$\ (\textit{col}\ M\ j)$
assumes $\textit{dim-col}\ M > 0$
shows $\textit{proper-design}\ \{0..<\textit{dim-row}\ M\}\ \{\#\ \textit{map-col-to-block}\ c\ .\ c \in\#\ \textit{mset}\ (\textit{cols}\ M)\#\}$

⟨*proof*⟩

Show the 01 injective function preserves system properties

lemma *inj-on-01-hom-index*:
assumes *inj-on-01-hom f*
assumes *i* < *dim-row M j* < *dim-col M*
shows *M* $$ (*i*, *j*) = *1* ⟷ (*map-mat f M*) $$ (*i*, *j*) = *1*
and *M* $$ (*i*, *j*) = *0* ⟷ (*map-mat f M*) $$ (*i*, *j*) = *0*
⟨*proof*⟩

lemma *preserve-non-empty*:
assumes *inj-on-01-hom f*
assumes *j* < *dim-col M*
shows *non-empty-col M j* ⟷ *non-empty-col* (*map-mat f M*) *j*
⟨*proof*⟩

lemma *preserve-mat-rep-num*:
assumes *inj-on-01-hom f*
assumes *i* < *dim-row M*
shows *mat-rep-num M i* = *mat-rep-num* (*map-mat f M*) *i*
⟨*proof*⟩

lemma *preserve-mat-block-size*:
assumes *inj-on-01-hom f*
assumes *j* < *dim-col M*
shows *mat-block-size M j* = *mat-block-size* (*map-mat f M*) *j*
⟨*proof*⟩

lemma *preserve-mat-point-index*:
assumes *inj-on-01-hom f*
assumes ⋀ *i*. *i* ∈ *I* ⟹ *i* < *dim-row M*
shows *mat-point-index M I* = *mat-point-index* (*map-mat f M*) *I*
⟨*proof*⟩

lemma *preserve-mat-inter-num*:
assumes *inj-on-01-hom f*
assumes *j1* < *dim-col M j2* < *dim-col M*
shows *mat-inter-num M j1 j2* = *mat-inter-num* (*map-mat f M*) *j1 j2*
⟨*proof*⟩

lemma *lift-mat-01-index-iff*:
i < *dim-row M* ⟹ *j* < *dim-col M* ⟹ (*lift-01-mat M*) $$ (*i*, *j*) = *0* ⟷ *M* $$ (*i*, *j*) = *0*
i < *dim-row M* ⟹ *j* < *dim-col M* ⟹ (*lift-01-mat M*) $$ (*i*, *j*) = *1* ⟷ *M* $$ (*i*, *j*) = *1*
⟨*proof*⟩

lemma *lift-mat-elems*: *elements-mat* (*lift-01-mat M*) ⊆ {*0*, *1*}

⟨*proof*⟩

lemma *lift-mat-is-0-1*: *zero-one-matrix* (*lift-01-mat M*)
⟨*proof*⟩

lemma *lift-01-mat-distinct-cols*: *distinct* (*cols M*) ⟹ *distinct* (*cols* (*lift-01-mat M*))
⟨*proof*⟩

end

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

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

lemma *map-col-block-eq*:
assumes *c* ∈ *set*(*cols M*)
shows *inc-vec-of* [*0..<dim-vec c*] (*map-col-to-block c*) = *c*
⟨*proof*⟩

lemma *inc-mat-of-map-rev*: *inc-mat-of* [*0..<dim-row M*] (*map map-col-to-block* (*cols M*)) = *M*
⟨*proof*⟩

lemma *M-index-square-itself*: *j* < *dim-col M* ⟹ *i* < *dim-row M* ⟹ (*M* $$ (*i*, *j*))^2 = *M* $$ (*i*, *j*)
⟨*proof*⟩

lemma *M-col-index-square-itself*: *j* < *dim-col M* ⟹ *i* < *dim-row M* ⟹ ((*col M j*) $ *i*)^2 = (*col M j*) $ *i*
⟨*proof*⟩

Scalar Prod Alternative definitions for matrix properties

lemma *scalar-prod-inc-vec-block-size-mat*:
assumes *j* < *dim-col M*
shows (*col M j*) ⋅ (*col M j*) = *of-nat* (*mat-block-size M j*)
⟨*proof*⟩

lemma *scalar-prod-inc-vec-mat-inter-num*:
assumes *j1* < *dim-col M j2* < *dim-col M*
shows (*col M j1*) ⋅ (*col M j2*) = *of-nat* (*mat-inter-num M j1 j2*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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 list* **and** $\mathcal{B}s$ :: *'a set list*
  assumes *wf-list*: $b \in\# (mset\ \mathcal{B}s) \Longrightarrow b \subseteq set\ \mathcal{V}s$
  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* $\mathcal{V}s$ *mset* $\mathcal{B}s$
  ⟨*proof*⟩

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*⟩

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*⟩

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* $\mathcal{V}$
  ⟨*proof*⟩

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

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

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

lemma *blocks-list-empty-iff*: $\mathcal{B}s = [] \longleftrightarrow \mathcal{B} = \{\#\}$
  ⟨*proof*⟩

lemma *blocks-list-nempty*: *proper-design* $\mathcal{V}$ $\mathcal{B} \Longrightarrow \mathcal{B}s \neq []$
⟨*proof*⟩

lemma *points-list-nempty*: *proper-design* $\mathcal{V}$ $\mathcal{B} \Longrightarrow \mathcal{V}s \neq []$
⟨*proof*⟩

lemma *points-list-length*: *length* $\mathcal{V}s$ = v
⟨*proof*⟩

lemma *blocks-list-length*: *length* $\mathcal{B}s$ = b
⟨*proof*⟩

lemma *valid-points-index*: $i <$ v $\Longrightarrow \mathcal{V}s\ !\ i \in \mathcal{V}$
⟨*proof*⟩

lemma *valid-points-index-cons*: $x \in \mathcal{V} \Longrightarrow \exists\ i.\ \mathcal{V}s\ !\ i = x \wedge i <$ v
⟨*proof*⟩

lemma *valid-points-index-obtains*:
assumes $x \in \mathcal{V}$
obtains i **where** $\mathcal{V}s\ !\ i = x \wedge i <$ v
⟨*proof*⟩

lemma *valid-blocks-index*: $j <$ b $\Longrightarrow \mathcal{B}s\ !\ j \in\# \mathcal{B}$
⟨*proof*⟩

lemma *valid-blocks-index-cons*: $bl \in\# \mathcal{B} \Longrightarrow \exists\ j\ .\ \mathcal{B}s\ !\ j = bl \wedge j <$ b
⟨*proof*⟩

lemma *valid-blocks-index-obtains*:
assumes $bl \in\# \mathcal{B}$
obtains j **where** $\mathcal{B}s\ !\ j = bl \wedge j <$ b
⟨*proof*⟩

lemma *block-points-valid-point-index*:
assumes $bl \in\# \mathcal{B}$ $x \in bl$
obtains i **where** $i <$ *length* $\mathcal{V}s \wedge \mathcal{V}s\ !\ i = x$
⟨*proof*⟩

lemma *points-set-index-img*: $\mathcal{V}$ = *image*($\lambda\ i\ .\ (\mathcal{V}s\ !\ i)$) ({..<v})
⟨*proof*⟩

lemma *blocks-mset-image*: $\mathcal{B}$ = *image-mset* ($\lambda\ i\ .\ (\mathcal{B}s\ !\ i)$) (*mset-set* {..<b})
⟨*proof*⟩

lemma *incidence-cond-indexed*[*simp*]: $i <$ v $\Longrightarrow j <$ b $\Longrightarrow$ *incident* ($\mathcal{V}s\ !\ i$) ($\mathcal{B}s\ !\ j$) $\longleftrightarrow$ ($\mathcal{V}s\ !\ i$) $\in$ ($\mathcal{B}s\ !\ j$)
⟨*proof*⟩

lemma *bij-betw-points-index*: *bij-betw* (λ *i*. $\mathcal{V}s$! *i*) {*0*..<v} $\mathcal{V}$
⟨*proof*⟩

Some lemmas on cardinality due to different set descriptor filters

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

lemma *card-block-points-filter*:
assumes *j* < b
shows *card* ($\mathcal{B}s$! *j*) = *card* {*i* ∈ {*0*..<v} . ($\mathcal{V}s$! *i*) ∈ ($\mathcal{B}s$! *j*)}
⟨*proof*⟩

lemma *obtains-two-diff-block-indexes*:
assumes *j1* < b
assumes *j2* < b
assumes *j1* ≠ *j2*
assumes b ≥ *2*
obtains *bl1 bl2* **where** *bl1* ∈# $\mathcal{B}$ **and** $\mathcal{B}s$! *j1* = *bl1* **and** *bl2* ∈# $\mathcal{B}$ − {#*bl1*#} **and** $\mathcal{B}s$! *j2* = *bl2*
⟨*proof*⟩

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

lemma *blocks-index-ne-belong*:
assumes *i1* < *length* $\mathcal{B}s$
assumes *i2* < *length* $\mathcal{B}s$
assumes *i1* ≠ *i2*
shows $\mathcal{B}s$! *i2* ∈# $\mathcal{B}$ − {#($\mathcal{B}s$! *i1*)#}
⟨*proof*⟩

lemma *inter-num-points-filter-def*:
assumes *j1* < b *j2* < b *j1* ≠ *j2*
shows *card* {*x* ∈ {*0*..<v} . (($\mathcal{V}s$! *x*) ∈ ($\mathcal{B}s$! *j1*) ∧ ($\mathcal{V}s$! *x*) ∈ ($\mathcal{B}s$! *j2*)) } = ($\mathcal{B}s$! *j1*) |∩| ($\mathcal{B}s$! *j2*)
⟨*proof*⟩

Define an incidence matrix for this ordering of an incidence system

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

sublocale *zero-one-matrix-ring-1 N*
⟨*proof*⟩

lemma *N-alt-def-dim*: *N* = *mat* v b (λ (*i*,*j*) . *if* (*incident* ($\mathcal{V}s$! *i*) ($\mathcal{B}s$! *j*)) *then* *1* *else* *0*)

⟨*proof*⟩

Matrix Dimension related lemmas

lemma *N-carrier-mat*: $N \in$ *carrier-mat* v b
⟨*proof*⟩

lemma *dim-row-is-v*[*simp*]: *dim-row* N = v
⟨*proof*⟩

lemma *dim-col-is-b*[*simp*]: *dim-col* N = b
⟨*proof*⟩

lemma *dim-vec-row-N*: *dim-vec* (*row* N i) = b
⟨*proof*⟩

lemma *dim-vec-col-N*: *dim-vec* (*col* N i) = v ⟨*proof*⟩

lemma *dim-vec-N-col*:
assumes $j <$ b
shows *dim-vec* (*cols* N ! j) = v
⟨*proof*⟩

lemma *N-carrier-mat-01-lift*: *lift-01-mat* $N \in$ *carrier-mat* v b
⟨*proof*⟩

Transpose properties

lemma *transpose-N-mult-dim*: *dim-row* (N ∗ N^T) = v *dim-col* (N ∗ N^T) = v
⟨*proof*⟩

lemma *N-trans-index-val*: $i <$ *dim-col* $N \Longrightarrow j <$ *dim-row* $N \Longrightarrow$
N^T $$ (i, j) = (*if* ($\mathcal{V}s$! j) ∈ ($\mathcal{B}s$! i) *then 1 else 0*)
⟨*proof*⟩

Matrix element and index related lemmas

lemma *mat-row-elems*: $i <$ v $\Longrightarrow$ *vec-set* (*row* N i) ⊆ {*0*, *1*}
⟨*proof*⟩

lemma *mat-col-elems*: $j <$ b $\Longrightarrow$ *vec-set* (*col* N j) ⊆ {*0*, *1*}
⟨*proof*⟩

lemma *matrix-elems-one-zero*: $i <$ v $\Longrightarrow j <$ b $\Longrightarrow$ N $$ (i, j) = *0* ∨ N $$ (i, j) = *1*
⟨*proof*⟩

lemma *matrix-point-in-block-one*: $i <$ v $\Longrightarrow j <$ b $\Longrightarrow$ ($\mathcal{V}s$! i)∈ ($\mathcal{B}s$! j) $\Longrightarrow N$ $$ (i, j) = *1*
⟨*proof*⟩

lemma *matrix-point-not-in-block-zero*: $i <$ v $\Longrightarrow j <$ b $\Longrightarrow \mathcal{V}s$! $i \notin \mathcal{B}s$! $j \Longrightarrow N$ $$ (i, j) = *0*

⟨*proof*⟩

lemma *matrix-point-in-block*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow N\ \$\$\ (i, j) = 1 \Longrightarrow \mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j$
⟨*proof*⟩

lemma *matrix-point-not-in-block*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow N\ \$\$\ (i, j) = 0 \Longrightarrow \mathcal{V}s\ !\ i \notin \mathcal{B}s\ !\ j$
⟨*proof*⟩

lemma *matrix-point-not-in-block-iff*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow N\ \$\$\ (i, j) = 0 \longleftrightarrow \mathcal{V}s\ !\ i \notin \mathcal{B}s\ !\ j$
⟨*proof*⟩

lemma *matrix-point-in-block-iff*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow N\ \$\$\ (i, j) = 1 \longleftrightarrow \mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j$
⟨*proof*⟩

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

lemma *matrix-one-implies-membership*:
$I \subseteq \{..< \mathrm{v}\} \Longrightarrow j < \mathit{size}\ \mathcal{B} \Longrightarrow \forall\, i{\in}I.\ N\ \$\$\ (i, j) = 1 \Longrightarrow i \in I \Longrightarrow \mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j$
⟨*proof*⟩

Incidence Vector's of Incidence Matrix columns

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

lemma *inc-vec-eq-iff-blocks*:
assumes $bl \in\# \mathcal{B}$
assumes $bl' \in\# \mathcal{B}$
shows $\mathit{inc\text{-}vec\text{-}of}\ \mathcal{V}s\ bl = \mathit{inc\text{-}vec\text{-}of}\ \mathcal{V}s\ bl' \longleftrightarrow bl = bl'$
⟨*proof*⟩

Incidence matrix column properties

lemma *N-col-def*: $j < \mathrm{b} \Longrightarrow i < \mathrm{v} \Longrightarrow (\mathit{col}\ N\ j)\ \$\ i = (\mathit{if}\ (\mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j)\ \mathit{then}\ 1\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *N-col-def-indiv*: $j < \mathrm{b} \Longrightarrow i < \mathrm{v} \Longrightarrow \mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j \Longrightarrow (\mathit{col}\ N\ j)\ \$\ i = 1$
$j < \mathrm{b} \Longrightarrow i < \mathrm{v} \Longrightarrow \mathcal{V}s\ !\ i \notin \mathcal{B}s\ !\ j \Longrightarrow (\mathit{col}\ N\ j)\ \$\ i = 0$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *N-col-list-map*: $j < \mathrm{b} \Longrightarrow$ *col N j* = *map-vec* (λ *x* . *if* ($x \in (\mathcal{B}s\ !\ j)$) *then 1 else 0*) (*vec-of-list* $\mathcal{V}s$)
⟨*proof*⟩

lemma *N-col-mset-point-set-img*: $j < \mathrm{b} \Longrightarrow$
vec-mset (*col N j*) = *image-mset* (λ *x*. *if* ($x \in (\mathcal{B}s\ !\ j)$) *then 1 else 0*) (*mset-set* $\mathcal{V}$)
⟨*proof*⟩

lemma *matrix-col-to-block*:
assumes $j < \mathrm{b}$
shows $\mathcal{B}s\ !\ j = (\lambda\ k\ .\ \mathcal{V}s\ !\ k)$ ‘ $\{i \in \{..< \mathrm{v}\}\ .\ (col\ N\ j)\ \$\ i = 1\}$
⟨*proof*⟩

lemma *matrix-col-to-block-v2*: $j < \mathrm{b} \Longrightarrow \mathcal{B}s\ !\ j = (\lambda\ k\ .\ \mathcal{V}s\ !\ k)$ ‘ *map-col-to-block* (*col N j*)
⟨*proof*⟩

lemma *matrix-col-in-blocks*: $j < \mathrm{b} \Longrightarrow$ (!) $\mathcal{V}s$ ‘ *map-col-to-block* (*col N j*) $\in\#\ \mathcal{B}$
⟨*proof*⟩

lemma *inc-matrix-col-block*:
assumes $c \in$ *set* (*cols N*)
shows (λ *x*. $\mathcal{V}s\ !\ x$) ‘ (*map-col-to-block c*) $\in\#\ \mathcal{B}$
⟨*proof*⟩

Incidence Matrix Row Definitions

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

lemma *N-row-list-map-elem*: $j < \mathrm{b} \Longrightarrow i < \mathrm{v} \Longrightarrow$
row N i $ $ *j* = *map-vec* (λ *bl* . *if* (($\mathcal{V}s\ !\ i$) $\in$ *bl*) *then 1 else 0*) (*vec-of-list* $\mathcal{B}s$) $ $ *j*
⟨*proof*⟩

lemma *N-row-list-map*: $i < \mathrm{v} \Longrightarrow$
row N i = *map-vec* (λ *bl* . *if* (($\mathcal{V}s\ !\ i$) $\in$ *bl*) *then 1 else 0*) (*vec-of-list* $\mathcal{B}s$)
⟨*proof*⟩

lemma *N-row-mset-blocks-img*: $i < \mathrm{v} \Longrightarrow$
vec-mset (*row N i*) = *image-mset* (λ *x* . *if* (($\mathcal{V}s\ !\ i$) $\in$ *x*) *then 1 else 0*) $\mathcal{B}$
⟨*proof*⟩

Alternate Block representations

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

⟨*proof*⟩

lemma *block-mat-cond-rep'*: $j < \textit{length}\ \mathcal{B}s \Longrightarrow (\mathcal{B}s\ !\ j) = ((!)\ \mathcal{V}s)\ `\ \{i\ .\ i < \textit{length}\ \mathcal{V}s \wedge N\ \$\$\ (i, j) = 1\}$
⟨*proof*⟩

lemma *block-mat-cond-rev*:
assumes $j < \textit{length}\ \mathcal{B}s$
shows $\{i\ .\ i < \textit{length}\ \mathcal{V}s \wedge N\ \$\$\ (i, j) = 1\} = ((\textit{List-Index.index})\ \mathcal{V}s)\ `\ (\mathcal{B}s\ !\ j)$
⟨*proof*⟩

Incidence Matrix incidence system properties

lemma *incomplete-block-col*:
assumes $j < \mathrm{b}$
assumes *incomplete-block* $(\mathcal{B}s\ !\ j)$
shows $0 \in\$\ (\textit{col}\ N\ j)$
⟨*proof*⟩

lemma *mat-rep-num-N-row*:
assumes $i < \mathrm{v}$
shows $\textit{mat-rep-num}\ N\ i = \mathcal{B}\ \textit{rep}\ (\mathcal{V}s\ !\ i)$
⟨*proof*⟩

lemma *point-rep-mat-row-sum*: $i < \mathrm{v} \Longrightarrow \textit{sum-vec}\ (\textit{row}\ N\ i) = \mathcal{B}\ \textit{rep}\ (\mathcal{V}s\ !\ i)$
⟨*proof*⟩

lemma *mat-block-size-N-col*:
assumes $j < \mathrm{b}$
shows $\textit{mat-block-size}\ N\ j = \textit{card}\ (\mathcal{B}s\ !\ j)$
⟨*proof*⟩

lemma *block-size-mat-rep-sum*: $j < \mathrm{b} \Longrightarrow \textit{sum-vec}\ (\textit{col}\ N\ j) = \textit{mat-block-size}\ N\ j$
⟨*proof*⟩

lemma *mat-point-index-rep*:
assumes $I \subseteq \{..<\mathrm{v}\}$
shows $\textit{mat-point-index}\ N\ I = \mathcal{B}\ \textit{index}\ ((\lambda\ i.\ \mathcal{V}s\ !\ i)\ `\ I)$
⟨*proof*⟩

lemma *incidence-mat-two-index*: $i1 < \mathrm{v} \Longrightarrow i2 < \mathrm{v} \Longrightarrow$
$\textit{mat-point-index}\ N\ \{i1, i2\} = \mathcal{B}\ \textit{index}\ \{\mathcal{V}s\ !\ i1, \mathcal{V}s\ !\ i2\}$
⟨*proof*⟩

lemma *ones-incidence-mat-block-size*:
assumes $j < \mathrm{b}$
shows $((u_v\ \mathrm{v})\ {}_v{*}\ N)\ \$\ j = \textit{mat-block-size}\ N\ j$
⟨*proof*⟩

lemma *mat-block-size-conv*: $j < \textit{dim-col}\ N \Longrightarrow \textit{card}\ (\mathcal{B}s\ !\ j) = \textit{mat-block-size}\ N\ j$

⟨*proof*⟩

lemma *mat-inter-num-conv*:
assumes *j1* < *dim-col N j2* < *dim-col N*
shows (*Bs* ! *j1*) |∩| (*Bs* ! *j2*) = *mat-inter-num N j1 j2*
⟨*proof*⟩

lemma *non-empty-col-map-conv*:
assumes *j* < *dim-col N*
shows *non-empty-col N j* ⟷ *Bs* ! *j* ≠ {}
⟨*proof*⟩

lemma *scalar-prod-inc-vec-inter-num*:
assumes *j1* < b *j2* < b
shows (*col N j1*) ⋅ (*col N j2*) = (*Bs* ! *j1*) |∩| (*Bs* ! *j2*)
⟨*proof*⟩

lemma *scalar-prod-block-size-lift-01*:
assumes *i* < b
shows ((*col* (*lift-01-mat N*) *i*) ⋅ (*col* (*lift-01-mat N*) *i*)) = (*of-nat* (*card* (*Bs* ! *i*)) :: ('*b* :: {*ring-1*}))
⟨*proof*⟩

lemma *scalar-prod-inter-num-lift-01*:
assumes *j1* < b *j2* < b
shows ((*col* (*lift-01-mat N*) *j1*) ⋅ (*col* (*lift-01-mat N*) *j2*)) = (*of-nat* ((*Bs* ! *j1*) |∩| (*Bs* ! *j2*)) :: ('*b* :: {*ring-1*}))
⟨*proof*⟩

The System complement's incidence matrix flips 0's and 1's

lemma *map-block-complement-entry*: *j* < b ⟹ (*map block-complement Bs*) ! *j* = *block-complement* (*Bs* ! *j*)
⟨*proof*⟩

lemma *complement-mat-entries*:
assumes *i* < v **and** *j* < b
shows (*Vs* ! *i* ∉ *Bs* ! *j*) ⟷ (*Vs* ! *i* ∈ (*map block-complement Bs*) ! *j*)
⟨*proof*⟩

lemma *length-blocks-complement*: *length* (*map block-complement Bs*) = b
⟨*proof*⟩

lemma *ordered-complement*: *ordered-incidence-system Vs* (*map block-complement Bs*)
⟨*proof*⟩

interpretation *ordered-comp*: *ordered-incidence-system Vs* (*map block-complement Bs*)
⟨*proof*⟩

lemma *complement-mat-entries-val*:
assumes $i < \mathrm{v}$ **and** $j < \mathrm{b}$
shows *ordered-comp.N* $$ $(i, j) = (\mathit{if}\ \mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j\ \mathit{then}\ 0\ \mathit{else}\ 1)$
⟨*proof*⟩

lemma *ordered-complement-mat*: *ordered-comp.N* = *mat* v b (λ (*i*,*j*) . *if* ($\mathcal{V}s$! *i*) ∈ ($\mathcal{B}s$! *j*) *then 0 else 1*)
⟨*proof*⟩

lemma *ordered-complement-mat-map*: *ordered-comp.N* = *map-mat* (λ*x*. *if x* = *1 then 0 else 1*) *N*
⟨*proof*⟩

end

Establishing connection between incidence system and ordered incidence system locale

lemma (**in** *incidence-system*) *alt-ordering-sysI*: *Vs* ∈ *permutations-of-set* $\mathcal{V}$ ⟹ *Bs* ∈ *permutations-of-multiset* $\mathcal{B}$ ⟹
ordered-incidence-system Vs Bs
⟨*proof*⟩

lemma (**in** *finite-incidence-system*) *exists-ordering-sysI*: ∃ *Vs Bs* . *Vs* ∈ *permutations-of-set* $\mathcal{V}$ ∧
Bs ∈ *permutations-of-multiset* $\mathcal{B}$ ∧ *ordered-incidence-system Vs Bs*
⟨*proof*⟩

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*⟩

Generalise the idea of an incidence matrix to an unordered context

definition *is-incidence-matrix* :: *'c* :: {*ring-1*} *mat* ⇒ *'a set* ⇒ *'a set multiset* ⇒ *bool* **where**
is-incidence-matrix N V B ⟷
(∃ *Vs Bs* . (*Vs* ∈ *permutations-of-set V* ∧ *Bs* ∈ *permutations-of-multiset B* ∧ *N* = (*inc-mat-of Vs Bs*)))

lemma (**in** *incidence-system*) *is-incidence-mat-alt*: *is-incidence-matrix N* $\mathcal{V}$ $\mathcal{B}$ ⟷

(∃ *Vs Bs*. (*set Vs* = $\mathcal{V}$ ∧ *mset Bs* = $\mathcal{B}$ ∧ *ordered-incidence-system Vs Bs* ∧ *N* = (*inc-mat-of Vs Bs*)))
⟨*proof*⟩

lemma (**in** *ordered-incidence-system*) *is-incidence-mat-true*: *is-incidence-matrix N* $\mathcal{V}$ $\mathcal{B}$ = *True*

⟨*proof*⟩

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 $j < \mathrm{b}$
shows *1* ∈$ (*col N j*)
⟨*proof*⟩

lemma *all-cols-non-empty*: $j <$ *dim-col N* $\Longrightarrow$ *non-empty-col N j*
⟨*proof*⟩
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* $\mathcal{B}s$
⟨*proof*⟩

lemma *distinct-cols-N*: *distinct* (*cols N*)
⟨*proof*⟩

lemma *simp-blocks-length-card*: *length* $\mathcal{B}s$ = *card* (*set* $\mathcal{B}s$)
⟨*proof*⟩

lemma *blocks-index-inj-on*: *inj-on* (λ *i* . $\mathcal{B}s$! *i*) {*0..<length* $\mathcal{B}s$}
⟨*proof*⟩

lemma *x-in-block-set-img*: **assumes** *x* ∈ *set* $\mathcal{B}s$ **shows** *x* ∈ (!) $\mathcal{B}s$ ‘ {*0..<length* $\mathcal{B}s$}
⟨*proof*⟩

lemma *blocks-index-simp-bij-betw*: *bij-betw* (λ *i* . $\mathcal{B}s$! *i*) {*0..<length* $\mathcal{B}s$} (*set* $\mathcal{B}s$)
⟨*proof*⟩

lemma *blocks-index-simp-unique*: *i1* < *length* $\mathcal{B}s$ $\Longrightarrow$ *i2* < *length* $\mathcal{B}s$ $\Longrightarrow$ *i1* ≠ *i2* $\Longrightarrow$ $\mathcal{B}s$! *i1* ≠ $\mathcal{B}s$! *i2*
⟨*proof*⟩

lemma *lift-01-distinct-cols-N*: *distinct* (*cols* (*lift-01-mat N*))
⟨*proof*⟩

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*
  ⟨*proof*⟩

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 $\Longrightarrow$ *mat-rep-num N i* = r
  ⟨*proof*⟩

lemma *incidence-mat-rep-num-sum*: $i <$ v $\Longrightarrow$ *sum-vec* (*row N i*) = r
  ⟨*proof*⟩

lemma *transpose-N-mult-diag*:
  assumes $i = j$ **and** $i <$ v **and** $j <$ v
  shows ($N * N^T$) $$ (i, j) = r
⟨*proof*⟩

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 $\Longrightarrow$ *mat-block-size N j* = k
  ⟨*proof*⟩

lemma *incidence-mat-block-size-sum*: $j <$ b $\Longrightarrow$ *sum-vec* (*col N j*) = k
  ⟨*proof*⟩

lemma *ones-mult-incidence-mat-k-index*: $j <$ b $\Longrightarrow$ ((u_v v) $_v*$ N) $ j = k
  ⟨*proof*⟩

lemma *ones-mult-incidence-mat-k*: ((u_v v) $_v*$ N) = k $\cdot_v$ (u_v b)
  ⟨*proof*⟩

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*: $j < \mathrm{b} \Longrightarrow 0 \in\$ (col\ N\ j)$
  ⟨*proof*⟩

end

locale *ordered-t-wise-balance* = *ordered-proper-design* $\mathcal{V}s$ $\mathcal{B}s$ + *t-wise-balance set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ t Λ_t
  for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** t **and** Λ_t

begin

lemma *incidence-mat-des-index*:
  assumes $I \subseteq \{0..<\mathrm{v}\}$
  assumes *card* $I = \mathrm{t}$
  shows *mat-point-index* $N\ I = \Lambda_t$
⟨*proof*⟩

end

locale *ordered-pairwise-balance* = *ordered-t-wise-balance* $\mathcal{V}s$ $\mathcal{B}s$ *2* Λ + *pairwise-balance 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* $< \mathrm{v}$
  assumes *i2* $< \mathrm{v}$
  assumes *i1* $\neq$ *i2*
  shows *mat-point-index* $N\ \{i1, i2\} = \Lambda$
  ⟨*proof*⟩

lemma *transpose-N-mult-off-diag*:
  assumes $i \neq j$ **and** $i < \mathrm{v}$ **and** $j < \mathrm{v}$
  shows $(N * N^T)\ \$\$\ (i, j) = \Lambda$
⟨*proof*⟩

end

context *pairwise-balance*
begin

lemma *ordered-pbdI*:

assumes $\mathcal{B}$ = *mset* $\mathcal{B}s$ **and** $\mathcal{V}$ = *set* $\mathcal{V}s$ **and** *distinct* $\mathcal{V}s$
shows *ordered-pairwise-balance* $\mathcal{V}s$ $\mathcal{B}s$ Λ
⟨*proof*⟩
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* ⊆ *ordered-constant-rep*
⟨*proof*⟩

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*: $N * (N^T) = \Lambda \cdot_m (J_m\ \mathrm{v}) + (\mathrm{r} - \Lambda) \cdot_m (1_m\ \mathrm{v})$
⟨*proof*⟩
end

locale *ordered-bibd* = *ordered-proper-design* $\mathcal{V}s$ $\mathcal{B}s$ + *bibd set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ k Λ
for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** k **and** Λ

sublocale *ordered-bibd* ⊆ *ordered-incomplete-design*
⟨*proof*⟩

sublocale *ordered-bibd* ⊆ *ordered-constant-rep* $\mathcal{V}s$ $\mathcal{B}s$ r
⟨*proof*⟩

sublocale *ordered-bibd* ⊆ *ordered-pairwise-balance*
⟨*proof*⟩

locale *ordered-sym-bibd* = *ordered-bibd* $\mathcal{V}s$ $\mathcal{B}s$ k Λ + *symmetric-bibd set* $\mathcal{V}s$ *mset* $\mathcal{B}s$ k Λ
for $\mathcal{V}s$ **and** $\mathcal{B}s$ **and** k **and** Λ

sublocale *ordered-sym-bibd* ⊆ *ordered-simple-design*
⟨*proof*⟩

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 *card bl* = m
assumes *bl2* $\in\#$ $\mathcal{B}$ $-$ {#*bl*#}
shows m $<$ *card bl2*
⟨*proof*⟩

lemma *block-size-inter-num-cases*:
assumes *bl* $\in\#$ $\mathcal{B}$
assumes b $\geq$ *2*
shows m $<$ *card bl* $\vee$ (*card bl* = m $\wedge$ ($\forall$ *bl'* $\in\#$ ($\mathcal{B}$ $-$ {#*bl*#}) . m $<$ *card bl'*))
⟨*proof*⟩

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*⟩

lemma *const-intersect-block-size-diff*:
assumes *j'* $<$ b **and** *j* $<$ b **and** *j* $\neq$ *j'* **and** *card* ($\mathcal{B}s$! *j'*) = m **and** b $\geq$ *2*
shows *card* ($\mathcal{B}s$! *j*) $-$ m $>$ *0*
⟨*proof*⟩

lemma *scalar-prod-inc-vec-const-inter*:
assumes *j1* $<$ b *j2* $<$ b *j1* $\neq$ *j2*
shows (*col N j1*) $\cdot$ (*col N j2*) = m
⟨*proof*⟩

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..<*(*dim-row M*)] (*map* (*map-col-to-block*) (*cols M*))
⟨*proof*⟩

interpretation *mat-ord-inc-sys*: *ordered-incidence-system* [*0..<*(*dim-row M*)] (*map* (*map-col-to-block*) (*cols M*))

⟨*proof*⟩

lemma *mat-ord-inc-sys-N*: *mat-ord-inc-sys.N* = *lift-01-mat M*
⟨*proof*⟩

lemma *map-col-to-block-mat-rep-num*:
assumes $x <$*dim-row M*
shows ({# *map-col-to-block c* . $c \in\#$ *mset* (*cols M*)#} *rep x*) = *mat-rep-num M x*
⟨*proof*⟩

end

context *zero-one-matrix-ring-1*
begin

lemma *transpose-cond-index-vals*:
assumes $M * (M^T) = \Lambda \cdot_m (J_m\ (\textit{dim-row}\ M)) + (r - \Lambda) \cdot_m (1_m\ (\textit{dim-row}\ M))$
assumes $i <$ *dim-row* $(M * (M^T))$
assumes $j <$ *dim-col* $(M * (M^T))$
shows $i = j \Longrightarrow (M * (M^T))\ \$\$\ (i, j) = r\ i \neq j \Longrightarrow (M * (M^T))\ \$\$\ (i, j) = \Lambda$
⟨*proof*⟩

end

locale *zero-one-matrix-int* = *zero-one-matrix-ring-1 M* **for** *M* :: *int mat*
begin

Some useful conditions on the transpose product for matrix system properties

lemma *transpose-cond-diag-r*:
assumes $i <$ *dim-row* $(M * (M^T))$
assumes $\bigwedge j.\ i = j \Longrightarrow (M * (M^T))\ \$\$\ (i, j) = r$
shows *mat-rep-num M i* = *r*
⟨*proof*⟩

lemma *transpose-cond-non-diag*:
assumes *i1* $<$ *dim-row* $(M * (M^T))$
assumes *i2* $<$ *dim-row* $(M * (M^T))$
assumes *i1* $\neq$ *i2*
assumes $\bigwedge j\ i.\ j \neq i \Longrightarrow i <$ *dim-row* $(M * (M^T)) \Longrightarrow j <$ *dim-row* $(M * (M^T)) \Longrightarrow (M * (M^T))\ \$\$\ (i, j) = \Lambda$
shows Λ = *mat-point-index M* {*i1*, *i2*}
⟨*proof*⟩

lemma *trans-cond-implies-map-rep-num*:
assumes $M * (M^T) = \Lambda \cdot_m (J_m\ (\textit{dim-row}\ M)) + (r - \Lambda) \cdot_m (1_m\ (\textit{dim-row}\ M))$
assumes $x <$ *dim-row M*

shows (*image-mset map-col-to-block* (*mset* (*cols M*))) *rep x* = *r*
⟨*proof*⟩

lemma *trans-cond-implies-map-index*:
assumes $M * (M^T) = \Lambda \cdot_m (J_m\ (\textit{dim-row}\ M)) + (r - \Lambda) \cdot_m (1_m\ (\textit{dim-row}\ M))$
assumes *ps* ⊆ {*0..<dim-row M*}
assumes *card ps* = *2*
shows (*image-mset map-col-to-block* (*mset* (*cols M*))) *index ps* = Λ
⟨*proof*⟩

Stinson Theorem 1.15 existence direction

lemma *rpbd-exists*:
assumes *dim-row M* ≥ *2* — Min two points
assumes *dim-col M* ≥ *1* — Min one block
assumes ⋀ *j*. *j* < *dim-col M* ⟹ *1* ∈$ *col M j* — no empty blocks
assumes $M * (M^T) = \Lambda \cdot_m (J_m\ (\textit{dim-row}\ M)) + (r - \Lambda) \cdot_m (1_m\ (\textit{dim-row}\ M))$
shows *ordered-regular-pairwise-balance* [*0..<dim-row M*] (*map map-col-to-block* (*cols M*)) Λ *r*
⟨*proof*⟩

lemma *vec-k-uniform-mat-block-size*:
assumes $((u_v\ (\textit{dim-row}\ M))\ {}_v{*}\ M) = k \cdot_v (u_v\ (\textit{dim-col}\ M))$
assumes *j* < *dim-col M*
shows *mat-block-size M j* = *k*
⟨*proof*⟩

lemma *vec-k-impl-uniform-block-size*:
assumes $((u_v\ (\textit{dim-row}\ M))\ {}_v{*}\ M) = k \cdot_v (u_v\ (\textit{dim-col}\ M))$
assumes *bl* ∈# (*image-mset map-col-to-block* (*mset* (*cols M*)))
shows *card bl* = *k*
⟨*proof*⟩

lemma *bibd-exists*:
assumes *dim-col M* ≥ *1* — Min one block
assumes ⋀ *j*. *j* < *dim-col M* ⟹ *1* ∈$ *col M j* — no empty blocks
assumes $M * (M^T) = \Lambda \cdot_m (J_m\ (\textit{dim-row}\ M)) + (r - \Lambda) \cdot_m (1_m\ (\textit{dim-row}\ M))$
assumes $((u_v\ (\textit{dim-row}\ M))\ {}_v{*}\ M) = k \cdot_v (u_v\ (\textit{dim-col}\ M))$
assumes (*r* ::*nat*) ≥ *0*
assumes *k* ≥ *2 k* < *dim-row M*
shows *ordered-bibd* [*0..<dim-row M*] (*map map-col-to-block* (*cols M*)) *k* Λ
⟨*proof*⟩

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.*$\mathcal{B}$ (λ *x* . $\mathcal{V}s'$! (*List-Index.index* $\mathcal{V}s$ *x*))
⟨*proof*⟩

lemma *equal-inc-mat-isomorphism-ex*: *D1.N* = *D2.N* $\Longrightarrow$ $\exists$ π . *incidence-system-isomorphism D1.*$\mathcal{V}$ *D1.*$\mathcal{B}$ *D2.*$\mathcal{V}$ *D2.*$\mathcal{B}$ π
⟨*proof*⟩

lemma *equal-inc-mat-isomorphism-obtain*:
assumes *D1.N* = *D2.N*
obtains π **where** *incidence-system-isomorphism D1.*$\mathcal{V}$ *D1.*$\mathcal{B}$ *D2.*$\mathcal{V}$ *D2.*$\mathcal{B}$ π
⟨*proof*⟩

end

context *incidence-system-isomorphism*
begin

lemma *exists-eq-inc-mats*:
assumes *finite* $\mathcal{V}$ *finite* $\mathcal{V}'$
obtains *N* **where** *is-incidence-matrix N* $\mathcal{V}$ $\mathcal{B}$ **and** *is-incidence-matrix N* $\mathcal{V}'$ $\mathcal{B}'$
⟨*proof*⟩

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.. <length Bs

definition *dual-blocks* :: *'a set* ⇒ *'a set list* ⇒ *nat set multiset* **where**
dual-blocks $\mathcal{V}$ $\mathcal{B}s$ ≡ {# {*y* . *y* < *length* $\mathcal{B}s$ ∧ *x* ∈ $\mathcal{B}s$! *y*} . *x* ∈# (*mset-set* $\mathcal{V}$)#}

lemma *dual-blocks-wf*: *b* ∈# *dual-blocks V Bs* ⟹ *b* ⊆ {*0..*<*length Bs*}
⟨*proof*⟩

context *ordered-incidence-system*
begin

definition *dual-blocks-ordered* :: *nat set list* ($\mathcal{B}s*$) **where**
dual-blocks-ordered ≡ *map* (λ *x* . {*y* . *y* < *length* $\mathcal{B}s$ ∧ *x* ∈ $\mathcal{B}s$! *y*}) $\mathcal{V}s$

lemma *dual-blocks-ordered-eq*: *dual-blocks* $\mathcal{V}$ $\mathcal{B}s$= *mset* ($\mathcal{B}s*$)
⟨*proof*⟩

lemma *dual-blocks-len*: *length* $\mathcal{B}s*$ = *length* $\mathcal{V}s$
⟨*proof*⟩

A dual system is an incidence system

sublocale *dual-sys*: *finite-incidence-system* {*0..*<*length* $\mathcal{B}s$} *dual-blocks* $\mathcal{V}$ $\mathcal{B}s$
⟨*proof*⟩

lemma *dual-is-ordered-inc-sys*: *ordered-incidence-system* [*0..*<*length* $\mathcal{B}s$] $\mathcal{B}s*$
⟨*proof*⟩

interpretation *ordered-dual-sys*: *ordered-incidence-system* [*0..*<*length* $\mathcal{B}s$] $\mathcal{B}s*$
⟨*proof*⟩

5.2 Basic Dual Properties

lemma *ord-dual-blocks-b*: *ordered-dual-sys*.b = v
⟨*proof*⟩

lemma *dual-blocks-b*: *dual-sys*.b = v
⟨*proof*⟩

lemma *dual-blocks-v*: *dual-sys*.v = b
⟨*proof*⟩

lemma *ord-dual-blocks-v*: *ordered-dual-sys*.v = b
⟨*proof*⟩

lemma *dual-point-block*: *i* < v ⟹ $\mathcal{B}s*$! *i* = {*y*. *y* < *length* $\mathcal{B}s$ ∧ ($\mathcal{V}s$! *i*) ∈ $\mathcal{B}s$! *y*}
⟨*proof*⟩

lemma *dual-incidence-iff*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow \mathcal{B}s\ !\ j = bl \Longrightarrow \mathcal{V}s\ !\ i = x \Longrightarrow (x \in bl \longleftrightarrow j \in \mathcal{B}s{*}\ !\ i)$
⟨*proof*⟩

lemma *dual-incidence-iff2*: $i < \mathrm{v} \Longrightarrow j < \mathrm{b} \Longrightarrow (\mathcal{V}s\ !\ i \in \mathcal{B}s\ !\ j \longleftrightarrow j \in \mathcal{B}s{*}\ !\ i)$
⟨*proof*⟩

lemma *dual-blocks-point-exists*: $bl \in\# \mathit{dual\text{-}blocks}\ \mathcal{V}\ \mathcal{B}s \Longrightarrow \exists\ x.\ x \in \mathcal{V} \wedge bl = \{y\ .\ y < \mathit{length}\ \mathcal{B}s \wedge x \in \mathcal{B}s\ !\ y\}$
⟨*proof*⟩

lemma *dual-blocks-ne-index-ne*: $j1 < \mathit{length}\ \mathcal{B}s{*} \Longrightarrow j2 < \mathit{length}\ \mathcal{B}s{*} \Longrightarrow \mathcal{B}s{*}\ !\ j1 \neq \mathcal{B}s{*}\ !\ j2 \Longrightarrow j1 \neq j2$
⟨*proof*⟩

lemma *dual-blocks-list-index-img*: $\mathit{image\text{-}mset}\ (\lambda x\ .\ \mathcal{B}s{*}\ !\ x)\ (\mathit{mset\text{-}set}\ \{0..<\mathit{length}\ \mathcal{B}s{*}\}) = \mathit{mset}\ \mathcal{B}s{*}$
⟨*proof*⟩

lemma *dual-blocks-elem-iff*:
assumes $j < \mathrm{v}$
shows $x \in (\mathcal{B}s{*}\ !\ j) \longleftrightarrow \mathcal{V}s\ !\ j \in \mathcal{B}s\ !\ x \wedge x < \mathrm{b}$
⟨*proof*⟩

The incidence matrix of the dual of a design is just the transpose

lemma *dual-incidence-mat-eq-trans*: $\mathit{ordered\text{-}dual\text{-}sys.N} = N^T$
⟨*proof*⟩

lemma *dual-incidence-mat-eq-trans-rev*: $(\mathit{ordered\text{-}dual\text{-}sys.N})^T = N$
⟨*proof*⟩

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 $(\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ \mathit{rep}\ x > 0)$
assumes $bl \in\# \mathit{dual\text{-}blocks}\ \mathcal{V}\ \mathcal{B}s$
shows $bl \neq \{\}$
⟨*proof*⟩

lemma *dual-blocks-size-is-rep*: $j < \mathit{length}\ \mathcal{B}s{*} \Longrightarrow \mathit{card}\ (\mathcal{B}s{*}\ !\ j) = \mathcal{B}\ \mathit{rep}\ (\mathcal{V}s\ !\ j)$
⟨*proof*⟩

lemma *dual-blocks-size-is-rep-obtain*:
assumes $bl \in\# \mathit{dual\text{-}blocks}\ \mathcal{V}\ \mathcal{B}s$
obtains x **where** $x \in \mathcal{V}$ **and** $\mathit{card}\ bl = \mathcal{B}\ \mathit{rep}\ x$
⟨*proof*⟩

lemma *dual-blocks-rep-is-size*:
assumes $i < \mathit{length}\ \mathcal{B}s$
shows $(\mathit{mset}\ \mathcal{B}s*)\ \mathit{rep}\ i = \mathit{card}\ (\mathcal{B}s\ !\ i)$
⟨*proof*⟩

lemma *dual-blocks-inter-index*:
assumes $j1 < \mathit{length}\ \mathcal{B}s*\ j2 < \mathit{length}\ \mathcal{B}s*$
shows $(\mathcal{B}s*\ !\ j1)\ |\cap|\ (\mathcal{B}s*\ !\ j2) = \mathit{points\text{-}index}\ \mathcal{B}\ \{\mathcal{V}s\ !\ j1, \mathcal{V}s\ !\ j2\}$
⟨*proof*⟩

lemma *dual-blocks-points-index-inter*:
assumes $i1 < \mathrm{b}\ i2 < \mathrm{b}$
shows $(\mathit{mset}\ \mathcal{B}s*)\ \mathit{index}\ \{i1, i2\} = (\mathcal{B}s\ !\ i1)\ |\cap|\ (\mathcal{B}s\ !\ i2)$
⟨*proof*⟩

end

5.4 Dual Properties for Design sub types

context *ordered-design*
begin

lemma *dual-is-design*:
assumes $(\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ \mathit{rep}\ x > 0)$ — Required to ensure no blocks are empty
shows $\mathit{design}\ \{0..<\mathit{length}\ \mathcal{B}s\}\ (\mathit{dual\text{-}blocks}\ \mathcal{V}\ \mathcal{B}s)$
⟨*proof*⟩
end

context *ordered-proper-design*
begin

lemma *dual-sys-b-non-zero*: $\mathit{dual\text{-}sys}.\mathrm{b} \neq 0$
⟨*proof*⟩

lemma *dual-is-proper-design*:
assumes $(\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ \mathit{rep}\ x > 0)$ — Required to ensure no blocks are empty
shows $\mathit{proper\text{-}design}\ \{0..<\mathit{length}\ \mathcal{B}s\}\ (\mathit{dual\text{-}blocks}\ \mathcal{V}\ \mathcal{B}s)$

⟨*proof*⟩

end

context *ordered-block-design*
begin

lemma *dual-blocks-const-rep*: $i \in \{0..<length\ \mathcal{B}s\} \Longrightarrow (mset\ \mathcal{B}s*)\ rep\ i = \mathrm{k}$
⟨*proof*⟩

lemma *dual-blocks-constant-rep-design*:
assumes $(\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ rep\ x > 0)$
shows *constant-rep-design* $\{0..<length\ \mathcal{B}s\}$ (*dual-blocks* $\mathcal{V}\ \mathcal{B}s$) k
⟨*proof*⟩

end

context *ordered-constant-rep*
begin

lemma *dual-blocks-const-size*: $j < length\ \mathcal{B}s* \Longrightarrow card\ (\mathcal{B}s*\ !\ j) = \mathrm{r}$
⟨*proof*⟩

lemma *dual-is-block-design*: *block-design* $\{0..<length\ \mathcal{B}s\}$ (*dual-blocks* $\mathcal{V}\ \mathcal{B}s$) r
⟨*proof*⟩

end

context *ordered-pairwise-balance*
begin

lemma *dual-blocks-const-intersect*:
assumes $j1 < length\ \mathcal{B}s*\ j2 < length\ \mathcal{B}s*$
assumes $j1 \neq j2$
shows $(\mathcal{B}s*\ !\ j1)\ |\cap|\ (\mathcal{B}s*\ !\ j2) = \Lambda$
⟨*proof*⟩

lemma *dual-is-const-intersect-des*:
assumes $\Lambda > 0$
shows *const-intersect-design* $\{0..<(length\ \mathcal{B}s)\}$ (*dual-blocks* $\mathcal{V}\ \mathcal{B}s$) Λ
⟨*proof*⟩

lemma *dual-is-simp-const-inter-des*:
assumes $\Lambda > 0$
assumes $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow incomplete\text{-}block\ bl$
shows *simple-const-intersect-design* $\{0..<(length\ \mathcal{B}s)\}$ (*dual-blocks* $\mathcal{V}\ \mathcal{B}s$) Λ
⟨*proof*⟩

end

context *ordered-const-intersect-design*
begin

lemma *dual-is-balanced*:
  assumes $ps \subseteq \{0..<length\ \mathcal{B}s\}$
  assumes $card\ ps = 2$
  shows $(dual\text{-}blocks\ \mathcal{V}\ \mathcal{B}s)\ index\ ps = \mathrm{m}$
⟨*proof*⟩

lemma *dual-is-pbd*:
  assumes $(\bigwedge x\ .\ x \in \mathcal{V} \Longrightarrow \mathcal{B}\ rep\ x > 0)$
  assumes $\mathrm{b} \geq 2$
  shows $pairwise\text{-}balance\ \{0..<(length\ \mathcal{B}s)\}\ (dual\text{-}blocks\ \mathcal{V}\ \mathcal{B}s)\ \mathrm{m}$
⟨*proof*⟩

end

context *ordered-sym-bibd*
begin

lemma *dual-is-balanced*:
  assumes $ps \subseteq \{0..<length\ \mathcal{B}s\}$
  assumes $card\ ps = 2$
  shows $(dual\text{-}blocks\ \mathcal{V}\ \mathcal{B}s)\ index\ ps = \Lambda$
⟨*proof*⟩

lemma *dual-bibd*: $bibd\ \{0..<(length\ \mathcal{B}s)\}\ (dual\text{-}blocks\ \mathcal{V}\ \mathcal{B}s)\ \mathrm{r}\ \Lambda$
⟨*proof*⟩

The dual of a BIBD must by symmetric

lemma *dual-bibd-symmetric*: $symmetric\text{-}bibd\ \{0..<(length\ \mathcal{B}s)\}\ (dual\text{-}blocks\ \mathcal{V}\ \mathcal{B}s)\ \mathrm{r}\ \Lambda$
⟨*proof*⟩

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* ⇒ *'b set list* ⇒ *bool* **where**
is-dual Vs' Bs' ≡ *ordered-incidence-system Vs' Bs'* ∧ (*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'*
⟨*proof*⟩

lemma *is-dualD1*:
assumes *is-dual Vs' Bs'*
shows (*inc-mat-of Vs' Bs'* = N^T)
⟨*proof*⟩

lemma *is-dualD2*:
assumes *is-dual Vs' Bs'*
shows *ordered-incidence-system Vs' Bs'*
⟨*proof*⟩

lemma *generated-is-dual*: *is-dual* [*0..<*(*length* $\mathcal{B}s$)] $\mathcal{B}s*$
⟨*proof*⟩

lemma *is-dual-isomorphism-generated*:
assumes *is-dual Vs' Bs'*
shows ∃ *π*. *incidence-system-isomorphism* (*set Vs'*) (*mset Bs'*) ({*0..<*(*length* $\mathcal{B}s$)}) (*dual-blocks* $\mathcal{V}$ $\mathcal{B}s$) *π*
⟨*proof*⟩

interpretation *ordered-dual-sys*: *ordered-incidence-system* [*0..<length* $\mathcal{B}s$] $\mathcal{B}s*$
⟨*proof*⟩

Original system is dual of the dual

lemma *is-dual-rev*: *ordered-dual-sys.is-dual* $\mathcal{V}s$ $\mathcal{B}s$
⟨*proof*⟩

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-row A* ⟹ *j* < *dim-col A* ⟹ *addrow a k l A* $$ (*i*,*j*) = (*if k* = *i then* (*a* ∗ (*A* $$ (*l*,*j*)) + (*A* $$ (*i*,*j*))) *else A* $$ (*i*,*j*))
i < *dim-row A* ⟹ *j* < *dim-col A* ⟹ *addrow a i l A* $$ (*i*,*j*) = (*a* ∗ (*A* $$ (*l*,*j*)) + (*A* $$ (*i*,*j*)))
i < *dim-row A* ⟹ *j* < *dim-col A* ⟹ *k* ≠ *i* ⟹ *addrow a k l A* $$ (*i*,*j*) = *A* $$(*i*,*j*)
dim-row (*addrow a k l A*) = *dim-row A dim-col* (*addrow a k l A*) = *dim-col A*
⟨*proof*⟩

Function to add a column to multiple other columns

fun *add-col-to-multiple* :: *'a* :: *semiring-1* ⇒ *nat list* ⇒ *nat* ⇒ *'a mat* ⇒ *'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* ⇒ *nat list* ⇒ *nat* ⇒ *'a mat* ⇒ *'a mat* **where**
add-row-to-multiple a [] *l A* = *A* |
add-row-to-multiple a (*k* # *ks*) *l A* = (*addrow a k l* (*add-row-to-multiple a ks l A*))

Function to add multiple rows to a single row

fun *add-multiple-rows* :: *'a* :: *semiring-1* ⇒ *nat* ⇒ *nat list* ⇒ *'a mat* ⇒ *'a mat* **where**
add-multiple-rows a k [] *A* = *A* |
add-multiple-rows a k (*l* # *ls*) *A* = (*addrow a k l* (*add-multiple-rows a k ls A*))

Function to add multiple columns to a single col

fun *add-multiple-cols* :: *'a* :: *semiring-1* ⇒ *nat* ⇒ *nat list* ⇒ *'a mat* ⇒ *'a mat* **where**
add-multiple-cols a k [] *A* = *A* |
add-multiple-cols a k (*l* # *ls*) *A* = (*addcol a k l* (*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*]:
dim-row (*add-multiple-rows a k ls A*) = *dim-row A*
dim-col (*add-multiple-rows a k ls A*) = *dim-col A*
⟨*proof*⟩

lemma *add-multiple-rows-index-unchanged* [*simp*]:
i < *dim-row A* ⟹ *j* < *dim-col A* ⟹ *k* ≠ *i* ⟹ *add-multiple-rows a k ls A* $$ (*i*,*j*) = *A* $$(*i*,*j*)
⟨*proof*⟩

lemma *add-multiple-rows-index-eq*:
assumes $i < \textit{dim-row}\ A$ **and** $j < \textit{dim-col}\ A$ **and** $i \notin \textit{set}\ ls$ **and** $\bigwedge l\ .\ l \in \textit{set}\ ls \Longrightarrow l < \textit{dim-row}\ A$
shows $\textit{add-multiple-rows}\ a\ i\ ls\ A\ \$\$\ (i,j) = (\sum l{\leftarrow}ls.\ a * A\ \$\$(l,j)) + A\$\$(i,j)$
⟨*proof*⟩

lemma *add-multiple-rows-index-eq-bounds*:
assumes $i < \textit{dim-row}\ A$ **and** $j < \textit{dim-col}\ A$ **and** $i < low \vee i \geq up$ **and** $up \leq \textit{dim-row}\ A$
shows $\textit{add-multiple-rows}\ a\ i\ [low..{<}up]\ A\ \$\$\ (i,j) = (\sum l{=}low..{<}up.\ a * A\ \$\$(l,j)) + A\$\$(i,j)$
⟨*proof*⟩

lemma *add-multiple-cols-dim* [*simp*]:
$\textit{dim-row}\ (\textit{add-multiple-cols}\ a\ k\ ls\ A) = \textit{dim-row}\ A$
$\textit{dim-col}\ (\textit{add-multiple-cols}\ a\ k\ ls\ A) = \textit{dim-col}\ A$
⟨*proof*⟩

lemma *add-multiple-cols-index-unchanged* [*simp*]:
$i < \textit{dim-row}\ A \Longrightarrow j < \textit{dim-col}\ A \Longrightarrow k \neq j \Longrightarrow \textit{add-multiple-cols}\ a\ k\ ls\ A\ \$\$\ (i,j) = A\ \$\$(i,j)$
⟨*proof*⟩

lemma *add-multiple-cols-index-eq*:
assumes $i < \textit{dim-row}\ A$ **and** $j < \textit{dim-col}\ A$ **and** $j \notin \textit{set}\ ls$ **and** $\bigwedge l\ .\ l \in \textit{set}\ ls \Longrightarrow l < \textit{dim-col}\ A$
shows $\textit{add-multiple-cols}\ a\ j\ ls\ A\ \$\$\ (i,j) = (\sum l{\leftarrow}ls.\ a * A\ \$\$(i,l)) + A\$\$(i,j)$
⟨*proof*⟩

lemma *add-multiple-cols-index-eq-bounds*:
assumes $i < \textit{dim-row}\ A$ **and** $j < \textit{dim-col}\ A$ **and** $j < low \vee j \geq up$ **and** $up \leq \textit{dim-col}\ A$
shows $\textit{add-multiple-cols}\ a\ j\ [low..{<}up]\ A\ \$\$\ (i,j) = (\sum l{=}low..{<}up.\ a * A\ \$\$(i,l)) + A\$\$(i,j)$
⟨*proof*⟩

lemma *add-row-to-multiple-dim* [*simp*]:
$\textit{dim-row}\ (\textit{add-row-to-multiple}\ a\ ks\ l\ A) = \textit{dim-row}\ A$
$\textit{dim-col}\ (\textit{add-row-to-multiple}\ a\ ks\ l\ A) = \textit{dim-col}\ A$
⟨*proof*⟩

lemma *add-row-to-multiple-index-unchanged* [*simp*]:
$i < \textit{dim-row}\ A \Longrightarrow j < \textit{dim-col}\ A \Longrightarrow i \notin \textit{set}\ ks \Longrightarrow \textit{add-row-to-multiple}\ a\ ks\ l\ A\ \$\$\ (i,j) = A\ \$\$(i,j)$
⟨*proof*⟩

lemma *add-row-to-multiple-index-unchanged-bound*:
$i < \textit{dim-row}\ A \Longrightarrow j < \textit{dim-col}\ A \Longrightarrow i < low \Longrightarrow i \geq up \Longrightarrow$

add-row-to-multiple a [low..<up] l A $$ (i,j) = A $$(i,j)
⟨*proof*⟩

lemma *add-row-to-multiple-index-change*:
assumes $i <$ *dim-row A* **and** $j <$ *dim-col A* **and** $i \in$ *set ks* **and** *distinct ks* **and** $l \notin$ *set ks*
and $l <$ *dim-row A*
shows *add-row-to-multiple a ks l A $$ (i,j) = (a * A$$(l, j)) + A$$(i,j)*
⟨*proof*⟩

lemma *add-row-to-multiple-index-change-bounds*:
assumes $i <$ *dim-row A* **and** $j <$ *dim-col A* **and** $i \geq$ *low* **and** $i <$ *up* **and** $l <$ *low* $\vee$ $l \geq$ *up*
and $l <$ *dim-row A*
shows *add-row-to-multiple a [low..<up] l A $$ (i,j) = (a * A$$(l, j)) + A$$(i,j)*
⟨*proof*⟩

lemma *add-col-to-multiple-dim* [*simp*]:
dim-row (add-col-to-multiple a ks l A) = dim-row A
dim-col (add-col-to-multiple a ks l A) = dim-col A
⟨*proof*⟩

lemma *add-col-to-multiple-index-unchanged* [*simp*]:
$i <$ *dim-row A* $\Longrightarrow$ $j <$ *dim-col A* $\Longrightarrow$ $j \notin$ *set ks* $\Longrightarrow$ *add-col-to-multiple a ks l A $$ (i,j) = A $$(i,j)*
⟨*proof*⟩

lemma *add-col-to-multiple-index-unchanged-bound*:
$i <$ *dim-row A* $\Longrightarrow$ $j <$ *dim-col A* $\Longrightarrow$ $j <$ *low* $\Longrightarrow$ $j \geq$ *up* $\Longrightarrow$
add-col-to-multiple a [low..<up] l A $$ (i,j) = A $$(i,j)
⟨*proof*⟩

lemma *add-col-to-multiple-index-change*:
assumes $i <$ *dim-row A* **and** $j <$ *dim-col A* **and** $j \in$ *set ks* **and** *distinct ks* **and** $l \notin$ *set ks*
and $l <$ *dim-col A*
shows *add-col-to-multiple a ks l A $$ (i,j) = (a * A$$(i, l)) + A$$(i,j)*
⟨*proof*⟩

lemma *add-col-to-multiple-index-change-bounds*:
assumes $i <$ *dim-row A* **and** $j <$ *dim-col A* **and** $j \geq$ *low* **and** $j <$ *up* **and** $l <$ *low* $\vee$ $l \geq$ *up*
and $l <$ *dim-col A*
shows *add-col-to-multiple a [low..<up] l A $$ (i,j) = (a * A$$(i, l)) + A$$(i,j)*
⟨*proof*⟩

Operations specifically on 1st row/column

lemma *add-first-row-to-multiple-index*:

assumes $i < dim\text{-}row\ M$ **and** $j < dim\text{-}col\ M$
shows $i = 0 \Longrightarrow (add\text{-}row\text{-}to\text{-}multiple\ a\ [1..<dim\text{-}row\ M]\ 0\ M)\ \$\$\ (i, j) = M\ \$\$\ (i, j)$
and $i \neq 0 \Longrightarrow (add\text{-}row\text{-}to\text{-}multiple\ a\ [1..<dim\text{-}row\ M]\ 0\ M)\ \$\$\ (i, j) = (a * M\$\$(0, j)) + M\$\$(i,j)$
⟨*proof*⟩

lemma *add-all-cols-to-first*:
assumes $i < dim\text{-}row\ (M)$
assumes $j < dim\text{-}col\ (M)$
shows $j \neq 0 \Longrightarrow add\text{-}multiple\text{-}cols\ 1\ 0\ [1..<dim\text{-}col\ M]\ M\ \$\$\ (i, j) = M\ \$\$\ (i, j)$
and $j = 0 \Longrightarrow add\text{-}multiple\text{-}cols\ 1\ 0\ [1..<dim\text{-}col\ M]\ M\ \$\$\ (i, j) = (\sum l = 1..<dim\text{-}col\ M.\ M\ \$\$(i,l)) + M\$\$(i,0)$
⟨*proof*⟩

Lemmas on the determinant of the matrix under extended row/column operations

lemma *add-row-to-multiple-carrier*:
$A \in carrier\text{-}mat\ n\ n \Longrightarrow add\text{-}row\text{-}to\text{-}multiple\ a\ ks\ l\ A \in carrier\text{-}mat\ n\ n$
⟨*proof*⟩

lemma *add-col-to-multiple-carrier*:
$A \in carrier\text{-}mat\ n\ n \Longrightarrow add\text{-}col\text{-}to\text{-}multiple\ a\ ks\ l\ A \in carrier\text{-}mat\ n\ n$
⟨*proof*⟩

lemma *add-multiple-rows-carrier*:
$A \in carrier\text{-}mat\ n\ n \Longrightarrow add\text{-}multiple\text{-}rows\ a\ k\ ls\ A \in carrier\text{-}mat\ n\ n$
⟨*proof*⟩

lemma *add-multiple-cols-carrier*:
$A \in carrier\text{-}mat\ n\ n \Longrightarrow add\text{-}multiple\text{-}cols\ a\ k\ ls\ A \in carrier\text{-}mat\ n\ n$
⟨*proof*⟩

lemma *add-row-to-multiple-det*:
assumes $l \notin set\ ks$ **and** $l < n$ **and** $A \in carrier\text{-}mat\ n\ n$
shows $det\ (add\text{-}row\text{-}to\text{-}multiple\ a\ ks\ l\ A) = det\ A$
⟨*proof*⟩

lemma *add-col-to-multiple-det*:
assumes $l \notin set\ ks$ **and** $l < n$ **and** $A \in carrier\text{-}mat\ n\ n$
shows $det\ (add\text{-}col\text{-}to\text{-}multiple\ a\ ks\ l\ A) = det\ A$
⟨*proof*⟩

lemma *add-multiple-cols-det*:
assumes $k \notin set\ ls$ **and** $\bigwedge l.\ l \in set\ ls \Longrightarrow l < n$ **and** $A \in carrier\text{-}mat\ n\ n$
shows $det\ (add\text{-}multiple\text{-}cols\ a\ k\ ls\ A) = det\ A$
⟨*proof*⟩

lemma *add-multiple-rows-det*:
assumes $k \notin$ *set ls* **and** $\bigwedge l.\ l \in$ *set ls* $\Longrightarrow l < n$ **and** $A \in$ *carrier-mat n n*
shows *det* (*add-multiple-rows a k ls A*) = *det A*
⟨*proof*⟩

6.2 Rank and Linear Independence

abbreviation *rank v M* ≡ *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*::{*conjugatable-ordered-field*} *mat*
assumes *A* ∈ *carrier-mat n m*
assumes *B* ∈ *carrier-mat m nc*
shows *rank n* (*A* ∗ *B*) ≤ *min* (*rank n A*) (*rank m B*)
⟨*proof*⟩

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* :: {*conjugatable-ordered-field*}) *mat*
assumes *M* ∈ *carrier-mat x y*
assumes *vec-space.rank x* (*M*∗ *M*^{T}) = *x*
shows *x* ≤ *y*
⟨*proof*⟩

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* :: {*conjugatable-ordered-field*}) *mat*
assumes *M* ∈ *carrier-mat x y*
assumes *det* (*M*∗ *M*^{T}) ≠ *0*
shows *x* ≤ *y*
⟨*proof*⟩

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 $A :: 'a\ vec\ set$
assumes $A \subseteq carrier\text{-}vec\ n$
assumes *lin-indpt A*
shows $card\ A \leq dim$
⟨*proof*⟩

lemma *lin-indpt-dim-col-lt-dim*:
fixes $A :: 'a\ mat$
assumes $A \in carrier\text{-}mat\ n\ nc$
assumes *distinct* (*cols A*)
assumes *lin-indpt* (*set* (*cols A*))
shows $nc \leq dim$
⟨*proof*⟩

lemma *lin-comb-imp-lin-dep-fin*:
fixes $A :: 'a\ vec\ set$
assumes *finite A*
assumes $A \subseteq carrier\text{-}vec\ n$
assumes $lincomb\ c\ A = 0_v\ n$
assumes $\exists\ a.\ a \in A \wedge c\ a \neq 0$
shows *lin-dep A*
⟨*proof*⟩

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 $set\ (A) \subseteq carrier\text{-}vec\ n$
assumes *distinct A*
assumes $\bigwedge f.\ lincomb\text{-}list\ (\lambda i.\ f\ (A\ !\ i))\ A = 0_v\ n \Longrightarrow \forall\, v \in (set\ A).\ f\ v = 0$
shows *lin-indpt* (*set A*)
⟨*proof*⟩

lemma *lin-comb-mat-lin-indpt*:
fixes $A :: 'a\ vec\ list$

assumes *set* (*A*) ⊆ *carrier-vec n*
assumes *distinct A*
assumes ⋀ *f*. *mat-of-cols n A* ∗$_v$ *vec* (*length A*) (λ*i*. *f* (*A* ! *i*)) = 0$_v$ *n* ⟹ ∀ *v*∈ (*set A*). *f v* = *0*
shows *lin-indpt* (*set A*)
⟨*proof*⟩

lemma *lin-comb-mat-lin-indpt-vs*:
fixes *A* :: *'a vec list*
assumes *set* (*A*) ⊆ *carrier-vec n*
assumes *distinct A*
assumes ⋀ *f*. *mat-of-cols n A* ∗$_v$ *vec* (*length A*) (λ*i*. *f* (*A* ! *i*)) = 0$_v$ *n* ⟹ ∀ *v*∈ (*set A*). *f v* = *0*
shows *lin-indpt-vs* (*set A*)
⟨*proof*⟩

end

7.2 Linear Bound Argument Lemmas

Three general representations of the linear bound argument, requiring a direct fact of linear independence onthe 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* ⊆ *carrier-vec nr*
assumes *vec-space.lin-indpt-vs nr A*
shows *card A* ≤ *nr*
⟨*proof*⟩

lemma *lin-bound-arg-general-list*:
fixes *xs* ::(*'a* :: {*field*})*vec list*
assumes *distinct xs*
assumes (*set xs*) ⊆ *carrier-vec nr*
assumes *vec-space.lin-indpt-vs nr* (*set xs*)
shows *length* (*xs*) ≤ *nr*
⟨*proof*⟩

lemma *lin-bound-arg-general*:
fixes *A* ::(*'a* :: {*field*}) *mat*
assumes *distinct* (*cols A*)
assumes *A* ∈ *carrier-mat nr nc*
assumes *vec-space.lin-indpt-vs nr* (*set* (*cols A*))
shows *nc* ≤ *nr*
⟨*proof*⟩

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 $A \in carrier\text{-}mat\ nr\ nc$
assumes $\bigwedge f.\ A *_v vec\ nc\ (\lambda i.\ f\ (col\ A\ i)) = 0_v\ nr \Longrightarrow \forall v \in (set\ (cols\ A)).\ f\ v = 0$
shows $nc \leq nr$
⟨*proof*⟩

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 $A :: ('a :: \{field\})\ mat$
assumes *distinct* $(cols\ A)$
assumes $A \in carrier\text{-}mat\ nr\ nc$
assumes $\bigwedge f.\ vec\ nr\ (\lambda i.\ \sum\ j \in \{0..<nc\}\ .\ f\ (col\ A\ j) * (col\ A\ j)\ \$\ i) = 0_v\ nr \Longrightarrow$
$\forall v \in (set\ (cols\ A)).\ f\ v = 0$
shows $nc \leq nr$
⟨*proof*⟩

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 $i <$ *dim-row* M
assumes $j <$ *dim-col* M

shows $i = 0 \Longrightarrow j = 0 \Longrightarrow$ (*add-row-to-multiple* (-1) [*1..<dim-row M*] *0 M*) $$ $(i, j) =$ (*int* r) — top left elem
and $i \neq 0 \Longrightarrow j = 0 \Longrightarrow$ (*add-row-to-multiple* (-1) [*1..<dim-row M*] *0 M*) $$ $(i, j) =$ (*int* Λ) $-$ (*int* r) — first column ex. 1
and $i = 0 \Longrightarrow j \neq 0 \Longrightarrow$ (*add-row-to-multiple* (-1) [*1..<dim-row M*] *0 M*) $$ $(i, j) =$ (*int* Λ) — first row ex. 1
and $i \neq 0 \Longrightarrow j \neq 0 \Longrightarrow i = j \Longrightarrow$ (*add-row-to-multiple* (-1) [*1..<dim-row M*] *0 M*) $$ $(i, j) =$ (*int* r) $-$ (*int* Λ) — diagonal ex. 1
and $i \neq 0 \Longrightarrow j \neq 0 \Longrightarrow i \neq j \Longrightarrow$ (*add-row-to-multiple* (-1) [*1..<dim-row M*] *0 M*) $$ $(i, j) = 0$ — everything else
⟨*proof*⟩

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 $i = 0 \Longrightarrow j = 0 \Longrightarrow$ *add-multiple-cols 1 0* [*1..<dim-col M*] *M* $$ $(i, j) =$ (*int* r) + (*int* Λ) $*$ (v $-$ *1*) — top left element
and $i = 0 \Longrightarrow j \neq 0 \Longrightarrow$ *add-multiple-cols 1 0* [*1..<dim-col M*] *M* $$ $(i, j) =$ (*int* Λ) — top row
and $i \neq 0 \Longrightarrow i = j \Longrightarrow$ *add-multiple-cols 1 0* [*1..<dim-col M*] *M* $$ $(i, j) =$ (*int* r) $-$ (*int* Λ) — Diagonal
and $i \neq 0 \Longrightarrow i \neq j \Longrightarrow$ *add-multiple-cols 1 0* [*1..<dim-col M*] *M* $$ $(i, j) = 0$ — Everything else
⟨*proof*⟩

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*)
⟨*proof*⟩

Find the determinant of the NN^T matrix using transformed matrix values

lemma *determinant-inc-mat-square*: *det* $(N * N^T) =$ (r + Λ $*$ (v $-$ *1*))$*$ (r $-$ Λ)^(v $-$ *1*)
⟨*proof*⟩

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*⟩

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 $c :: nat \Rightarrow real$
assumes $b \geq 2$
assumes $m > 0$
assumes $j' < b$
assumes $0 = (\sum j \in \{0..<b\} . (c\ j)\hat{}2 * ((card\ (\mathcal{B}s\ !\ j)) - (int\ m))) +$
$m * ((\sum j \in \{0..<b\} . c\ j)\hat{}2)$
shows $c\ j' = 0$
⟨*proof*⟩

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*⟩

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 $\bigwedge bl.\ bl \in\# \mathcal{B} \Longrightarrow$ *incomplete-block bl*
— i.e. not a super trivial design with only complete blocks
shows $v \leq b$
⟨*proof*⟩

corollary *general-nonuniform-fishers-comp*:
assumes $\Lambda > 0$
assumes *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*⟩

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* ⇒ *int mat* **where**
to-int-mat ≡ *map-mat to-int-mod-ring*

abbreviation *to-int-vec* :: *'a* :: *finite mod-ring vec* ⇒ *int vec* **where**
to-int-vec ≡ *map-vec to-int-mod-ring*

interpretation *of-int-mod-ring-hom-sr*: *semiring-hom of-int-mod-ring*
⟨*proof*⟩

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*⟩

lemma *mat-hom-inj*: *map-mat hom A* = *map-mat hom B* ⟹ *A* = *B*
⟨*proof*⟩

lemma *vec-hom-inj*: *map-vec hom v* = *map-vec hom w* ⟹ *v* = *w*
⟨*proof*⟩

lemma *vec-hom-set-distinct-iff*:
fixes *xs* :: *'a vec list*
shows *distinct xs* ⟷ *distinct* (*map* (*map-vec hom*) *xs*)
⟨*proof*⟩
end

9.1 Basic Mod Context

locale *mat-mod* = **fixes** *m* :: *int*
assumes *non-triv-m*: *m* > *1*
begin

First define the mod operations on vectors

definition *vec-mod* :: *int vec* ⇒ *int vec* **where**
vec-mod v ≡ *map-vec* (λ *x* . *x mod m*) *v*

lemma *vec-mod-dim*[*simp*]: *dim-vec* (*vec-mod v*) = *dim-vec v*
⟨*proof*⟩

lemma *vec-mod-index*[*simp*]: *i* < *dim-vec v* ⟹ (*vec-mod v*) $ *i* = (*v* $ *i*) *mod m*
⟨*proof*⟩

lemma *vec-mod-index-same*[*simp*]: *i* < *dim-vec v* ⟹ *v* $ *i* < *m* ⟹ *v* $ *i* ≥ *0*
⟹ (*vec-mod v*) $ *i* = *v* $ *i*
⟨*proof*⟩

lemma *vec-setI2*: *i* < *dim-vec v* ⟹ *v* $ *i* ∈ set_v *v*
⟨*proof*⟩

lemma *vec-mod-eq*: set_v *v* ⊆ {*0*..<*m*} ⟹ *vec-mod v* = *v*
⟨*proof*⟩

lemma *vec-mod-set-vec-same*:set_v *v* ⊆ {*0*..<*m*} ⟹ set_v (*vec-mod v*) = set_v *v*
⟨*proof*⟩

Define the mod operation on matrices

definition *mat-mod* :: *int mat* ⇒ *int mat* **where**
mat-mod M ≡ *map-mat* (λ *x*. *x mod m*) *M*

lemma *mat-mod-dim*[*simp*]: *dim-row* (*mat-mod M*) = *dim-row M dim-col* (*mat-mod M*) = *dim-col M*
⟨*proof*⟩

lemma *mat-mod-index* [*simp*]: *i* < *dim-row M* ⟹ *j* < *dim-col M* ⟹ (*mat-mod M*) $$ (*i*, *j*) = (*M* $$ (*i*, *j*)) *mod m*
⟨*proof*⟩

lemma *mat-mod-index-same*[*simp*]: *i* < *dim-row M* ⟹ *j* < *dim-col M* ⟹ *M* $$ (*i*, *j*) < *m* ⟹
M $$ (*i*, *j*) ≥ *0* ⟹ *mat-mod M* $$ (*i*, *j*) = *M* $$ (*i*, *j*)
⟨*proof*⟩

lemma *elements-matI2*: *i* < *dim-row A* ⟹ *j* < *dim-col A* ⟹ *A* $$ (*i*, *j*) ∈ *elements-mat A*
⟨*proof*⟩

lemma *mat-mod-elements-in*:
assumes *x* ∈ *elements-mat M*
shows *x mod m* ∈ *elements-mat* (*mat-mod M*)
⟨*proof*⟩

lemma *mat-mod-elements-map*:
assumes $x \in$ *elements-mat* M
shows *elements-mat* (*mat-mod* M) = (λ x. x *mod* m) ‘ (*elements-mat* M)
⟨*proof*⟩

lemma *mat-mod-eq-cond*:
assumes *elements-mat* $M \subseteq \{0..<m\}$
shows *mat-mod* $M = M$
⟨*proof*⟩

lemma *mat-mod-eq-elements-cond*: *elements-mat* $M \subseteq \{0..<m\} \Longrightarrow$ *elements-mat* (*mat-mod* M) = *elements-mat* M
⟨*proof*⟩

lemma *mat-mod-vec-mod-row*: $i <$ *dim-row* $A \Longrightarrow$ *row* (*mat-mod* A) i = *vec-mod* (*row* A i)
⟨*proof*⟩

lemma *mat-mod-vec-mod-col*: $j <$ *dim-col* $A \Longrightarrow$ *col* (*mat-mod* A) j = *vec-mod* (*col* A j)
⟨*proof*⟩

lemma *count-vec-mod-eq*: $set_v\ v \subseteq \{0..<m\} \Longrightarrow$ *count-vec* v x = *count-vec* (*vec-mod* v) x
⟨*proof*⟩

lemma *elems-mat-setv-row-0m*: $i <$ *dim-row* $M \Longrightarrow$ *elements-mat* $M \subseteq \{0..<m\} \Longrightarrow set_v$ (*row* M i) $\subseteq \{0..<m\}$
⟨*proof*⟩

lemma *elems-mat-setv-col-0m*: $j <$ *dim-col* $M \Longrightarrow$ *elements-mat* $M \subseteq \{0..<m\} \Longrightarrow set_v$ (*col* M j) $\subseteq \{0..<m\}$
⟨*proof*⟩

lemma *mat-mod-count-row-eq*: $i <$ *dim-row* $M \Longrightarrow$ *elements-mat* $M \subseteq \{0..<m\} \Longrightarrow$
count-vec (*row* (*mat-mod* M) i) x = *count-vec* (*row* M i) x
⟨*proof*⟩

lemma *mat-mod-count-col-eq*: $j <$ *dim-col* $M \Longrightarrow$ *elements-mat* $M \subseteq \{0..<m\} \Longrightarrow$

count-vec (*col* (*mat-mod* M) j) x = *count-vec* (*col* M j) x
⟨*proof*⟩

lemma *mod-mat-one*: *mat-mod* (1_m n) = (1_m n)
⟨*proof*⟩

lemma *mod-mat-zero*: *mat-mod* (0_m nr nc) = (0_m nr nc)

⟨*proof*⟩

lemma *vec-mod-unit*: *vec-mod* (*unit-vec n i*) = (*unit-vec n i*)
⟨*proof*⟩

lemma *vec-mod-zero*: *vec-mod* (0_v *n*) = (0_v *n*)
⟨*proof*⟩

lemma *mat-mod-cond-iff*: *elements-mat M* ⊆ {*0*..<*m*} ⟹ *P M* ⟷ *P* (*mat-mod M*)
⟨*proof*⟩

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* ⟨*proof*⟩

definition *M* :: *int* ⇒ *int* **where** *M x* = *x mod m*

lemma *M-0*[*simp*]: *M 0* = *0*
⟨*proof*⟩

lemma *M-M*[*simp*]: *M* (*M x*) = *M x*
⟨*proof*⟩

lemma *M-plus*[*simp*]: *M* (*M x* + *y*) = *M* (*x* + *y*) *M* (*x* + *M y*) = *M* (*x* + *y*)
⟨*proof*⟩

lemma *M-minus*[*simp*]: *M* (*M x* − *y*) = *M* (*x* − *y*) *M* (*x* − *M y*) = *M* (*x* − *y*)
⟨*proof*⟩

lemma *M-times*[*simp*]: *M* (*M x* ∗ *y*) = *M* (*x* ∗ *y*) *M* (*x* ∗ *M y*) = *M* (*x* ∗ *y*)
⟨*proof*⟩

lemma *M-1*[*simp*]: *M 1* = *1* ⟨*proof*⟩

lemma *M-sum*: *M* (*sum* (λ *x*. *M* (*f x*)) *A*) = *M* (*sum f A*)
⟨*proof*⟩

definition *inv-M* :: *int* ⇒ *int* **where**
inv-M x = (*if x* + *x* ≤ *m then x else x* − *m*)

lemma *M-inv-M-id*[*simp*]: *M* (*inv-M x*) = *M x*
⟨*proof*⟩

definition *M-Rel* :: *int* ⇒ *'a mod-ring* ⇒ *bool*
where *M-Rel x x'* ≡ (*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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *eq-M-Rel*[*transfer-rule*]: (*M-Rel* ===> *M-Rel* ===> (=)) (λ *x y*. *M x* = *M y*) (=)
⟨*proof*⟩

lemma *one-M-Rel*[*transfer-rule*]: *M-Rel 1 1*
⟨*proof*⟩

lemma *zero-M-Rel*[*transfer-rule*]: *M-Rel 0 0*
⟨*proof*⟩

lemma *M-to-int-mod-ring*: *M* (*to-int-mod-ring* (*x* :: *'a mod-ring*)) = *to-int-mod-ring x*
⟨*proof*⟩

lemma *right-total-M-Rel*[*transfer-rule*]: *right-total M-Rel*
⟨*proof*⟩

lemma *left-total-M-Rel*[*transfer-rule*]: *left-total M-Rel*
⟨*proof*⟩

lemma *bi-total-M-Rel*[*transfer-rule*]: *bi-total M-Rel*
⟨*proof*⟩

lemma *to-int-mod-ring-of-int-M*: *to-int-mod-ring* (*of-int x* :: *'a mod-ring*) = *M x*
⟨*proof*⟩

lemma *UNIV-M-Rel*[*transfer-rule*]: *rel-set M-Rel* {*0..*<*m*} *UNIV*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *to-int-vec-index*[*simp*]: *i* < *dim-vec* *v* ⟹ (*to-int-vec* *v* $*i*) = *to-int-mod-ring* (*v* $*i*)
⟨*proof*⟩

lemma *eq-dim-row-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> (=)) *dim-row* *dim-row*

⟨*proof*⟩

lemma *lt-dim-row-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> (=) ===> (=)) (λ *M* *i*. *i* < *dim-row* *M*) (λ *M* *i*. *i* < *dim-row* *M*)
⟨*proof*⟩

lemma *eq-dim-col-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> (=)) *dim-col* *dim-col*
⟨*proof*⟩

lemma *lt-dim-col-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> (=) ===> (=)) (λ *M* *j*. *j* < *dim-col* *M*) (λ *M* *j*. *j* < *dim-col* *M*)
⟨*proof*⟩

lemma *eq-dim-vec-MV-Rel*[*transfer-rule*]: (*MV-Rel* ===> (=)) *dim-vec* *dim-vec*
⟨*proof*⟩

lemma *lt-dim-vec-MV-Rel*[*transfer-rule*]: (*MV-Rel* ===> (=) ===> (=)) (λ *v* *j*. *j* < *dim-vec* *v*) (λ *v* *j*. *j* < *dim-vec* *v*)

⟨*proof*⟩

lemma *eq-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> *MM-Rel* ===> (=)) (λ *f f′* . *mat-mod f* = *mat-mod f′*) (=)
⟨*proof*⟩

lemma *eq-MV-Rel*[*transfer-rule*]: (*MV-Rel* ===> *MV-Rel* ===> (=)) (λ *v v′* . *vec-mod v* = *vec-mod v′*) (=)
⟨*proof*⟩

lemma *index-MV-Rel*[*transfer-rule*]: (*MV-Rel* ===> (=) ===> *M-Rel*)
(λ *v i*. *if i* < *dim-vec v then v* $ *i else 0*) (λ *v i*. *if i* < *dim-vec v then v* $ *i else 0*)
⟨*proof*⟩

lemma *index-MM-Rel*[*transfer-rule*]: (*MM-Rel* ===> (=) ===> (=) ===> *M-Rel*)
(λ *M i j*. *if* (*i* < *dim-row M* ∧ *j* < *dim-col M*) *then M* $$ (*i*, *j*) *else 0*)
(λ *M i j*. *if* (*i* < *dim-row M* ∧ *j* < *dim-col M*) *then M* $$ (*i*, *j*) *else 0*)
⟨*proof*⟩

lemma *index-MM-Rel-explicit*:
assumes *MM-Rel N N′*
assumes *i* < *dim-row N j* < *dim-col N*
shows (*N* $$ (*i*, *j*)) *mod m* = *to-int-mod-ring* (*N′* $$ (*i*, *j*))
⟨*proof*⟩

lemma *one-MV-Rel*[*transfer-rule*]: *MV-Rel* (*unit-vec n i*) (*unit-vec n i*)
⟨*proof*⟩

lemma *one-MM-Rel*[*transfer-rule*]: *MM-Rel* (1_m *n*) (1_m *n*)
⟨*proof*⟩

lemma *zero-MM-Rel*[*transfer-rule*]: *MM-Rel* (0_m *nr nc*) (0_m *nr nc*)
⟨*proof*⟩

lemma *zero-MV-Rel*[*transfer-rule*]: *MV-Rel* (0_v *n*) (0_v *n*)
⟨*proof*⟩

lemma *right-unique-MV-Rel*[*transfer-rule*]: *right-unique MV-Rel*
⟨*proof*⟩

lemma *right-unique-MM-Rel*[*transfer-rule*]: *right-unique MM-Rel*
⟨*proof*⟩

lemma *mod-to-int-mod-ring*: (*to-int-mod-ring* (*x* :: *′a mod-ring*)) *mod m* = *to-int-mod-ring x*
⟨*proof*⟩

lemma *mat-mod-to-int-mat*: *mat-mod* (*to-int-mat* (*N* :: *'a mod-ring mat*)) = *to-int-mat N*
⟨*proof*⟩

lemma *vec-mod-to-int-vec*: *vec-mod* (*to-int-vec* (*v* :: *'a mod-ring vec*)) = *to-int-vec v*
⟨*proof*⟩

lemma *right-total-MM-Rel*[*transfer-rule*]: *right-total MM-Rel*
⟨*proof*⟩

lemma *right-total-MV-Rel*[*transfer-rule*]: *right-total MV-Rel*
⟨*proof*⟩

lemma *to-int-mod-ring-of-int-mod*: *to-int-mod-ring* (*of-int x* :: *'a mod-ring*) = *x mod m*
⟨*proof*⟩

lemma *vec-mod-v-representative*: *vec-mod v* = *to-int-vec* (*map-vec of-int v* :: *'a mod-ring vec*)
⟨*proof*⟩

lemma *mat-mod-N-representative*: *mat-mod N* = *to-int-mat* (*map-mat of-int N* :: *'a mod-ring mat*)
⟨*proof*⟩

lemma *left-total-MV-Rel*[*transfer-rule*]: *left-total MV-Rel*
⟨*proof*⟩

lemma *left-total-MM-Rel*[*transfer-rule*]: *left-total MM-Rel*
⟨*proof*⟩

lemma *bi-total-MV-Rel*[*transfer-rule*]: *bi-total MV-Rel*
⟨*proof*⟩

lemma *bi-total-MM-Rel*[*transfer-rule*]: *bi-total MM-Rel*
⟨*proof*⟩

lemma *domain-MV-rel*[*transfer-domain-rule*]: *Domainp MV-Rel* = (λ *f*. *True*)
⟨*proof*⟩

lemma *domain-MM-rel*[*transfer-domain-rule*]: *Domainp MM-Rel* = (λ *f*. *True*)
⟨*proof*⟩

lemma *mem-MV-Rel*[*transfer-rule*]:
(*MV-Rel* ===> *rel-set MV-Rel* ===> (=)) (λ *x Y*. ∃ *y* ∈ *Y*. *vec-mod x* = *vec-mod y*) (∈)
⟨*proof*⟩

lemma *mem-MM-Rel*[*transfer-rule*]:
(*MM-Rel* ===> *rel-set MM-Rel* ===> (=)) (λ *x Y*. ∃ *y* ∈ *Y*. *mat-mod x* = *mat-mod y*) (∈)
⟨*proof*⟩

lemma *conversep-MM-Rel-OO-MM-Rel* [*simp*]: *MM-Rel*$^{-1-1}$ *OO MM-Rel* = (=)
⟨*proof*⟩

lemma *MM-Rel-OO-conversep-MM-Rel* [*simp*]: *MM-Rel OO MM-Rel*$^{-1-1}$ = (λ *M M'*. *mat-mod M* = *mat-mod M'*)
⟨*proof*⟩

lemma *conversep-MM-Rel-OO-eq-m* [*simp*]: *MM-Rel*$^{-1-1}$ *OO* (λ *M M'*. *mat-mod M* = *mat-mod M'*) = *MM-Rel*$^{-1-1}$
⟨*proof*⟩

lemma *eq-m-OO-MM-Rel* [*simp*]: (λ *M M'* . *mat-mod M* = *mat-mod M'*) *OO MM-Rel* = *MM-Rel*
⟨*proof*⟩

lemma *eq-mset-MM-Rel* [*transfer-rule*]:
(*rel-mset MM-Rel* ===> *rel-mset MM-Rel* ===> (=)) (*rel-mset* (λ *M M'* . *mat-mod M* = *mat-mod M'*)) (=)
⟨*proof*⟩

lemma *vec-mset-MV-Rel*[*transfer-rule*]:
(*MV-Rel* ===> (=)) (λ *v*. *vec-mset* (*vec-mod v*)) (λ *v*. *image-mset* (*to-int-mod-ring*) (*vec-mset v*))
⟨*proof*⟩

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*⟩

lemma *MM-Rel-MV-Rel-row*: *MM-Rel A B* ⟹ *i* < *dim-row A* ⟹ *MV-Rel* (*row A i*) (*row B i*)
⟨*proof*⟩

lemma *MM-Rel-MV-Rel-col*: *MM-Rel A B* ⟹ *j* < *dim-col A* ⟹ *MV-Rel* (*col A j*) (*col B j*)
⟨*proof*⟩

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*: *proper-inc-mat* (*mat-mod N*) $\longleftrightarrow$ *proper-inc-mat N*
⟨*proof*⟩

lemma *mat-mod-rep-num-eq*: $i <$ *dim-row N* $\Longrightarrow$ *elements-mat N* $\subseteq \{0..<m\}$ $\Longrightarrow$
mat-rep-num (*mat-mod N*) *i* = *mat-rep-num N i*
⟨*proof*⟩

lemma *mat-point-index-eq*: *elements-mat N* $\subseteq \{0..<m\} \Longrightarrow$
mat-point-index (*mat-mod N*) *I* = *mat-point-index N I*
⟨*proof*⟩

lemma *mod-mat-inter-num-eq*: *elements-mat N* $\subseteq \{0..<m\} \Longrightarrow$
mat-inter-num (*mat-mod N*) *j1 j2* = *mat-inter-num N j1 j2*
⟨*proof*⟩

lemma *mod-mat-block-size*: *elements-mat N* $\subseteq \{0..<m\} \Longrightarrow$ *mat-block-size* (*mat-mod N*) *j* = *mat-block-size N j*
⟨*proof*⟩

lemma *mat-mod-non-empty-col-iff*: *elements-mat M* $\subseteq \{0..<m\} \Longrightarrow$
non-empty-col (*mat-mod M*) *j* $\longleftrightarrow$ *non-empty-col M j*
⟨*proof*⟩
end

context *mat-mod-type*
begin

lemma *mat-rep-num-MM-Rel*:
assumes *MM-Rel A B*
assumes *i* < *dim-row A*
shows *mat-rep-num* (*mat-mod A*) *i* = *mat-rep-num B i*
⟨*proof*⟩

lemma *mat-block-size-MM-Rel*:
assumes *MM-Rel A B*
assumes $j < dim\text{-}col\ A$
shows *mat-block-size* (*mat-mod A*) *j* = *mat-block-size B j*
⟨*proof*⟩

lemma *mat-inter-num-MM-Rel*:
assumes *MM-Rel A B*
assumes $j1 < dim\text{-}col\ A$ $j2 < dim\text{-}col\ B$
shows *mat-inter-num* (*mat-mod A*) *j1 j2* = *mat-inter-num B j1 j2*
⟨*proof*⟩

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* = (*lift-01-mat*) *N*
⟨*proof*⟩

lemma *to-int-mod-ring-lift-01-eq*:
assumes *zero-one-matrix N*
shows *to-int-mat N* = (*lift-01-mat*) *N*
⟨*proof*⟩

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*: $bl \in\# \mathcal{B} \Longrightarrow odd\ (card\ bl)$
and *even-inters*: $bl1 \in\# \mathcal{B} \Longrightarrow bl2 \in\# (\mathcal{B} - \{\#bl1\#\}) \Longrightarrow even\ (bl1 \mid\cap\mid bl2)$
begin

lemma *odd-town-no-repeat-clubs*: *distinct-mset* $\mathcal{B}$
⟨*proof*⟩

lemma *odd-blocks-mat-block-size*: $j < dim\text{-}col\ N \Longrightarrow odd\ (mat\text{-}block\text{-}size\ N\ j)$
⟨*proof*⟩

lemma *odd-block-size-mod-2*:
  assumes *CARD*('b::*prime-card*) = *2*
  assumes $j < \mathrm{b}$
  shows *of-nat* (*card* ($\mathcal{B}s$! j)) = (*1* :: 'b *mod-ring*)
⟨*proof*⟩

lemma *valid-indices-block-min*: *j1* < *dim-col N* ⟹ *j2* < *dim-col N* ⟹ *j1* ≠ *j2* ⟹ $\mathrm{b} \geq$ *2*
  ⟨*proof*⟩

lemma *even-inter-mat-intersections*: *j1* < *dim-col N* ⟹ *j2* < *dim-col N* ⟹ *j1* ≠ *j2*
  ⟹ *even* (*mat-inter-num N j1 j2*)
  ⟨*proof*⟩

lemma *even-inter-mod-2*:
  assumes *CARD*('b::*prime-card*) = *2*
  assumes $i < \mathrm{b}$ **and** *jlt*: $j < \mathrm{b}$ **and** *ne*: $i \neq j$
  shows *of-nat* (($\mathcal{B}s$! i) |∩| ($\mathcal{B}s$! j)) = (*0* :: 'b *mod-ring*)
⟨*proof*⟩

end

The odd town locale must be simple by definition

sublocale *odd-town* ⊆ *ordered-simple-design*
  ⟨*proof*⟩

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-card*) = *2*
  shows $\mathrm{b} \leq \mathrm{v}$
⟨*proof*⟩

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.