Hypergraph Colouring Bounds using Probabilistic Methods

Chelsea Edmonds and Lawrence C. Paulson

October 16, 2023

Abstract

This library includes several example applications of the probabilistic method for combinatorics to establish bounds for hypergraph colourings. This focuses on *Property B* — the existence of a two-colouring of the vertex set of a hypergraph. A stricter bound was formalised using the Lovász local lemma, which in turn required a surprisingly complex proof of the mutual independence principle for hypergraph edges that is often omitted on paper. The formalisation uncovered several interesting examples of circular intuition on proofs involving independence on paper. The formalisation is based on the textbook proofs from Alon and Spencer's famous textbook, *The Probabilistic Method*[1], further supported by [3]. The mutual independence principle proof is inspired by the less precise proof provided in Molloy and Reed's textbook on graph colourings [2], as it was omitted in all other sources. Additionally, this library demonstrates how locales can be used to establish a reusable probability space framework, thus minimizing the setup required for future formalisations requiring a probability space on numerous possible properties around an incidence system's vertex set.

Contents

1 Hypergraph Colourings

theory *Hypergraph-Colourings* **imports** *Card-Partitions.Card-Partitions*
Hypergraph-Basics.Hypergraph-Variations HOL−Library.Extended-Real
Girth-Chromatic.Girth-Chromatic-Misc
begin

1.1 Function and Number extras

lemma *surj-PiE*:
  assumes $f \in A \rightarrow_E B$
  assumes f ‘ $A = B$
  assumes $b \in B$
  obtains a **where** $a \in A$ **and** $f\ a = b$
  using *assms*(*2*) *assms*(*3*) **by** *blast*

lemma *Stirling-gt-0*: $n \geq k \Longrightarrow k \neq 0 \Longrightarrow$ *Stirling* $n\ k > 0$
  apply (*induct n k rule*: *Stirling.induct*, *simp-all*)
  using *Stirling-same Suc-lessI gr0I zero-neq-one* **by** (*metis Suc-leI*)

lemma *card-partition-on-ne*:
  assumes *card* $A \geq n$ $n \neq 0$
  shows $\{P.$ *partition-on* $A\ P \wedge$ *card* $P = n\} \neq \{\}$
proof −
  have *finite* A **using** *assms*
    using *card-eq-0-iff* **by** *force*
  then have *card* $\{P.$ *partition-on* $A\ P \wedge$ *card* $P = n\} > 0$
    using *card-partition-on Stirling-gt-0 assms* **by** *fastforce*
  thus *?thesis* **using** *card.empty*
    by *fastforce*
qed

lemma *enat-lt-INF*:
  fixes $f :: 'a \Rightarrow$ *enat*
  assumes (*INF* $x \in S.\ f\ x) < t$
  obtains x **where** $x \in S$ **and** $f\ x < t$
proof −
  from *assms* **have** (*INF* $x \in S.\ f\ x) \neq$ *top*
    by *fastforce*
  then obtain y **where** $y \in S$ **and** $f\ y =$ (*INF* $x \in S\ .\ f\ x$) **using** *enat-in-INF*
    by *metis*
  thus *?thesis* **using** *assms*

by (*simp add*: *that*)
qed

1.2 Basic Definitions

context *hypergraph*
begin

Edge colourings - using older partition approach

definition *edge-colouring* :: (*'a hyp-edge* ⇒ *colour*) ⇒ *colour set* ⇒ *bool* **where**
edge-colouring f C ≡ *partition-on-mset E* {# {#*h* ∈# *E* . *f h* = *c*#} . *c* ∈# (*mset-set C*)#}

definition *proper-edge-colouring* :: (*'a hyp-edge* ⇒ *colour*) ⇒ *colour set* ⇒ *bool* **where**
proper-edge-colouring f C ≡ *edge-colouring f C* ∧
(∀ *e1 e2 c*. *e1* ∈# *E* ∧ *e2* ∈# *E* − {#*e1*#} ∧ *c* ∈ *C* ∧ *f e1* = *c* ∧ *f e2* = *c* ⟶ *e1* ∩ *e2* = {})

A vertex colouring function with no edge monochromatic requirements

abbreviation *vertex-colouring* :: (*'a* ⇒ *colour*) ⇒ *nat* ⇒ *bool* **where**
vertex-colouring f n ≡ *f* ∈ $\mathcal{V}$ $\rightarrow_E$ {*0..<n*}

lemma *vertex-colouring-union*:
  assumes *vertex-colouring f n*
  shows ⋃ {{*v* ∈ $\mathcal{V}$. *f v* = *c*} |*c*. *c* ∈ {*0..<n*}} = $\mathcal{V}$
  using *assms* **by** (*intro subset-antisym subsetI*) *blast+*

lemma *vertex-colouring-disj*:
  assumes *vertex-colouring f n*
  assumes *p* ∈ {{*v* ∈ $\mathcal{V}$. *f v* = *c*} |*c*. *c* ∈ {*0..<n*}}
  assumes *p'* ∈ {{*v* ∈ $\mathcal{V}$. *f v* = *c*} |*c*. *c* ∈ {*0..<n*}}
  assumes *p* ≠ *p'*
  shows *p* ∩ *p'* = {}
proof (*rule ccontr*)
  assume *a*: *p* ∩ *p'* ≠ {}
  obtain *c c'* **where** *c* ∈ {*0..<n*} **and** *c'* ∈ {*0..<n*} *p* = {*v* ∈ $\mathcal{V}$. *f v* = *c*} **and**
      p' = {*v* ∈ $\mathcal{V}$. *f v* = *c*} **and** *c* ≠ *c'*
    using *assms*(*4*) *assms*(*2*) *assms*(*3*) *a* **by** *blast*
  then obtain *v* **where** *v* ∈ $\mathcal{V}$ **and** *v* ∈ *p* **and** *v* ∈ *p'* **using** *a* **by** *blast*
  then show *False* **using** *Fun.apply-inverse*
    using ‹*p* = {*v* ∈ $\mathcal{V}$. *f v* = *c*}› ‹*p'* = {*v* ∈ $\mathcal{V}$. *f v* = *c*}› *assms*(*4*) **by** *blast*
qed

lemma *vertex-colouring-n0*: $\mathcal{V}$ ≠ {} ⟹ ¬ *vertex-colouring f 0*
  by *auto*

lemma *vertex-colouring-image*: *vertex-colouring f n* ⟹ *v* ∈ $\mathcal{V}$ ⟹ *f v* ∈ {*0..<n*}

using *funcset-mem* **by** *blast*

lemma *vertex-colouring-image-edge-ss*: *vertex-colouring f n* $\Longrightarrow$ *e* $\in\#$ *E* $\Longrightarrow$ *f* ' *e* $\subseteq$ {*0..<n*}
using *wellformed vertex-colouring-image* **by** *blast*

lemma *vertex-colour-edge-map-ne*: *vertex-colouring f n* $\Longrightarrow$ *e* $\in\#$ *E* $\Longrightarrow$ *f* ' *e* $\neq$ {}
using *blocks-nempty* **by** *simp*

lemma *vertex-colouring-ne*: *vertex-colouring f n* $\Longrightarrow$ *f u* $\neq$ *f v* $\Longrightarrow$ *u* $\neq$ *v*
by *auto*

lemma *vertex-colour-one*: $\mathcal{V} \neq$ {} $\Longrightarrow$ *vertex-colouring f 1* $\Longrightarrow$ *v* $\in \mathcal{V}$ $\Longrightarrow$ *f v* = (*0*::*nat*)
using *atLeastLessThan-iff less-one vertex-colouring-image* **by** *simp*

lemma *vertex-colour-one-alt*:
assumes $\mathcal{V} \neq$ {}
shows *vertex-colouring f* (*1*::*nat*) $\longleftrightarrow$ *f* = (λ *v* $\in \mathcal{V}$. *0*::*nat*)
proof (*intro iffI*)
assume *a*: *vertex-colouring f 1*
show *f* = ($\lambda v\in\mathcal{V}$. *0*::*nat*)
proof (*rule ccontr*)
assume *f* $\neq$ ($\lambda v\in\mathcal{V}$. *0*)
then have $\exists$ *v*$\in$ $\mathcal{V}$. *f v* $\neq$ *0*
using *a* **by** *auto*
thus *False* **using** *vertex-colour-one assms a*
by *meson*
qed
next
show *f* = ($\lambda v\in\mathcal{V}$. *0*) $\Longrightarrow$ *f* $\in \mathcal{V} \rightarrow_E$ {*0..<1*} **using** *PiE-eq-singleton* **by** *auto*
qed

lemma *vertex-colouring-partition*:
assumes *vertex-colouring f n*
assumes *f* ' $\mathcal{V}$ = {*0..<n*}
shows *partition-on* $\mathcal{V}$ { {*v* $\in \mathcal{V}$. *f v* = *c*} | *c*. *c*$\in$ {*0..<n*}}
proof (*intro partition-onI*)
fix *p* **assume** *p* $\in$ {{*v* $\in \mathcal{V}$. *f v* = *c*} |*c*. *c* $\in$ {*0..<n*}}
then obtain *c* **where** *peq*: *p* = {*v* $\in \mathcal{V}$. *f v* = *c*} **and** *cin*: *c* $\in$ {*0..<n*} **by** *blast*
have *f* $\in \mathcal{V} \rightarrow_E$ {*0..<n*} **using** *assms*(*1*) **by** *presburger*
then obtain *v* **where** *v* $\in \mathcal{V}$ **and** *f v* = *c*
using *surj-PiE*[*of f* $\mathcal{V}$ {*0..<n*} *c*] *cin assms*(*2*) **by** *auto*
then show *p* $\neq$ {} **using** *peq* **by** *auto*
next
show $\bigcup$ {{*v* $\in \mathcal{V}$. *f v* = *c*} |*c*. *c* $\in$ {*0..<n*}} = $\mathcal{V}$ **using** *vertex-colouring-union assms* **by** *auto*
next

show $\bigwedge p\ p'.\ p \in \{\{v \in \mathcal{V}.\ f\ v = c\}\ |c.\ c \in \{0..<n\}\} \Longrightarrow$
$p' \in \{\{v \in \mathcal{V}.\ f\ v = c\}\ |c.\ c \in \{0..<n\}\} \Longrightarrow p \neq p' \Longrightarrow p \cap p' = \{\}$
by *auto*
qed

1.3 Monochromatic Edges

definition *mono-edge* :: $('a \Rightarrow colour) \Rightarrow 'a\ hyp\text{-}edge \Rightarrow bool$ **where**
mono-edge $f\ e \equiv \exists\ c.\ \forall\ v \in e.\ f\ v = c$

lemma *mono-edge-single*:
assumes $e \in\#\ E$
shows *mono-edge* $f\ e \longleftrightarrow$ *is-singleton* $(f\ `\ e)$
unfolding *mono-edge-def*
proof (*intro iffI*)
assume $\exists\ c.\ \forall\ v{\in}e.\ f\ v = c$
then obtain c **where** *ceq*: $\bigwedge\ v\ .\ v \in e \Longrightarrow f\ v = c$ **by** *blast*
then have $f\ `\ e = \{c\}$ **using** *image-singleton assms blocks-nempty* **by** *metis*
then show *is-singleton* $(f\ `\ e)$ **by** *simp*
next
assume *is-singleton* $(f\ `\ e)$
then obtain c **where** $f\ `\ e = \{c\}$ **by** (*meson is-singletonE*)
then show $\exists\ c.\ \forall\ v{\in}e.\ f\ v = c$ **by** *auto*
qed

definition *mono-edge-col* :: $('a \Rightarrow colour) \Rightarrow 'a\ hyp\text{-}edge \Rightarrow colour \Rightarrow bool$ **where**
mono-edge-col $f\ e\ c \equiv \forall\ v \in e.\ f\ v = c$

lemma *mono-edge-colI*: $(\bigwedge\ v.\ v \in e \Longrightarrow f\ v = c) \Longrightarrow$ *mono-edge-col* $f\ e\ c$
unfolding *mono-edge-col-def* **by** *simp*

lemma *mono-edge-colD*: *mono-edge-col* $f\ e\ c \Longrightarrow (\bigwedge\ v.\ v \in e \Longrightarrow f\ v = c)$
unfolding *mono-edge-col-def* **by** *simp*

lemma *mono-edge-alt-col*: *mono-edge* $f\ e \equiv \exists\ c.$ *mono-edge-col* $f\ e\ c$
unfolding *mono-edge-def mono-edge-col-def* **by** *auto*

1.4 Proper colourings

A proper vertex colouring brings in the monochromatic edge decision. Note that this allows for a colouring of up to n colours, not precisely n colours

definition *is-proper-colouring* :: $('a \Rightarrow colour) \Rightarrow nat \Rightarrow bool$ **where**
is-proper-colouring $f\ n \equiv$ *vertex-colouring* $f\ n \wedge (\forall\ e \in\#\ E.\ \forall\ c \in \{0..<n\}.\ f\ `\ e \neq \{c\})$

lemma *is-proper-colouring-alt*: *is-proper-colouring* $f\ n \longleftrightarrow$ *vertex-colouring* $f\ n \wedge(\forall\ e \in\#\ E.\ \neg$ *is-singleton* $(f\ `\ e))$
unfolding *is-proper-colouring-def* **using** *vertex-colouring-image-edge-ss*
by (*auto*) (*metis insert-subset is-singleton-def*)

```
lemma is-proper-colouring-alt2: is-proper-colouring f n ⟷ vertex-colouring f n
∧(∀ e ∈# E. ¬ mono-edge f e)
 unfolding is-proper-colouring-def using vertex-colouring-image-edge-ss mono-edge-single

    is-proper-colouring-alt is-proper-colouring-def by force

lemma is-proper-colouringI[intro]: vertex-colouring f n ⟹ (⋀ e .e ∈# E ⟹
    ¬ is-singleton (f ‘ e)) ⟹ is-proper-colouring f n
  using is-proper-colouring-alt by simp

lemma is-proper-colouringI2[intro]: vertex-colouring f n ⟹ (⋀ e .e ∈# E ⟹ ¬
mono-edge f e)
    ⟹ is-proper-colouring f n
  using is-proper-colouring-alt2 by simp

lemma is-proper-colouring-n0: 𝒱 ≠ {} ⟹ ¬ is-proper-colouring f 0
  unfolding is-proper-colouring-def using vertex-colouring-n0 by auto

lemma is-proper-colouring-empty:
  assumes 𝒱 = {}
  shows is-proper-colouring f n ⟷ f = (λ x . undefined)
  unfolding is-proper-colouring-def using PiE-empty-domain assms
  using vertex-colouring-image-edge-ss by fastforce

lemma is-proper-colouring-n1:
  assumes 𝒱 ≠ {} E ≠ {#}
  shows ¬ is-proper-colouring f 1
proof (rule ccontr)
  assume ¬ ¬ is-proper-colouring f 1
  then have vc: vertex-colouring f 1 and em: (∀ e ∈# E. ¬ mono-edge f e)
    using is-proper-colouring-alt2 by auto
  then obtain e where ein: e ∈# E using assms by blast
  have f ∈ 𝒱 →E {0} using vc by auto
  then have ∀ v∈ 𝒱. f v = 0
    by simp
  then have ∀ v ∈ e. f v = 0  using wellformed‹e ∈# E› by blast
  then have mono-edge f e using ein mono-edge-def by auto
  then show False using em ein by simp
qed

lemma (in fin-hypergraph) is-proper-colouring-image-card:
  assumes 𝒱 ≠ {} E ≠ {#}
  assumes n > 1
  assumes is-proper-colouring f n
  shows card (f ‘ 𝒱) > 1
proof (rule ccontr)
  assume ¬ 1 < card (f ‘ 𝒱)
  then have a: card (f ‘ 𝒱) = 1
```

using *assms* **by** (*meson card-0-eq finite-imageI finite-sets image-is-empty less-one linorder-neqE-nat*)
then obtain *c* **where** *ceq*: f ‘ $\mathcal{V} = \{c\}$
using *card-1-singletonE* **by** *blast*
then obtain *e* **where** *ein*: $e \in\# E$ **using** *assms*(*2*) **by** *blast*
then have *ss*: $e \subseteq \mathcal{V}$ **using** *wellformed* **by** *auto*
then have $\forall\ v \in e.\ f\ v = c$ **using** *ceq*
by *blast*
then have *mono-edge f e* **using** *ein mono-edge-def* **by** *auto*
then show *False* **using** *is-proper-colouring-alt2 ein*
using *assms*(*4*) **by** *blast*
qed

More monochromatic edges

lemma *no-monochomatic-is-colouring*:
assumes $\forall\ e \in\# E\ .\ \neg$ *mono-edge f e*
assumes *vertex-colouring f n*
shows *is-proper-colouring f n*
using *assms mono-edge-single is-proper-colouringI* **by** (*auto*)

lemma *ex-monochomatic-not-colouring*:
assumes $\exists\ e \in\# E\ .$ *mono-edge f e*
assumes *vertex-colouring f n*
shows $\neg$ *is-proper-colouring f n*
using *assms*(*1*) **by** (*simp add*: *mono-edge-single is-proper-colouring-alt*)

lemma *mono-edge-colour-obtain*:
assumes *mono-edge f e*
assumes *vertex-colouring f n*
assumes $e \in\# E$
obtains *c* **where** $c \in \{0..<n\}$ **and** *mono-edge-col f e c*
proof –
have *ss*: f ‘ $e \subseteq \{0..<n\}$ **using** *vertex-colouring-image-edge-ss assms* **by** *simp*
obtain *c* **where** *all*: $\forall\ v \in e.\ f\ v = c$ **using** *mono-edge-def*
using *assms*(*1*) **by** *fastforce*
have f ‘ $e \neq \{\}$ **using** *blocks-nempty* **by** (*simp add*: *assms*(*3*))
then have $c \in f$ ‘ e **using** *all*
by *fastforce*
thus *?thesis* **using** *ss that all mono-edge-col-def* **by** *blast*
qed

Complete proper colourings - i.e. when n colours are required

definition *is-complete-proper-colouring*:: $('a \Rightarrow colour) \Rightarrow nat \Rightarrow bool$ **where**
is-complete-proper-colouring f n $\equiv$ *is-proper-colouring f n* $\wedge\ f$ ‘ $\mathcal{V} = \{0..<n\}$

lemma *is-complete-proper-colouring-part*:
assumes *is-complete-proper-colouring f n*
shows *partition-on* $\mathcal{V}$ $\{\ \{v \in \mathcal{V}\ .\ f\ v = c\}\ |\ c.\ c \in \{0..<n\}\}$
using *vertex-colouring-partition assms is-complete-proper-colouring-def is-proper-colouring-def*

by *auto*

lemma *is-complete-proper-colouring-n0*: $\mathcal{V} \neq \{\} \Longrightarrow \neg$ *is-complete-proper-colouring f 0*
unfolding *is-complete-proper-colouring-def* **using** *is-proper-colouring-n0* **by** *simp*

lemma *is-complete-proper-colouring-n1*:
assumes $\mathcal{V} \neq \{\}$ $E \neq \{\#\}$
shows $\neg$ *is-complete-proper-colouring f 1*
unfolding *is-complete-proper-colouring-def* **using** *is-proper-colouring-n1 assms* **by** *simp*

lemma (**in** *fin-hypergraph*) *is-proper-colouring-reduce*:
assumes *is-proper-colouring f n*
obtains f' **where** *is-complete-proper-colouring* f' (*card* (f ‘ $\mathcal{V}$))
proof (*cases* f ‘ $\mathcal{V} = \{0..<(n::nat)\}$)
case *True*
then have *card* (f ‘ $\mathcal{V}$) = n **by** *simp*
then show *?thesis* **using** *is-complete-proper-colouring-def assms*
using *True that* **by** *auto*
next
case *False*
obtain g :: *nat* ⇒ *nat* **where** *bij*: *bij-betw* g (f ‘ $\mathcal{V}$) $\{0..<(card\ (f$ ‘ $\mathcal{V}))\}$
using *ex-bij-betw-finite-nat finite-sets* **by** *blast*
let $?f' = \lambda\ x\ .\ if\ x \in \mathcal{V}\ then\ (g \circ f)\ x\ else\ undefined$
have *img*: $?f'$ ‘ $\mathcal{V} = \{0..<card\ (f$ ‘ $\mathcal{V})\}$ **using** *bij bij-betw-imp-surj-on image-comp*
by (*smt* (*verit*) *image-cong*)
have *is-proper-colouring* $?f'$ (*card* (f ‘ $\mathcal{V}$))
proof (*intro is-proper-colouringI*)
show *vertex-colouring* $?f'$ (*card* (f ‘ $\mathcal{V}$))
using *img* **by** *auto*
next
fix e **assume** *ein*: $e \in\#\ E$
then have *ns*: $\neg$ *is-singleton* (f ‘ e) **using** *assms is-proper-colouring-alt* **by** *blast*
have *ss*: (f ‘ e) $\subseteq$ (f ‘ $\mathcal{V}$) **using** *wellformed*
by (*simp add*: *ein image-mono*)
have $e \subseteq \mathcal{V}$ **using** *wellformed ein* **by** *simp*
then have $?f'$ ‘ $e = g$ ‘ (f ‘ e) **by** *auto*
then show $\neg$ *is-singleton* ($?f'$ ‘ e) **using** *bij ns ss bij-betw-singleton-image* **by** *metis*
qed
then show *?thesis* **using** *is-complete-proper-colouring-def img*
by (*meson that*)
qed

lemma (**in** *fin-hypergraph*) *two-colouring-is-complete*:
assumes $\mathcal{V} \neq \{\}$
assumes $E \neq \{\#\}$

assumes *is-proper-colouring f 2*
shows *is-complete-proper-colouring f 2*
proof −
have *gt*: *card* $(f \text{ ` } \mathcal{V}) > 1$ **using** *is-proper-colouring-image-card assms*
using *one-less-numeral-iff semiring-norm*(*76*) **by** *blast*
have $f \in \mathcal{V} \rightarrow_E \{0..<2\}$ **using** *is-proper-colouring-def assms*(*3*) **by** *auto*
then have $f \text{ ` } \mathcal{V} \subseteq \{0..<2\}$ **by** *blast*
then have *card* $(f \text{ ` } \mathcal{V}) = 2$
by (*metis Nat.le-diff-conv2 gt leI less-one less-zeroE nat-1-add-1 order-antisym-conv*

subset-eq-atLeast0-lessThan-card zero-less-diff)
thus *?thesis* **using** *is-complete-proper-colouring-def assms*
by (*metis* ‹$f \text{ ` } \mathcal{V} \subseteq \{0..<2\}$› *plus-nat.add-0 subset-card-intvl-is-intvl*)
qed

1.5 n vertex colourings

definition *is-n-colourable* :: *nat* ⇒ *bool* **where**
is-n-colourable n ≡ ∃ *f* . *is-proper-colouring f n*

definition *is-n-edge-colourable* :: *nat* ⇒ *bool* **where**
is-n-edge-colourable n ≡ ∃ *f C* . *card C* = *n* ⟶ *proper-edge-colouring f C*

definition *all-n-vertex-colourings* :: *nat* ⇒ (*'a* ⇒ *colour*) *set* **where**
all-n-vertex-colourings n ≡ {*f* . *vertex-colouring f n*}

notation *all-n-vertex-colourings* (($\mathcal{C}$^) [*502*] *500*)

lemma *all-n-vertex-colourings-alt*: $\mathcal{C}^n = \mathcal{V} \rightarrow_E \{0..<n\}$
unfolding *all-n-vertex-colourings-def* **by** *auto*

lemma *vertex-colourings-empty*: $\mathcal{V} \neq \{\} \Longrightarrow$ *all-n-vertex-colourings 0* = {}
unfolding *all-n-vertex-colourings-def* **using** *vertex-colouring-n0*
by *simp*

lemma (**in** *fin-hypergraph*) *vertex-colourings-fin* : *finite* $(\mathcal{C}^n)$
using *all-n-vertex-colourings-alt finite-PiE finite-sets* **by** (*metis finite-atLeastLessThan*)

lemma (**in** *fin-hypergraph*) *count-vertex-colourings*: *card* $(\mathcal{C}^n) = n$ ^ *horder*
using *all-n-vertex-colourings-alt card-funcsetE*
by (*metis card-atLeastLessThan finite-sets minus-nat.diff-0*)

lemma *vertex-colourings-nempty*:
assumes *card* $\mathcal{V} \geq n$
assumes $n \neq 0$
shows $\mathcal{C}^n \neq \{\}$
using *all-n-vertex-colourings-alt assms*
by (*simp add*: *PiE-eq-empty-iff*)

lemma *vertex-colourings-one*:
assumes $\mathcal{V} \neq \{\}$
shows $\mathcal{C}^1 = \{\lambda\ v\in \mathcal{V}\ .\ 0\}$
using *vertex-colour-one-alt assms*
by (*simp add*: *all-n-vertex-colourings-def*)

lemma *mono-edge-set-union*:
assumes $e \in\#\ E$
shows $\{f \in \mathcal{C}^n\ .\ \textit{mono-edge}\ f\ e\} = (\bigcup c \in \{0..<n\}.\ \{f \in \mathcal{C}^n\ .\ \textit{mono-edge-col}\ f\ e\ c\})$
proof (*intro subset-antisym subsetI*)
fix *g* **assume** *a1*: $g \in \{f \in \mathcal{C}^n.\ \textit{mono-edge}\ f\ e\}$
then have *vertex-colouring g n* **using** *all-n-vertex-colourings-def* **by** *blast*
then obtain *c* **where** $c \in \{0..<n\}$ **and** *mono-edge-col g e c* **using** *a1 assms mono-edge-colour-obtain*
by *blast*
then show $g \in (\bigcup c\in\{0..<n\}.\ \{f \in \mathcal{C}^n.\ \textit{mono-edge-col}\ f\ e\ c\})$
using ‹*vertex-colouring g n*› *all-n-vertex-colourings-def* **by** *auto*
next
fix *h* **assume** $h \in (\bigcup c\in\{0..<n\}.\ \{f \in \mathcal{C}^n.\ \textit{mono-edge-col}\ f\ e\ c\})$
then obtain *c* **where** $c \in \{0..<n\}$ **and** $h \in \{f \in \mathcal{C}^n.\ \textit{mono-edge-col}\ f\ e\ c\}$
by *blast*
then show $h \in \{f \in \mathcal{C}^n.\ \textit{mono-edge}\ f\ e\}$
using *mono-edge-alt-col* **by** *blast*
qed

end

Property B set up

abbreviation (**in** *hypergraph*) *has-property-B* :: *bool* **where**
has-property-B ≡ *is-n-colourable 2*

abbreviation *hyp-graph-order*:: *'a hyp-graph* ⇒ *nat* **where**
hyp-graph-order h ≡ *card* (*hyp-verts h*)

definition *not-col-n-uni-hyps*:: *nat* ⇒ *'a hyp-graph set*
where *not-col-n-uni-hyps n* ≡ { *h* . *fin-kuniform-hypergraph-nt* (*hyp-verts h*) (*hyp-edges h*) *n* ∧
¬ (*hypergraph.has-property-B* (*hyp-verts h*) (*hyp-edges h*)) }

definition *min-edges-colouring* :: *nat* ⇒ *'a itself* ⇒ *enat* **where**
min-edges-colouring n - ≡ *INF h* ∈ ((*not-col-n-uni-hyps n*) :: *'a hyp-graph set*) . *enat* (*size* (*hyp-edges h*))

lemma *obtains-min-edge-colouring*:
fixes *z* :: *'a itself*
assumes *min-edges-colouring n z* < *x*
obtains *h* :: *'a hyp-graph* **where** *h* ∈ *not-col-n-uni-hyps n* **and** *enat* (*size*

(*hyp-edges h*)) < *x*
proof –
have (*INF h* ∈ ((*not-col-n-uni-hyps n*) :: *'a hyp-graph set*) . *enat* (*size* (*hyp-edges h*))) < *x*
using *min-edges-colouring-def*[*of n z*] *assms* **by** *auto*
thus *?thesis* **using** *enat-lt-INF*[*of* λ *h*. *enat* (*size* (*hyp-edges h*)) *not-col-n-uni-hyps n x*]
using *that* **by** *blast*
qed

1.6 Alternate Partition Definition.

Note that the indexed definition should be used most of the time instead

context *hypergraph*
begin

definition *is-proper-colouring-part* :: *'a set set* ⇒ *bool* **where**
is-proper-colouring-part C ≡ *partition-on* 𝒱 *C* ∧ (∀ *c* ∈ *C*. ∀ *e* ∈# *E*. ¬ *e* ⊆ *c*)

definition *is-n-colourable-part* :: *nat* ⇒ *bool* **where**
is-n-colourable-part n ≡ ∃ *C* . *card C* = *n* ⟶ *is-proper-colouring-part C*

abbreviation *has-property-B-part* :: *bool* **where**
has-property-B-part ≡ *is-n-colourable-part 2*

definition *mono-edge-ss* :: *'a set set* ⇒ *'a hyp-edge* ⇒ *bool* **where**
mono-edge-ss C e ≡ ∃ *c* ∈ *C*. *e* ⊆ *c*

lemma *is-proper-colouring-partI*: *partition-on* 𝒱 *C* ⟹ (∀ *c* ∈ *C*. ∀ *e* ∈# *E*. ¬ *e* ⊆ *c*) ⟹
is-proper-colouring-part C
by (*simp add*: *is-proper-colouring-part-def*)

lemma *no-monochomatic-is-colouring-part*:
assumes ∀ *e* ∈# *E* . ¬ *mono-edge-ss C e*
assumes *partition-on* 𝒱 *C*
shows *is-proper-colouring-part C*
using *assms*(*1*) *mono-edge-ss-def* **by**(*intro is-proper-colouring-partI*) (*simp-all add*: *assms*)

lemma *ex-monochomatic-not-colouring-part*:
assumes ∃ *e* ∈# *E* . *mono-edge-ss C e*
assumes *partition-on* 𝒱 *C*
shows ¬ *is-proper-colouring-part C*
using *assms*(*1*) *mono-edge-ss-def is-proper-colouring-part-def* **by** *auto*

definition *all-n-vertex-colourings-part* :: *nat* ⇒ *'a set set set* **where**
all-n-vertex-colourings-part n ≡ {*C* . *partition-on* 𝒱 *C* ∧ *card C* = *n*}

lemma (**in** *fin-hypergraph*) *all-vertex-colourings-part-fin*: *finite* (*all-n-vertex-colourings-part* *n*)
unfolding *all-n-vertex-colourings-part-def is-proper-colouring-part-def*
using *finitely-many-partition-on finite-sets* **by** *fastforce*

lemma *all-vertex-colourings-part-nempty*: *card* $\mathcal{V} \geq n \Longrightarrow n \neq 0 \Longrightarrow$ *all-n-vertex-colourings-part* $n \neq \{\}$
unfolding *all-n-vertex-colourings-part-def* **using** *card-partition-on-ne* **by** *blast*

lemma *disjoint-family-on-colourings*:
assumes $e \in\# E$
shows *disjoint-family-on* (λ *c*. $\{f \in \mathcal{C}^n$. *mono-edge-col f e c*$\}$) $\{0..<n\}$
using *blocks-nempty mono-edge-col-def assms* **by** (*auto intro*: *disjoint-family-onI*)

end

end

2 Basic Probabilistic Method Application

This section establishes step (1) of the basic framework for incidence set systems, as well as some basic bounds on hypergraph colourings

theory *Basic-Bounds-Application* **imports** *Lovasz-Local.Basic-Method Hypergraph-Colourings*
begin

2.1 Probability Spaces for Incidence Set Systems

This is effectively step (1) of the formal framework for probabilistic method. Unlike stages (3) and (4), which were formalised in the Lovasz_Local_Lemma AFP entry, this stage required a formalisation of incidence set systems as well as the background probability space locales

A basic probability space for a point measure on a non-trivial structure

locale *vertex-fn-space* = *fin-hypersystem-vne* +
fixes F :: $'a$ *set* $\Rightarrow$ $'b$ *set*
fixes p :: $'b \Rightarrow$ *real*
assumes *ne*: $F\ \mathcal{V} \neq \{\}$
assumes *fin*: *finite* $(F\ \mathcal{V})$
assumes *pgte0*: $\bigwedge fv\ .\ fv \in F\ \mathcal{V} \Longrightarrow p\ fv \geq 0$
assumes *sump*: $(\sum x \in (F\ \mathcal{V})\ .\ p\ x) = 1$
begin

definition $\Omega \equiv F\ \mathcal{V}$

lemma *fin*-Ω: *finite* Ω
unfolding Ω-*def* **using** *fin* **by** *auto*

lemma *ne*-Ω: $\Omega \neq \{\}$

unfolding *Ω-def* **using** *ne* **by** *simp*

definition *M* = *point-measure* Ω *p*

lemma *space-eq*: *space M* = Ω
unfolding *M-def* *Ω-def* **by** (*simp add*: *space-point-measure*)

lemma *sets-eq*: *sets M* = *Pow* (Ω)
unfolding *M-def* **by** (*simp add*: *sets-point-measure*)

lemma *finite-event*: *A* ⊆ Ω ⟹ *finite A*
by (*simp add*: *finite-subset fin-Ω*)

lemma *emeasure-eq*: *emeasure M A* = (*if* (*A* ⊆ Ω) *then* (∑ *a*∈*A*. *p a*) *else 0*)
proof (*cases A* ⊆ Ω)
case *True*
then have *finite A* **using** *finite-event* **by** *auto*
moreover have *ennreal* (*sum p A*) = (∑ *a*∈*A*. *ennreal* (*p a*))
using *sum-ennreal pgte0 True* **by** (*simp add*: *subset-iff Ω-def*)
ultimately have *emeasure M A* = (∑ *a*∈*A*. *p a*)
using *emeasure-point-measure-finite2*[*of A* Ω *p*] *M-def*
using *True* **by** *presburger*
then show *?thesis* **using** *True* **by** *auto*
next
case *False*
then show *?thesis* **using** *emeasure-notin-sets sets-eq* **by** *auto*
qed

lemma *integrable-M*[*intro*, *simp*]: *integrable M* (*f*::- ⇒ *real*)
using *fin-Ω* **by** (*simp add*: *integrable-point-measure-finite M-def*)

lemma *borel-measurable-M*[*measurable*]: *f* ∈ *borel-measurable M*
unfolding *M-def* **by** *simp*

lemma *prob-space-M*: *prob-space M*
unfolding *M-def* **using** *fin-Ω ne-Ω pgte0 sump Ω-def*
by (*intro prob-space-point-measure*) (*simp-all*)

end

sublocale *vertex-fn-space* ⊆ *prob-space M*
using *prob-space-M* **.**

A uniform variation of the space

locale *vertex-fn-space-uniform* = *fin-hypersystem-vne* +
fixes *F* :: *'a set* ⇒ *'b set*
assumes *ne*: *F* 𝒱 ≠ {}
assumes *fin*: *finite* (*F* 𝒱)
begin

definition $\Omega U \equiv F\ \mathcal{V}$

definition $MU \equiv$ *uniform-count-measure* ΩU

end

sublocale *vertex-fn-space-uniform* $\subseteq$ *vertex-fn-space* $\mathcal{V}\ E\ F\ (\lambda x.\ 1\ /\ card\ \Omega U)$
 rewrites $\Omega = \Omega U$ **and** $M = MU$
proof (*unfold-locales*)
 show *1*: $F\ \mathcal{V} \neq \{\}$ **and** *2*: *finite* $(F\ \mathcal{V})$ **by** (*simp-all add*: *fin ne*)
 show *3*: $\bigwedge fv.\ fv \in F\ \mathcal{V} \Longrightarrow 0 \leq 1\ /\ real\ (card\ (\Omega U))$ **by** *auto*
 show *4*: $(\sum x \in F\ \mathcal{V}.\ 1\ /\ real\ (card\ \Omega U)) = 1$
  using *sum-constant ne fin* ΩU*-def* **by** *auto*
 interpret *vf*: *vertex-fn-space* $\mathcal{V}\ E\ F\ (\lambda x.\ 1\ /\ card\ (\Omega U))$
  using *1 2 3 4* **by** (*unfold-locales*)
 show $vf.\Omega = \Omega U$ **unfolding** $vf.\Omega$*-def* ΩU*-def* **by** *simp*
 show $vf.M = MU$ **unfolding** *vf.M-def* $vf.\Omega$*-def MU-def uniform-count-measure-def*
using ΩU*-def* **by** *auto*
qed

context *vertex-fn-space-uniform*
begin

lemma *emeasure-eq*: *emeasure* $MU\ A = (if\ (A \subseteq \Omega U)\ then\ ((card\ A)/card\ (\Omega U))$ *else* $0)$
 using *fin-*Ω *MU-def emeasure-uniform-count-measure*[*of* $\Omega U\ A$]
  sets-uniform-count-measure emeasure-notin-sets Pow-iff ennreal-0 **by** (*metis* (*full-types*))

lemma *measure-eq-valid*: $A \in events \Longrightarrow measure\ MU\ A = (card\ A)/card\ (\Omega U)$
 using *sets-eq* **by** (*simp add*: *MU-def* ΩU*-def fin measure-uniform-count-measure*)

lemma *expectation-eq*:
 shows $expectation\ f = (\sum\ x \in \Omega U.\ f\ x)\ /\ card\ \Omega U$
proof–
 have $(\bigwedge a.\ a \in \Omega U \Longrightarrow 0 \leq 1\ /\ real\ (card\ \Omega U))$
  using *fin-*Ω *ne-*Ω **by** *auto*
 moreover have $\bigwedge\ a.\ a \in \Omega U \Longrightarrow (1\ /\ real\ (card\ \Omega U)) *_R f\ a = 1\ /\ (card\ \Omega U) * f\ a$
  using *real-scaleR-def* **by** *simp*
 ultimately have $expectation\ f = (\sum\ x \in \Omega U.\ f\ x * (1\ /\ (card\ \Omega U)))$
 using *uniform-count-measure-def*[*of* ΩU] *lebesgue-integral-point-measure-finite*[*of* $\Omega U\ (\lambda\ x.\ 1\ /\ card\ \Omega U)\ f$]
  MU-def fin-Ω **by** *auto*
 then show *?thesis* **using** *sum-distrib-right*[*symmetric*, *of f 1* / (*card* ΩU) ΩU]
by *auto*
qed

end

A probability space over the full vertex set

locale *vertex-space* = *fin-hypersystem-vne* +
fixes $p :: 'a \Rightarrow real$
assumes *pgte0*: $\bigwedge fv . fv \in \mathcal{V} \Longrightarrow p\ fv \geq 0$
assumes *sump*: $(\sum x \in (\mathcal{V}) . p\ x) = 1$

sublocale *vertex-space* ⊆ *vertex-fn-space* $\mathcal{V}\ E\ \lambda\ i . i\ p$
rewrites $\Omega = \mathcal{V}$
proof (*unfold-locales*)
interpret *vertex-fn-space* $\mathcal{V}\ E\ \lambda\ i . i\ p$
by *unfold-locales* (*simp-all add*: *pgte0 sump V-nempty finite-sets*)
show $\Omega = \mathcal{V}$
using Ω-*def* **by** *simp*
qed (*simp-all add*: *pgte0 sump V-nempty finite-sets*)

A uniform variation of the probability space over the vertex set

locale *vertex-space-uniform* = *fin-hypersystem-vne*

sublocale *vertex-space-uniform* ⊆ *vertex-fn-space-uniform* $\mathcal{V}\ E\ \lambda\ i . i$
rewrites $\Omega U = \mathcal{V}$
proof (*unfold-locales*)
interpret *vertex-fn-space-uniform* $\mathcal{V}\ E\ \lambda\ i . i$
by *unfold-locales* (*simp-all add*: *V-nempty finite-sets*)
show $\Omega U = \mathcal{V}$ **unfolding** Ω*U-def* **by** *simp*
qed (*simp-all add*: *V-nempty finite-sets*)

A uniform probability space over a vertex subset

locale *vertex-ss-space-uniform* = *fin-hypersystem-vne* +
fixes *VS*
assumes *vs-ss*: $VS \subseteq \mathcal{V}$
assumes *ne-vs*: $VS \neq \{\}$
begin

lemma *finite-vs*: *finite VS*
using *vs-ss finite-subset finite-sets* **by** *auto*

end

sublocale *vertex-ss-space-uniform* ⊆ *vertex-fn-space-uniform* $\mathcal{V}\ E\ \lambda\ i.\ VS$
rewrites $\Omega = VS$
proof (*unfold-locales*)
interpret *vertex-fn-space-uniform* $\mathcal{V}\ E\ \lambda\ i . VS$
by *unfold-locales* (*simp-all add*: *ne-vs finite-vs*)
show $\Omega = VS$
using Ω-*def* **by** *simp*

qed (*simp-all add*: *ne-vs finite-vs*)

A non-uniform prob space over a vertex subset

locale *vertex-ss-space* = *fin-hypersystem-vne* +
fixes *VS*
assumes *vs-ss*: $VS \subseteq \mathcal{V}$
assumes *ne-vs*: $VS \neq \{\}$
fixes $p :: 'a \Rightarrow real$
assumes *pgte0*: $\bigwedge fv \,.\, fv \in VS \Longrightarrow p\ fv \geq 0$
assumes *sump*: $(\sum x \in (VS)\ .\ p\ x) = 1$
begin

lemma *finite-vs*: *finite VS*
using *vs-ss finite-subset finite-sets* **by** *auto*

end

sublocale *vertex-ss-space* ⊆ *vertex-fn-space* $\mathcal{V}$ *E* $\lambda\ i\ .\ VS\ p$
rewrites $\Omega = VS$
proof (*unfold-locales*)
interpret *vertex-fn-space* $\mathcal{V}$ *E* $\lambda\ i\ .\ VS\ p$
by *unfold-locales* (*simp-all add*: *pgte0 sump ne-vs finite-vs*)
show $\Omega = VS$
using *Ω-def* **by** *simp*
qed (*simp-all add*: *pgte0 sump ne-vs finite-vs*)

A uniform probability space over a property on the vertex set

locale *vertex-prop-space* = *fin-hypersystem-vne* +
fixes $P :: 'b\ set$
assumes *finP*: *finite P*
assumes *nempty-P*: $P \neq \{\}$

sublocale *vertex-prop-space* ⊆ *vertex-fn-space-uniform* $\mathcal{V}$ *E* $\lambda\ V.\ V \rightarrow_E P$
rewrites $\Omega U = \mathcal{V} \rightarrow_E P$
proof −
interpret *vertex-fn-space-uniform* $\mathcal{V}$ *E* $\lambda\ V.\ V \rightarrow_E P$
proof (*unfold-locales*)
show $\mathcal{V} \rightarrow_E P \neq \{\}$ **using** *finP V-nempty PiE-eq-empty-iff nempty-P* **by** *meson*
show *finite* $(\mathcal{V} \rightarrow_E P)$ **using** *finP finite-PiE finite-sets* **by** *meson*
qed
show *vertex-fn-space-uniform* $\mathcal{V}$ *E* $(\lambda V.\ V \rightarrow_E P)$ **by** *unfold-locales*
show $\Omega U = \mathcal{V} \rightarrow_E P$ **unfolding** *ΩU-def* **by** *simp*
qed

context *vertex-prop-space*
begin

lemma *prob-uniform-vertex-subset*:
assumes $b \in P$

assumes $d \subseteq \mathcal{V}$
shows *prob* $\{f \in \Omega\ .\ (\forall\ v \in d\ .f\ v = b)\} = 1/((card\ P)\ powi\ (card\ d))$
using *finP nempty-P V-nempty finite-sets MU-def ΩU-def*
by (*simp add*: *assms*(*1*) *assms*(*2*) *prob-uniform-ex-fun-space*)

lemma *prob-uniform-vertex*:
assumes $b \in P$
assumes $v \in \mathcal{V}$
shows *prob* $\{f \in \Omega U\ .\ f\ v = b\} = 1/(card\ P)$
proof −
have *prob* $\{f \in \Omega U\ .\ f\ v = b\} = card\ \{f \in \Omega U\ .\ f\ v = b\}/card\ \Omega U$
using *measure-eq-valid sets-eq* **by** *auto*
then show *?thesis*
using *card-PiE-val-indiv-eq*[*of* $\mathcal{V}$ *b P v*] *Ω-def finite-sets finP nempty-P assms*
by *auto*
qed

end

A uniform vertex colouring space

locale *vertex-colour-space* = *fin-hypergraph-nt* +
fixes *n* :: *nat*
assumes *n-lt-order*: $n \leq horder$
assumes *n-not-zero*: $n \neq 0$

sublocale *vertex-colour-space* ⊆ *vertex-prop-space* $\mathcal{V}$ *E* $\{0..<n\}$
rewrites $\Omega U = \mathcal{C}^n$
proof −
have $\{0..<n\} \neq \{\}$ **using** *n-not-zero* **by** *simp*
then interpret *vertex-prop-space* $\mathcal{V}$ *E* $\{0..<n\}$
by (*unfold-locales*) (*simp-all*)
show *vertex-prop-space* $\mathcal{V}$ *E* $\{0..<n\}$ **by** (*unfold-locales*)
show $\Omega U = \mathcal{C}^n$
using *Ω-def all-n-vertex-colourings-alt* **by** *auto*
qed

This probability space contains several useful lemmas on basic vertex colouring probabilities (and monochromatic edges), which are facts that are typically either not proven, or have very short proofs on paper

context *vertex-colour-space*
begin

lemma *colour-set-event*: $\{f \in \mathcal{C}^n.\ mono\text{-}edge\text{-}col\ f\ e\ c\} \in events$
using *sets-eq* **by** *simp*

lemma *colour-functions-event*: $(\lambda\ c.\ \{f \in \mathcal{C}^n\ .\ mono\text{-}edge\text{-}col\ f\ e\ c\})$ ‘ $\{0..<n\} \subseteq$ *events*
using *sets-eq* **by** *blast*

lemma *prob-vertex-colour*: $v \in \mathcal{V} \Longrightarrow c \in \{0..<n\} \Longrightarrow$ *prob* $\{f \in \mathcal{C}^n \text{ . } f\ v = c\} =$ $1/n$
using *prob-uniform-vertex* **by** *simp*

lemma *prob-edge-colour*:
assumes $e \in\# E\ c \in \{0..<n\}$
shows *prob* $\{f \in \mathcal{C}^n$. *mono-edge-col f e c*$\} = 1/(n$ *powi* (*card e*))
proof –
have *card* $\{0..<n\} = n$ **by** *simp*
moreover have $\mathcal{C}^n = \mathcal{V} \rightarrow_E \{0..<n\}$ **using** *all-n-vertex-colourings-alt* **by** *blast*
moreover have $\{0..<n\} \neq \{\}$ **using** *n-not-zero* **by** *simp*
ultimately show *?thesis* **using** *prob-uniform-ex-fun-space*[*of* $\mathcal{V}$ - $\{0..<n\}$ *e*] *n-not-zero*
finite-sets wellformed assms **by** (*simp add*: *MU-def V-nempty mono-edge-col-def*)
qed

lemma *prob-monochromatic-edge-inv*:
assumes $e \in\# E$
shows *prob*$\{f \in \mathcal{C}^n$. *mono-edge f e*$\} = 1/(n$ *powi* (*int* (*card e*) $-$ *1*))
proof –
have *finite* $\{0..<n\}$**by** *auto*
then have *prob* $\{f \in \mathcal{C}^n$. *mono-edge f e*$\} = (\sum c \in \{0..<n\}$. *prob* $\{f \in \mathcal{C}^n$. *mono-edge-col f e c*$\})$
using *finite-measure-finite-Union*[*of* $\{0..<n\}$ (λ *c* . $\{f \in \mathcal{C}^n$. *mono-edge-col f e c*$\}$)]
disjoint-family-on-colourings colour-functions-event mono-edge-set-union assms **by** *auto*
also have ... $= n/(n$ *powi* (*int* (*card e*))) **using** *prob-edge-colour assms* **by** *simp*
also have ... $= n/(n*$ (*n powi* ((*int* (*card e*)) $-$ *1*))) **using** *n-not-zero*
power-int-commutes power-int-minus-mult **by** (*metis of-nat-0-eq-iff*)
finally show *?thesis* **using** *n-not-zero* **by** *simp*
qed

lemma *prob-monochromatic-edge*:
assumes $e \in\# E$
shows *prob*$\{f \in \mathcal{C}^n$. *mono-edge f e*$\} = n$ *powi* (*1* $-$ *int* (*card e*))
using *prob-monochromatic-edge-inv assms n-not-zero* **by** (*simp add*: *power-int-diff*)

lemma *prob-monochromatic-edge-bound*:
assumes $e \in\# E$
assumes $\bigwedge$ *e*. $e \in\# E \Longrightarrow$ *card* $e \geq k$
assumes $k > 0$
shows *prob*$\{f \in \mathcal{C}^n$. *mono-edge f e*$\} \leq 1/(($*real n*) *powi* $(k-1))$
proof –
have (*int* (*card* (*e*)) $-$ *1*) $\geq k - 1$ **using** *assms*(*3*) *assms*(*1*)
using *assms*(*2*) *int-ops*(*6*) *of-nat-0-less-iff of-nat-mono* **by** *fastforce*
then have ((*n* :: *real*) *powi* (*int* (*card* (*e*)) $-$ *1*)) $\geq$ (*n powi* ($k - 1$))

using *power-int-increasing*[*of k* − *1* (*int* (*card* (*e*)) − *1*) *n*] *n-not-zero* **by** *linarith*

moreover have *prob*($\{f \in \mathcal{C}^n$. *mono-edge f e*$\}$) = *1*/(*n powi* (*int* (*card* (*e*)) − *1*))
using *prob-monochromatic-edge-inv assms*(*1*) **by** *simp*
ultimately show *?thesis* **using** *frac-le zero-less-power-int n-not-zero*
by (*smt* (*verit*) *less-imp-of-nat-less of-nat-0-eq-iff of-nat-0-le-iff of-nat-1 of-nat-diff zle-int*)
qed

end

2.2 More Hypergraph Colouring Results

context *fin-hypergraph-nt*
begin

lemma *not-proper-colouring-edge-mono*: $\{f \in \mathcal{C}^n$. ¬ *is-proper-colouring f n*$\}$ = ($\bigcup$ *e* ∈ (*set-mset E*). $\{f \in \mathcal{C}^n$. *mono-edge f e*$\}$)
proof −
have $\{f \in \mathcal{C}^n$. ¬ *is-proper-colouring f n*$\}$ = $\{f \in \mathcal{C}^n$. ∃ *e* ∈ *set-mset E* . *mono-edge f e*$\}$
using *ex-monochomatic-not-colouring no-monochomatic-is-colouring*
by (*metis* (*mono-tags*, *lifting*) *all-n-vertex-colourings-alt*)
then show *?thesis* **using** *Union-exists* **by** *simp*
qed

lemma *proper-colouring-edge-mono*: $\{f \in \mathcal{C}^n$. *is-proper-colouring f n*$\}$ = ($\bigcap$ *e* ∈ (*set-mset E*). $\{f \in \mathcal{C}^n$. ¬ *mono-edge f e*$\}$)
proof −
have $\{f \in \mathcal{C}^n$. *is-proper-colouring f n*$\}$ = $\{f \in \mathcal{C}^n$. ∀ *e* ∈ *set-mset E* . ¬ *mono-edge f e*$\}$
using *is-proper-colouring-alt2 all-n-vertex-colourings-alt* **by** *auto*
moreover have *set-mset E* ≠ {} **using** *E-nempty* **by** *simp*
ultimately show *?thesis* **using** *Inter-forall* **by** *auto*
qed

lemma *proper-colouring-edge-mono-compl*: $\{f \in \mathcal{C}^n$. *is-proper-colouring f n*$\}$ = ($\bigcap$ *e* ∈ (*set-mset E*). $\mathcal{C}^n$ − $\{f \in \mathcal{C}^n$. *mono-edge f e*$\}$)
using *proper-colouring-edge-mono* **by** *auto*

lemma *event-is-proper-colouring*:
assumes $g \in \mathcal{C}^n$
assumes *g* ∉($\bigcup$ *e* ∈ (*set-mset E*). $\{f \in \mathcal{C}^n$. *mono-edge f e*$\}$)
shows *is-proper-colouring g n*
proof −
have $\bigwedge$ *e*. *e* ∈# *E* ⟹ ¬ *mono-edge g e* **using** *assms*
by *blast*
then show *?thesis* **using** *assms*(*1*) *all-n-vertex-colourings-def* **by** (*auto*)

qed

end

2.3 The Basic Application

The comments below show the basic framework steps

context *fin-kuniform-hypergraph-nt*
begin
proposition *erdos-propertyB*:
  assumes *size E* < (*2*^(*k* − *1*))
  assumes *k* > *0*
  shows *has-property-B*
proof −
  — (1) Set up the probability space: "Colour V randomly with two colours"
  interpret *P*: *vertex-colour-space* $\mathcal{V}$ *E 2*
    by *unfold-locales* (*auto simp add*: *order-ge-two*)
  — (2) define the event to avoid - monochromatic edges
  define *A* **where** *A* ≡(λ *e*. {*f* ∈ $\mathcal{C}^2$. *mono-edge f e*})
  — (3) Calculation 2: Have Pr (of Ae for any e) ≤Sum over e (Pr (A e)) < 1
  have (∑ *e* ∈ *set-mset E*. *P.prob* (*A e*)) < *1*
  proof −
    have *int k* − *1* = *int* (*k* − *1*) **using** *assms* **by** *linarith*
    then have *card* (*set-mset E*) < *2 powi* (*int k* − *1*) **using** *card-size-set-mset*[*of E*] *assms* **by** *simp*
    then have (∑ *e* ∈ (*set-mset E*). *P.prob* (*A e*)) < *2 powi* (*int k* − *1*) ∗ *2 powi* (*1* − *int k*)
      unfolding *A-def* **using** *P.prob-monochromatic-edge uniform assms*(*1*) **by** *simp*
    moreover have ((*2* :: *real*) *powi* ((*int k*) − *1*)) ∗ (*2 powi* (*1* − (*int k*))) = *1*
      using *power-int-add*[*of 2 int k* − *1 1*− *int k*] **by** *force*
    ultimately show *?thesis* **using** *power-int-add*[*of 2 int k* − *1 1*− *int k*] **by** *simp*
  qed
  moreover have *A* ' (*set-mset E*) ⊆ *P.events* **unfolding** *A-def P.sets-eq* **by** *blast*
  — (4) obtain a colouring avoiding bad events
  ultimately obtain *f* **where** *f* ∈ $\mathcal{C}^2$ **and** *f* ∉ ⋃(*A* '(*set-mset E*))
    using *P.Union-bound-obtain-fun*[*of set-mset E A*] *finite-set-mset P.space-eq* **by** *auto*
  thus *?thesis* **using** *event-is-proper-colouring A-def is-n-colourable-def* **by** *auto*
qed

end

corollary *erdos-propertyB-min*:
  fixes *z* :: *'a itself*
  assumes *n* > *0*
  shows (*min-edges-colouring n z*) ≥ *2*^(*n* − *1*)
proof (*rule ccontr*)

assume ¬ *2* ^ (*n* − *1*) ≤ *min-edges-colouring n z*
then have *min-edges-colouring n z* < *2*^(*n* − *1*) **by** *simp*
then obtain *h* :: *'a hyp-graph* **where** *hin*: *h* ∈ *not-col-n-uni-hyps n* **and**
enat (*size* (*hyp-edges h*)) < *2*^(*n*−*1*)
using *obtains-min-edge-colouring* **by** *blast*
then have *lt*: *size* (*hyp-edges h*) < *2*^(*n* −*1*)
by (*metis of-nat-eq-enat of-nat-less-imp-less of-nat-numeral of-nat-power*)
then interpret *kuf*: *fin-kuniform-hypergraph-nt* (*hyp-verts h*) *hyp-edges h n*
using *not-col-n-uni-hyps-def hin* **by** *auto*
have *kuf.has-property-B* **using** *kuf.erdos-propertyB lt assms* **by** *simp*
then show *False* **using** *hin not-col-n-uni-hyps-def* **by** *auto*
qed

end

3 Lovasz Local Framework Application

theory *LLL-Applications* **imports** *Lovasz-Local.Lovasz-Local-Lemma*
Lovasz-Local.Indep-Events Twelvefold-Way.Twelvefold-Way-Core
Design-Theory.Multisets-Extras Basic-Bounds-Application
begin

3.1 More set extras

lemma *multiset-remove1-filter*: *a* ∈# *A* ⟹ *P a* ⟹
{#*b* ∈# *A* . *P b*#} = {#*b* ∈# *remove1-mset a A* . *P b*#} + {#*a*#}
by *auto*

lemma *card-partition-image*:
assumes *finite C*
assumes *finite* (⋃ *c* ∈ *C* . *f c*)
assumes (⋀ *c*. *c*∈*C* ⟹ *card* (*f c*) = *k*)
assumes (⋀ *c1 c2*. *c1* ∈ *C* ⟹ *c2* ∈ *C* ⟹ *c1* ≠ *c2* ⟹ *f c1* ∩ *f c2* = {})
shows *k* ∗ *card* (*f* ‘ *C*) = *card* (⋃ *c* ∈ *C* . *f c*)
proof −
have *card* (⋃ *c* ∈ *C* . *f c*) = *card* (⋃(*f* ‘ *C*)) **by** *simp*
moreover have *finite* (*f* ‘ *C*) **using** *assms*(*1*) **by** *auto*
moreover have *finite* (⋃(*f* ‘ *C*)) **using** *assms*(*2*) **by** *auto*
moreover have ⋀*c*. *c* ∈ *f* ‘ *C* ⟹ *card c* = *k* **using** *assms*(*3*) **by** *auto*
moreover have (⋀*c1 c2*. *c1* ∈ *f* ‘ *C* ⟹ *c2* ∈ *f* ‘ *C* ⟹ *c1* ≠ *c2* ⟹ *c1* ∩ *c2*
= {}) **using** *assms*(*4*) **by** *auto*
ultimately show *?thesis* **using** *card-partition*[*of f* ‘ *C k*] **by** *auto*
qed

lemma *mset-set-implies*:
assumes *image-mset f* (*mset-set A*) = *B*
assumes ⋀ *a* . *a* ∈ *A* ⟹ *P* (*f a*)
shows ⋀ *b*. *b* ∈# *B* ⟹ *P b*
proof −

fix *b* **assume** $b \in\# B$
then obtain *a* **where** $a \in A$ **and** *eqb*: $f\ a = b$
using *assms*(*1*) **by** (*meson bij-mset-obtain-set-elem*)
then show $P\ b$ **using** *assms*(*2*) **by** *auto*
qed

lemma *card-partition-image-inj*:
assumes *finite* C
assumes *inj-on* $f\ C$
assumes *finite* $(\bigcup c \in C .\ f\ c)$
assumes $(\bigwedge c.\ c \in C \Longrightarrow \mathit{card}\ (f\ c) = k)$
assumes $(\bigwedge c1\ c2.\ c1 \in C \Longrightarrow c2 \in C \Longrightarrow c1 \neq c2 \Longrightarrow f\ c1 \cap f\ c2 = \{\})$
shows $k * \mathit{card}\ (C) = \mathit{card}\ (\bigcup c \in C .\ f\ c)$
proof –
have $\mathit{card}\ C = \mathit{card}\ (f\ `\ C)$ **using** *assms*(*2*) *card-image*
by *fastforce*
then show *?thesis* **using** *card-partition-image assms*
by *metis*
qed

lemma *size-big-union-sum2*:
fixes $M :: 'a \Rightarrow 'b$ *multiset*
shows $\mathit{size}\ (\sum x \in\# X .\ M\ x) = (\sum x \in\# X .\ \mathit{size}\ (M\ x))$
by (*induct X*) *auto*

lemma *size-big-union-sum2-const*:
fixes $M :: 'a \Rightarrow 'b$ *multiset*
assumes $\bigwedge x .\ x \in\# X \Longrightarrow \mathit{size}\ (M\ x) = k$
shows $\mathit{size}\ (\sum x \in\# X .\ M\ x) = \mathit{size}\ X * k$
proof –
have $\mathit{size}\ (\sum x \in\# X .\ M\ x) = (\sum x \in\# X .\ \mathit{size}\ (M\ x))$
using *size-big-union-sum2* **by** *auto*
also have ... $= (\sum x \in\# X .\ k)$ **using** *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *count-sum-mset2*: $\mathit{count}\ (\sum x \in\# X .\ M\ x)\ a = (\sum x \in\# X .\ \mathit{count}\ (M\ x)\ a)$
using *count-sum-mset* **by** (*smt* (*verit*) *image-image-mset sum-over-fun-eq*)

lemma *mset-subset-eq-elemI*:
$(\bigwedge a .\ a \in\# A \Longrightarrow \mathit{count}\ A\ a \leq \mathit{count}\ B\ a) \Longrightarrow A \subseteq\# B$
by (*intro mset-subset-eqI*) (*metis zero-le count-eq-zero-iff*)

lemma *mset-obtain-from-filter*:
assumes $a \in\# \{\# b \in\# B .\ P\ b \#\}$
shows $a \in\# B$ **and** $P\ a$
using *assms* **apply** (*metis multiset-partition union-iff*)
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *Multiset.set-mset-filter mem-Collect-eq*)

3.2 Mutual Independence Principle for Hypergraphs

context *fin-hypergraph-nt*
begin

definition (**in** *incidence-system*) *block-intersect-count* :: *'a set* ⇒ *nat* **where**
$\textit{block-intersect-count}\ b \equiv \textit{size}\ \{\#\ b2 \in\#\ (\mathcal{B} - \{\#b\#\})\ .\ b2 \cap b \neq \{\}\ \#\}$

lemma (**in** *hypergraph*) *edge-intersect-count-inc*:
  assumes $e \in\# E$
  shows $\textit{size}\ \{\#\ f \in\# E\ .\ f \cap e \neq \{\}\#\} = \textit{block-intersect-count}\ e + 1$
  unfolding *block-intersect-count-def*
proof −
  have $e \cap e \neq \{\}$ **using** *blocks-nempty assms*(*1*) **by** *simp*
  then have $\{\#f \in\# E.\ f \cap e \neq \{\}\#\} = \{\#f \in\# \textit{remove1-mset}\ e\ E.\ f \cap e \neq \{\}\#\} + \{\#e\#\}$
    using *multiset-remove1-filter*[*of e E* $\lambda f.\ f \cap e \neq \{\}$] *assms*(*1*) **by** *blast*
  then show $\textit{size}\ \{\#f \in\# E.\ f \cap e \neq \{\}\#\} = \textit{size}\ \{\#b2 \in\# \textit{remove1-mset}\ e\ E.\ b2 \cap e \neq \{\}\#\} + 1$
    by (*metis size-single size-union*)
qed

lemma *disjoint-set-is-mutually-independent*:
  assumes *iin*: $i \in \{0..<(\textit{size}\ E)\}$
  assumes *idffn*: $\textit{idf} \in \{0..<\textit{size}\ E\} \rightarrow_E \textit{set-mset}\ E$
  assumes *Aefn*: $\bigwedge i.\ i \in \{0..<\textit{size}\ E\} \Longrightarrow \textit{Ae}\ i = \{f \in \mathcal{C}^2\ .\ \textit{mono-edge}\ f\ (\textit{idf}\ i)\}$
  shows *prob-space.mutual-indep-events* (*uniform-count-measure* $(\mathcal{C}^2)$) (*Ae i*) *Ae*
    $(\{j \in \{0..<(\textit{size}\ E)\}\ .\ (\textit{idf}\ j \cap \textit{idf}\ i) = \{\}\})$
proof −
  interpret *P*: *vertex-colour-space* $\mathcal{V}$ *E 2*
    using *order-ge-two* **by** (*unfold-locales*) (*auto*)
  have *P.mutual-indep-events* (*Ae i*) *Ae* $(\{j \in \{0..<(\textit{size}\ E)\}\ .\ (\textit{idf}\ j \cap \textit{idf}\ i) = \{\}\})$
  proof (*intro P.mutual-indep-eventsI*)
    show $\textit{Ae}\ i \in \textit{P.events}$ **using** *P.sets-eq iin Aefn* **by** *simp*
  next
    show $\textit{Ae}\ `\ \{j \in \{0..<\mathrm{b}\}.\ \textit{idf}\ j \cap \textit{idf}\ i = \{\}\} \subseteq \textit{P.events}$ **using** *P.sets-eq Aefn* **by** *auto*
  next
    show $\bigwedge J.\ J \subseteq \{j \in \{0..<\mathrm{b}\}.\ \textit{idf}\ j \cap \textit{idf}\ i = \{\}\} \Longrightarrow J \neq \{\} \Longrightarrow$
      $\textit{P.prob}\ (\textit{Ae}\ i \cap \bigcap\ (\textit{Ae}\ `\ J)) = \textit{P.prob}\ (\textit{Ae}\ i) * \textit{P.prob}\ (\bigcap\ (\textit{Ae}\ `\ J))$
    proof −
      let $?e = \textit{idf}\ i$
      fix *J* **assume** *jss*: $J \subseteq \{j \in \{0..<\mathrm{b}\}.\ \textit{idf}\ j \cap ?e = \{\}\}$ **and** *ne*: $J \neq \{\}$
    then have *finite J* **using** *finite-subset finite-nat-set-iff-bounded-le mem-Collect-eq*
        by (*metis* (*full-types*) *finite-Collect-conjI finite-atLeastLessThan*)
      have *jin*: $\bigwedge j\ .\ j \in J \Longrightarrow j \in \{0..<\mathrm{b}\}$ **using** *jss* **by** *auto*
      have *iedge*: $\bigwedge i.\ i \in \{0..<\textit{size}\ E\} \Longrightarrow \textit{idf}\ i \in\# E$ **using** *idffn* **by** *auto*
      define P' **where** $P' \equiv (\mathcal{V} - ?e) \rightarrow_E \{0..<2::\textit{colour}\}$
        then have *finP*: *finite* P' **using** *finite-PiE finite-sets* **by** (*metis P.finP*

finite-Diff)
  define T **where** $T \equiv \lambda\ p.\ \{f \in \mathcal{C}^2\ .\ \forall\ v \in (\mathcal{V} - ?e)\ .\ f\ v = p\ v\}$
  have *Tss*: $\bigwedge p.\ T\ p \subseteq \mathcal{C}^2$ **unfolding** *T-def* **by** *auto*
  have *Pdjnt*: $\bigwedge p1\ p2.\ p1 \in P' \Longrightarrow p2 \in P' \Longrightarrow p1 \neq p2 \Longrightarrow T\ p1 \cap T\ p2 = \{\}$
  proof −
    fix *p1 p2* **assume** *p1in*: $p1 \in P'$ **and** *p2in*: $p2 \in P'$ **and** *pne*: $p1 \neq p2$
    have $\bigwedge x.\ x \in T\ p1 \Longrightarrow x \notin T\ p2$
    proof (*rule ccontr*)
      fix x **assume** *xin*: $x \in T\ p1$ **and** $\neg\ x \notin T\ p2$
      then have *xin2*: $x \in T\ p2$ **by** *simp*
      then have $x \in \mathcal{C}^2$ **and** $\forall\ v \in (\mathcal{V} - ?e)\ .\ x\ v = p1\ v$ **and** $\forall\ v \in (\mathcal{V} - ?e)\ .\ x\ v = p2\ v$
        using *T-def xin* **by** *auto*
      then have $\bigwedge v.\ v \in (\mathcal{V} - ?e) \Longrightarrow p1\ v = p2\ v$ **by** *auto*
      then have $p1 = p2$ **using** *p1in p2in* **unfolding** *P′-def*
        using *PiE-ext* **by** *metis*
      then show *False* **using** *pne* **by** *simp*
    qed
    then show $T\ p1 \cap T\ p2 = \{\}$ **by** *auto*
  qed
  have *cp*: $\bigwedge p.\ p \in P' \Longrightarrow card\ (T\ p) = 2\ powi\ (card\ ?e)$
  proof −
    fix p **assume** *pin*: $p \in P'$
    have $card\ (\mathcal{V} - ?e) = card\ \mathcal{V} - card\ ?e$ **using** *iedge wellformed iin*
      using *block-complement-def block-complement-size* **by** *auto*
    moreover have $card\ \mathcal{V} \geq card\ ?e$ **using** *iedge wellformed iin*
      by (*simp add*: *block-size-lt-order*)
    ultimately have $(card\ \mathcal{V} - card\ (\mathcal{V} - ?e)) = card\ ?e$ **by** *simp*
    then have $card\ \{0..<2::colour\}\ \hat{}\ (card\ \mathcal{V} - card\ (\mathcal{V} - ?e)) = 2\ powi\ (card\ ?e)$ **by** *auto*
    moreover have $(\bigwedge a.\ a \in (\mathcal{V} - ?e) \Longrightarrow p\ a \in \{0..<2\})$
      using *pin P′-def* **by** *auto*
    ultimately show $card\ (T\ p) = 2\ powi\ (card\ ?e)$
    unfolding *T-def* **using** *card-PiE-filter-range-set*[*of* $\mathcal{V} - ?e\ p\ \{0..<2::colour\}\ \mathcal{V}$]
        finite-sets all-n-vertex-colourings-alt **by** *auto*
  qed
  define *Ps* **where** $Ps \equiv \{p \in P'\ .\ T\ p \subseteq \bigcap(Ae\ `\ J)\}$
  have *psss*: $Ps \subseteq P'$ **unfolding** *Ps-def P′-def* **by** *auto*
  have *p1*: $\bigwedge i.\ i \in \{0..<\mathrm{b}\} \Longrightarrow P.prob(Ae\ i) = 1/(2\ powi\ (int\ (card\ (idf\ i)) - 1))$
    using *Aefn P.prob-monochromatic-edge-inv iedge* **by** *simp*
  have *bunrep*: $\bigcap(Ae\ `\ J) = (\bigcup p \in Ps\ .\ T\ p)$
  proof (*intro subset-antisym subsetI*)
    fix x **assume** $x \in \bigcap(Ae\ `\ J)$
    then have *xin*: $x \in \mathcal{C}^2$ **and** *xmono*: $\bigwedge j.\ j \in J \Longrightarrow mono\text{-}edge\ x\ (idf\ j)$
      using *jin Aefn ne* **by** *auto*
    define p **where** $p = (\lambda\ v\ .\ if\ (v \in \mathcal{V} - ?e)\ then\ x\ v\ else\ undefined)$

then have *pin*: $p \in P'$ **unfolding** *P'-def* **using** *xin all-n-vertex-colourings-alt* **by** *auto*
then have *xtin*: $x \in T\ p$ **unfolding** *T-def p-def*
by (*simp add*: *xin*)
have $T\ p \subseteq \bigcap (Ae\ `\ J)$
proof (*intro subsetI*)
fix *y* **assume** *yin*: $y \in T\ p$
have $\bigwedge j\ .\ j \in J \Longrightarrow$ *mono-edge y* (*idf j*)
proof −
fix *j* **assume** *jin*: $j \in J$
then have $idf\ j \cap idf\ i = \{\}$ **using** *jss* **by** *auto*
then have $\bigwedge v\ .\ v \in idf\ j \Longrightarrow v \notin ?e$ **by** *auto*
then have $\bigwedge v\ .\ v \in idf\ j \Longrightarrow v \in \mathcal{V} - ?e$ **using** *jss wellformed*
by (*metis* (*no-types*, *lifting*) *DiffI* ‹$j \in J$› *basic-trans-rules*(*31*) *iedge mem-Collect-eq*)
then have $\bigwedge v\ .\ v \in idf\ j \Longrightarrow y\ v = x\ v$ **using** *yin T-def p-def* **by** *auto*
then show *mono-edge y* (*idf j*) **using** *xmono jin*
by (*simp add*: *mono-edge-def*)
qed
moreover have $y \in \mathcal{C}^2$
using *Tss yin* **by** *auto*
ultimately show $y \in \bigcap (Ae\ `\ J)$ **using** *Aefn jin* **by** *auto*
qed
then have $p \in Ps$ **unfolding** *Ps-def* **using** *pin* **by** *auto*
then show $x \in (\bigcup\ p \in Ps\ .\ T\ p)$
using *xtin* **by** *auto*
next
fix *x* **assume** $x \in (\bigcup\ p \in Ps\ .\ T\ p)$
then show $x \in \bigcap (Ae\ `\ J)$ **unfolding** *Ps-def* **by** *auto*
qed
moreover have *dfo*: *disjoint-family-on* (λ *p*. *T p*) *Ps*
using *psss Pdjnt disjoint-family-on-def* **by** *blast*
moreover have $(\lambda\ p\ .\ T\ p)\ `\ Ps \subseteq P.events$
unfolding *T-def Ps-def* **using** *P.sets-eq* **by** *auto*
moreover have *finPs*: *finite Ps* **using** *finP psss finite-subset* **by** *auto*
ultimately have $P.prob\ (\bigcap (Ae\ `\ J)) = (\sum\ p \in Ps.\ P.prob\ (T\ p))$
using *P.finite-measure-finite-Union*[*of Ps* λ *p* . *T p*] **by** *simp*
moreover have $\bigwedge p.\ p \in Ps \Longrightarrow P.prob\ (T\ p) = card\ (T\ p)/card\ (\mathcal{C}^2)$
using *measure-uniform-count-measure*[*of* $\mathcal{C}^2$] *Tss*
by (*simp add*: *P.MU-def P.fin-*Ω)
ultimately have $P.prob\ (\bigcap (Ae\ `\ J)) = (\sum\ p \in Ps.\ real\ (card\ (T\ p)))/(card\ (\mathcal{C}^2))$
using *sum-divide-distrib*[*of - Ps card* ($\mathcal{C}^2$)] **by** (*simp*)
also have ... $= (\sum\ p \in Ps.\ 2\ powi\ (card\ ?e))/(card\ (\mathcal{C}^2))$
using *cp psss* **by** (*simp add*: *Ps-def*)
finally have $P.prob\ (\bigcap (Ae\ `\ J)) = (card\ Ps * (2\ powi\ (card\ ?e)))/(card\ (\mathcal{C}^2))$
by *simp*
then have $P.prob\ (Ae\ i) * P.prob\ (\bigcap (Ae\ `\ J)) =$
$1/(2\ powi\ (int\ (card\ (?e)) - 1)) * ((card\ Ps * (2\ powi\ (card\ ?e)))/(card$

$(\mathcal{C}^2)))$

using *iin p1* **by** *simp*

also have ... = $(((2\ powi\ (card\ ?e)))/(2\ powi\ (int\ (card\ (?e)) - 1))) * (card\ Ps/card\ (\mathcal{C}^2))$

by (*simp add*: *field-simps*)

also have ... = $2 * (card\ Ps/card\ (\mathcal{C}^2))$

using *power-int-diff*[*of 2*::*real int* (*card ?e*) (*int* (*card* (*?e*)) − *1*)] **by** *simp*

finally have *prob*: $P.prob\ (Ae\ i) * P.prob\ (\bigcap(Ae\ `\ J)) = (card\ Ps * 2)/(card\ (\mathcal{C}^2))$ **by** *simp*

have $(Ae\ i) \cap (\bigcap(Ae\ `\ J)) = (\bigcup p \in Ps\ .\ ((Ae\ i) \cap T\ p))$

using *bunrep* **by** *auto*

moreover have *disjoint-family-on* $(\lambda\ p.\ (Ae\ i) \cap T\ p)\ Ps$

using *dfo disjoint-family-on-bisimulation*[*of T Ps* $(\lambda\ p.\ (Ae\ i) \cap T\ p)$] **by** *auto*

moreover have $(\lambda\ p\ .\ (Ae\ i) \cap T\ p)\ `\ Ps \subseteq P.events$

unfolding *T-def* **using** *P.sets-eq* **by** *auto*

ultimately have $P.prob\ ((Ae\ i) \cap (\bigcap(Ae\ `\ J))) = (\sum\ p \in Ps.\ P.prob\ ((Ae\ i) \cap T\ p))$

using *P.finite-measure-finite-Union*[*of Ps* $\lambda p.\ (Ae\ i) \cap T\ p$] *finPs* **by** *simp*

moreover have *tss2*: $\bigwedge\ p.\ (Ae\ i) \cap T\ p \subseteq \mathcal{C}^2$ **using** *Tss* **by** *auto*

moreover have $\bigwedge\ p.\ p \in Ps \Longrightarrow P.prob\ ((Ae\ i) \cap T\ p) = card\ ((Ae\ i) \cap T\ p)/card\ (\mathcal{C}^2)$

using *measure-uniform-count-measure*[*of* $\mathcal{C}^2$] *tss2 P.MU-def P.fin-*Ω **by** *simp*

ultimately have $P.prob\ ((Ae\ i) \cap (\bigcap(Ae\ `\ J))) = (\sum\ p \in Ps.\ card\ ((Ae\ i) \cap T\ p))/(card\ (\mathcal{C}^2))$

using *sum-divide-distrib*[*of - Ps card* $(\mathcal{C}^2)$] **by** (*simp*)

moreover have $\bigwedge\ p.\ p \in Ps \Longrightarrow card\ ((Ae\ i) \cap (T\ p)) = 2$

proof −

fix *p* **assume** $p \in Ps$

define *h* **where** $h \equiv \lambda\ c.\ (\lambda\ v.\ if\ (v \in \mathcal{V})\ then\ (if\ (v \in\ ?e)\ then\ c\ else\ p\ v)\ else\ undefined)$

have *hc*: $\bigwedge\ c\ v.\ v \in\ ?e \Longrightarrow h\ c\ v = c$ **unfolding** *h-def* **using** *wellformed iin iedge* **by** *auto*

have *hne*: $\bigwedge\ c1\ c2.\ c1 \neq c2 \Longrightarrow h\ c1 \neq h\ c2$

proof (*rule ccontr*)

fix *c1 c2* **assume** *ne*: $c1 \neq c2\ \neg\ h\ c1 \neq h\ c2$

then have *eq*: $h\ c1 = h\ c2$ **by** *simp*

have $\bigwedge\ v\ .\ v \in\ ?e \Longrightarrow h\ c1\ v = c1$ **using** *hc* **by** *simp*

then have $\bigwedge\ v\ .\ v \in\ ?e \Longrightarrow c1 = c2$ **using** *hc eq* **by** *auto*

then show *False* **using** *ne eq* **using** *V-nempty blocks-nempty iedge iin* **by** *blast*

qed

then have *hdjnt*: $\bigwedge\ n.\ (\forall\ c1 \in \{0..<n\}.\ \forall\ c2 \in \{0..<n\}.\ c1 \neq c2 \longrightarrow \{h\ c1\} \cap \{h\ c2\} = \{\})$

by *auto*

have *heq*: $\bigwedge\ c.\ c \in \{0..<2\} \Longrightarrow \{f \in \mathcal{C}^2\ .\ mono\text{-}edge\text{-}col\ f\ ?e\ c \wedge (\forall\ v \in (\mathcal{V} -\ ?e)\ .\ f\ v = p\ v)\} = \{h\ c\}$

proof −

fix *c* **assume** $c \in \{0..<2::nat\}$

have $\bigwedge$ *x f*. *f* ∈ {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c* ∧ (∀ *v* ∈ ($\mathcal{V}$ − *?e*) . *f v* = *p v*)} ⟹ *f x* = *h c x*
unfolding *h-def* **using** *mono-edge-colD all-n-vertex-colourings-alt* **by** *auto*
then have $\bigwedge$ *f* . *f* ∈ {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c* ∧ (∀ *v* ∈ ($\mathcal{V}$ − *?e*) . *f v* = *p v*)} ⟹ *f* = *h c*
by *auto*
moreover have *h c* ∈ {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c* ∧ (∀ *v* ∈ ($\mathcal{V}$ − *?e*) . *f v* = *p v*)}
proof −
have *h c* ∈ $\mathcal{C}^2$ **unfolding** *h-def* **using** *all-n-vertex-colourings-alt*
by (*smt* (*verit*, *ccfv-SIG*) *DiffI P′-def PiE-I PiE-mem* ‹*c* ∈ {*0..<2*}› ‹*p* ∈ *Ps*› *psss subset-eq*)
moreover have *mono-edge-col* (*h c*) *?e c* **using** *mono-edge-colI hc* **by** *auto*
ultimately show *?thesis* **unfolding** *h-def* **by** *auto*
qed
ultimately show {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c* ∧ (∀ *v* ∈ ($\mathcal{V}$ − *?e*) . *f v* = *p v*)} = {*h c*}
by *blast*
qed
have *Ae i* = (⋃ *c* ∈ {*0..<2*}. {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c*})
using *Aefn iedge iin mono-edge-set-union*[*of ?e 2*] **by** *auto*
then have (*Ae i*) ∩ (*T p*) = (⋃ *c* ∈ {*0..<2*}. {*f* ∈ $\mathcal{C}^2$. *mono-edge-col f ?e c* ∧ (∀ *v* ∈ ($\mathcal{V}$ − *?e*) . *f v* = *p v*)})
unfolding *T-def* **by** *auto*
then have *card* ((*Ae i*) ∩ (*T p*)) = *card* ((⋃ *c* ∈ {*0..<2*}. {*h c*})) **using** *heq* **by** *simp*
moreover have (*1*:: *nat*) ∗ *card* {*0..<2*::*nat*} = *card* ((⋃ *c* ∈ {*0..<2*}. {*h c*}))
proof −
have *inj-on* (λ*c*. {*h c*}) {*0..<2*} **using** *hdjnt inj-onI*
by (*metis* (*mono-tags*, *lifting*) *hne insertI1 singletonD*)
then show *?thesis*
using *card-partition-image-inj*[*of* {*0..<2*} λ *c* . {*h c*} *1*:: *nat*] *hdjnt* **by** *auto*
qed
ultimately show *card* ((*Ae i*) ∩ (*T p*)) = *2* **by** *auto*
qed
ultimately show *P.prob* (*Ae i* ∩ ⋂ (*Ae* ‘ *J*)) = *P.prob* (*Ae i*) ∗ *P.prob* (⋂ (*Ae* ‘ *J*))
using *prob* **by** *simp*
qed
qed
then show *?thesis* **using** *P.MU-def* **by** *auto*
qed

lemma *intersect-empty-set-size*:
assumes $\bigwedge$ *e* . *e* ∈#*E* ⟹ *size* {# *f* ∈# (*E* − {#*e*#}) . *f* ∩ *e* ≠ {}#} ≤ *d*

assumes *e2* ∈# *E*
shows *size* {#*e* ∈# *E* . *e* ∩ *e2* = {} #} ≥ *size E* − *d* − *1* (**is** *size ?S'* ≥ *size E* − *d* − *1*)
proof −
have *a1alt*: ⋀ *e* . *e* ∈#*E* ⟹ *size* {# *f* ∈# *E* . *f* ∩ *e* ≠ {}#} ≤ *d* + *1*
using *edge-intersect-count-inc assms*(*1*) *block-intersect-count-def* **by** *force*
have *E* = *?S'* + {#*e* ∈# *E*. *e* ∩ *e2* ≠ {}#} **by** *auto*
then have *size E* = *size ?S'* + *size* {#*e* ∈# *E*. *e* ∩ *e2* ≠ {}#}
by (*metis size-union*)
then have *size ?S'* = *size E* − *size* {#*e* ∈# *E*. *e* ∩ *e2* ≠ {}#} **by** *simp*
moreover have *size* {#*e* ∈# *E* . *e* ∩ *e2* ≠ {} #} ≤ *d* + *1* **using** *a1alt assms*(*2*) **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

3.3 Application Property B

Probabilistic framework clearly notated

proposition *erdos-propertyB-LLL*:
assumes ⋀ *e*. *e* ∈#*E* ⟹ *card e* ≥ *k*
assumes ⋀ *e* . *e* ∈#*E* ⟹ *size* {# *f* ∈# (*E* − {#*e*#}) . *f* ∩ *e* ≠ {}#} ≤ *d*
assumes *exp*(*1*)∗(*d*+*1*) ≤ (*2 powi* (*k* − *1*))
assumes *k* > *0*
shows *has-property-B*
proof −
— 1 set up probability space
interpret *P*: *vertex-colour-space* $\mathcal{V}$ *E 2*
by *unfold-locales* (*auto simp add*: *order-ge-two*)
let *?N* = {*0..*<*size E*}
obtain *id* **where** *ideq*: *image-mset id* (*mset-set ?N*) = *E* **and** *idin*: *id* ∈ *?N* $\rightarrow_E$ *set-mset E*
using *obtain-function-on-ext-funcset*[*of ?N E*] **by** *auto*
then have *iedge*: ⋀*i*. *i* ∈ *?N* ⟹ *id i* ∈# *E* **by** *auto*
— 2 define event
define *Ae* **where** *Ae* ≡ λ *i*. {*f* ∈ $\mathcal{C}^2$. *mono-edge f* (*id i*)}
— (3) Prove each event A is mutually independent of all other mono events for other edges that don't intersect.
have *0* < *P.prob* (⋂ *Ai*∈*?N*. *space P.MU* − *Ae Ai*)
proof (*intro P.lovasz-local-symmetric*[*of ?N Ae d* (*1*/(*2 powi* (*k*−*1*)))])
have *mis*: ⋀*i* . *i* ∈ *?N* ⟹ *P.mutual-indep-events* (*Ae i*) *Ae* ({*j* ∈*?N* . (*id j* ∩ *id i*) = {}})
using *disjoint-set-is-mutually-independent*[*of - id Ae*] *P.MU-def assms idin* **by** (*simp add*: *Ae-def*)
then show ⋀ *i* . *i* ∈ *?N* ⟹ ∃ *S*. *S* ⊆ *?N* − {*i*} ∧ *card S* ≥ *card ?N* − *d* − *1* ∧
P.mutual-indep-events (*Ae i*) *Ae S*
proof −
fix *i* **assume** *iin*: *i* ∈ *?N*
define *S'* **where** *S'* ≡ {*j* ∈ *?N*. (*id j*) ∩ (*id i*) = {}}

then have $S' \subseteq$ *?N* $-$ $\{i\}$ **using** *iedge assms*(*1*) **using** *blocks-nempty iin* **by** *auto*
moreover have *P.mutual-indep-events* (*Ae i*) *Ae* S' **using** *mis iin* S'*-def* **by** *simp*
moreover have *card* $S' \geq$ *card ?N* $-$ *d* $-$ *1*
unfolding S'*-def* **using** *function-map-multi-filter-size*[*of id ?N E* λ *e . e* $\cap$ (*id i*) = $\{\}$]
ideq intersect-empty-set-size[*of d id i*] *iin iedge assms*(*2*) **by** *auto*
ultimately show $\exists$ *S. S* $\subseteq$ *?N* $-$ $\{i\}$ $\wedge$ *card S* $\geq$ *card ?N* $-$ *d* $-$ *1* $\wedge$ *P.mutual-indep-events* (*Ae i*) *Ae S*
by *blast*
qed
show $\bigwedge$ *i. i* $\in$ *?N* $\Longrightarrow$ *P.prob*(*Ae i*) $\leq$ *1*/(*2 powi* (*k*$-$*1*))
unfolding *Ae-def* **using** *P.prob-monochromatic-edge-bound*[*of - k*] *iedge assms*(*4*) *assms*(*1*) **by** *auto*
show *exp*(*1*) $*$ (*1* / *2 powi int* (*k* $-$ *1*)) $*$ (*d* + *1*) $\leq$ *1*
using *assms*(*3*) **by** (*simp add*: *field-simps del*:*One-nat-def*)
(*metis Num.of-nat-simps*(*2*) *assms*(*4*) *diff-is-0-eq diff-less less-one of-nat-diff power-int-of-nat*)
qed (*auto simp add*: *Ae-def E-nempty P.sets-eq P.space-eq*)
— 4 obtain
then obtain *f* **where** *fin*: *f* $\in$ $\mathcal{C}^2$ **and** $\bigwedge$ *i. i* $\in$ *?N* $\Longrightarrow$ $\neg$ *mono-edge f* (*id i*) **using** *Ae-def*
P.obtain-intersection-prop[*of Ae ?N* λ *f i. mono-edge f* (*id i*)] *P.space-eq P.sets-eq* **by** *auto*
then have $\bigwedge$ *e. e* $\in\#$ *E* $\Longrightarrow$ $\neg$ *mono-edge f e*
using *ideq mset-set-implies*[*of id ?N E* λ *e.* $\neg$ *mono-edge f e*] **by** *blast*
then show *?thesis* **unfolding** *is-n-colourable-def*
using *is-proper-colouring-alt2 fin all-n-vertex-colourings-def*[*of 2*] **by** *auto*
qed

end

3.4 Application Corollary

A corollary on hypergraphs where k ≥9

lemma *exp-ineq-k9*:
fixes *k*:: *nat*
assumes *k* $\geq$ *9*
shows *exp*(*1*::*real*) $*$ (*k* $*$(*k* $-$*1*) + *1*) < *2*^(*k*$-$*1*)
using *assms*
proof (*induct k rule*: *nat-induct-at-least*)
case *base*
show *?case* **using** *exp-le* **by** *auto*
next
case (*Suc n*)
have *Suc n* $*$ (*Suc n* $-$ *1*) + *1* = *n* $*$ (*n* $-$ *1*) + *1* + *2*$*$*n* **by** (*simp add*: *algebra-simps power2-eq-square*)
then have *exp* (*1*::*real*) $*$ (*Suc n* $*$ (*Suc n* $-$ *1*) + *1*) = *exp 1* $*$ (*n* $*$ (*n* $-$ *1*) +

```
1) + exp 1 ∗ 2∗n
    by (simp add: field-simps)
  moreover have exp (1::real) ∗ (n ∗ (n − 1) + 1) < 2^(n−1) using Suc.hyps
by simp
  moreover have exp (1:: real) ∗ 2∗n ≤ exp (1::real) ∗ (n ∗ (n − 1) + 1)
  proof −
   have 2∗n ≤ (n ∗ (n − 1) + 1) using Suc.hyps(1) Groups.mult-ac(2) diff-le-mono
mult-le-mono1
      nat-le-linear numeral-Bit1 numerals(1) ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add

      by (metis (no-types, opaque-lifting) Suc-eq-plus1 not-less-eq-eq numeral-Bit0
trans-le-add1)
    then have 2 ∗ real n ≤ real (n ∗ (n − 1) + 1) using Suc.hyps by linarith
    then show ?thesis using exp-gt-one[of 1::real] by simp
  qed
  moreover have 2 ^(Suc n − 1) = 2 ∗ 2^(n−1)
     by (metis Nat.le-imp-diff-is-add Suc(1) add-leD1 cross3-simps(8) diff-Suc-1
eval-nat-numeral(3)
        power.simps(2) semiring-norm(174))
  ultimately show ?case by (smt (verit))
qed

context fin-kuniform-regular-hypgraph-nt
begin
```

Good example of a combinatorial counting proof in a formal environment

```
lemma (in fin-dregular-hypergraph) hdeg-remove-one:
  assumes e ∈# E
  assumes v ∈# mset-set e
  shows size {# f ∈# (E − {#e#}) . v ∈ f#} = d − 1
proof −
  have v ∈ e
    using assms by (meson count-mset-set(3) not-in-iff)
  then have vvertex: v ∈ 𝒱 using wellformed[of e] assms(1) by auto
  then have {# f ∈# (E − {#e#}) . v ∈ f#} = {# f ∈# E . v ∈ f#} − {#e#}
    by (simp add: diff-union-cancelR assms(1) finite-blocks)
  then have size {# f ∈# (E − {#e#}) . v ∈ f#} = size {# f ∈# E . v ∈ f#}
− 1
    by (metis ‹v ∈# mset-set e› count-eq-zero-iff count-mset-set(3) assms(1) mul-
tiset-remove1-filter
        size-Diff-singleton union-iff union-single-eq-member)
  moreover have size {# f ∈# E . v ∈ f#} = d
    using hdegree-def const-degree vvertex by auto
  ultimately show size {# f ∈# (E − {#e#}) . v ∈ f#} =  d − 1 by simp
qed

lemma max-intersecting-edges:
  assumes e ∈# E
  shows size {# f ∈# (E − {#e#}) . f ∩ e ≠ {}#} ≤ k ∗ (k −1)
```

proof −
 have *eq*: {# *f* ∈# (*E* − {#*e*#}) . *f* ∩ *e* ≠ {}#} ⊆# (∑ *v* ∈# (*mset-set e*) . {#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #})
 proof (*intro mset-subset-eq-elemI*)
  fix *a* **assume** *a* ∈# {#*f* ∈# *remove1-mset e E*. *f* ∩ *e* ≠ {}#} (**is** *a* ∈# *?E′*)
  then have *ain*: *a* ∈# *remove1-mset e E* **and** *a* ∩ *e* ≠ {}
   using *mset-obtain-from-filter* **by** *fast*+
  then obtain *v* **where** *v* ∈ *a* **and** *v* ∈ *e* **by** *blast*
  then have *vvertex*: *v* ∈ 𝒱 **using** *wellformed*[*of e*] *assms*(*1*) **by** *auto*
  have *count ?E′ a* ≤ *count* {#*f* ∈# *E* − {#*e*#}. *v* ∈ *f* #} *a*
  by (*metis* ‹*a* ∈# *?E′*› ‹*v* ∈ *a*› *count-filter-mset le-eq-less-or-eq mset-obtain-from-filter*(*2*))
  moreover have *count* {#*f* ∈# *E* − {#*e*#}. *v* ∈ *f* #} *a* ≤
   (∑ *v* ∈# (*mset-set e*) . *count* {#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #} *a*)
  by (*metis* ‹*v* ∈ *e*› *assms*(*1*) *finite-blocks finite-set-mset-mset-set sum-image-mset-mono-mem*)

  moreover have *count* (∑ *v* ∈# (*mset-set e*) . {#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #}) *a* =
   (∑ *v* ∈# (*mset-set e*) . *count* {#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #} *a*)
   using *count-sum-mset2* **by** *fast*
  ultimately show *count ?E′ a* ≤ *count* (∑ *v*∈#*mset-set e*. *filter-mset* ((∈) *v*) (*remove1-mset e E*)) *a*
   by *linarith*
 qed
 have *size* (*mset-set e*) = *k*
  using *uniform assms*(*1*) **by** *auto*
 then have *size* (∑ *v* ∈# (*mset-set e*) . {#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #}) = *k* ∗ (*k* − *1*)
  using *size-big-union-sum2-const*[*of mset-set e λ v* .{#*f* ∈# (*E* − {#*e*#}) . *v* ∈ *f* #} *k* − *1*]
   hdeg-remove-one assms(*1*) **by** *fast*
 then show *?thesis*
  using *eq* **by** (*metis size-mset-mono*)
qed

corollary *erdos-propertyB-LLL9*:
 assumes *k* ≥ *9*
 shows *has-property-B*
proof −
 define *d* **where** *d* = *k*∗(*k*−*1*)
 have ⋀ *e*. *e* ∈# *E* ⟹ *card e* ≥ *k*
  using *uniform* **by** *simp*
 moreover have ⋀ *e* . *e* ∈# *E* ⟹ *size* {# *f* ∈# (*E* − {#*e*#}) . *f* ∩ *e* ≠ {}#} ≤ *d*
  using *max-intersecting-edges d-def* **by** *simp*
 moreover have *exp*(*1*)∗(*d*+*1*) < (*2 powi* (*k* − *1*))
  unfolding *d-def* **using** *exp-ineq-k9 assms*(*1*) **by** *simp*
 moreover have *k* > *0* **using** *assms* **by** *auto*
 ultimately show *?thesis* **using** *erdos-propertyB-LLL*[*of k d*] *assms*
  using *int-ops*(*1*) *int-ops*(*2*) *int-ops*(*6*) *less-eq-real-def nat-less-as-int* **by** *auto*

qed

end

end
theory *Hypergraph-Colourings-Root*
imports
Hypergraph-Colourings
Basic-Bounds-Application
LLL-Applications
begin
end

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, Hoboken, N.J, 4th edition, 2016.

[2] M. Molloy and B. Reed. *Graph Colouring and the Probabilistic Method*. Algorithms and Combinatorics. Springer, 2002.

[3] Y. Zhao. Probabilistic methods in combinatorics, 2020. Lecture notes MIT 18.226, Fall 2020, https://ocw.mit.edu/courses/18-226-probabilistic-method-in-combinatorics-fall-2020/resources/mit18_226f20_full_notes/.