

Hypergraph Colouring Bounds using Probabilistic Methods

Chelsea Edmonds and Lawrence C. Paulson

April 18, 2024

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	2
1.1	Function and Number extras	2
1.2	Basic Definitions	3
1.3	Monochromatic Edges	5
1.4	Proper colourings	5
1.5	n vertex colourings	9
1.6	Alternate Partition Definition.	11
2	Basic Probabilistic Method Application	12
2.1	Probability Spaces for Incidence Set Systems	12
2.2	More Hypergraph Colouring Results	19
2.3	The Basic Application	20

3	Lovasz Local Framework Application	21
3.1	More set extras	21
3.2	Mutual Independence Principle for Hypergraphs	23
3.3	Application Property B	28
3.4	Application Corollary	29

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

lemma *enat-lt-INF*:
fixes $f :: 'a \Rightarrow \text{enat}$
assumes $(\text{INF } x \in S. f x) < t$
obtains x **where** $x \in S$ **and** $f x < t$
proof –
from *assms* **have** $(\text{INF } x \in S. f x) \neq \text{top}$
by *fastforce*
then obtain y **where** $y \in S$ **and** $f y = (\text{INF } x \in S. f x)$ **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 \Rightarrow colour) \Rightarrow colour set \Rightarrow bool **where**
edge-colouring f C \equiv partition-on-mset E {# {#h \in E . f h = c#} . c \in # (mset-set C)#}

definition *proper-edge-colouring* :: ('a hyp-edge \Rightarrow colour) \Rightarrow colour set \Rightarrow bool **where**
proper-edge-colouring f C \equiv *edge-colouring* f C \wedge
 $(\forall e1 e2 c. e1 \in \# E \wedge e2 \in \# E - \{\#e1\# \} \wedge c \in C \wedge f e1 = c \wedge f e2 = c \longrightarrow e1 \cap e2 = \{\})$

A vertex colouring function with no edge monochromatic requirements

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

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

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

lemma *vertex-colouring-n0*: $\mathcal{V} \neq \{\} \implies \neg$ *vertex-colouring* f 0
by *auto*

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

```

using funcset-mem by blast

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

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

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

lemma vertex-colour-one:  $\mathcal{V} \neq \{\}$   $\implies$  vertex-colouring f 1  $\implies$  v  $\in$   $\mathcal{V} \implies$  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 = ( $\lambda$  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)  $\implies$  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..
  shows partition-on  $\mathcal{V} \{ \{v \in \mathcal{V} . f v = c\} \mid c. c \in \{0..
proof (intro partition-onI)
  fix p assume p  $\in$   $\{ \{v \in \mathcal{V} . f v = c\} \mid c. c \in \{0..
  then obtain c where peq: p =  $\{v \in \mathcal{V} . f v = c\}$  and cin: c  $\in$   $\{0.. by blast
  have f  $\in$   $\mathcal{V} \rightarrow_E \{0.. 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.. c] cin assms(2) by auto
  then show p  $\neq \{\}$  using peq by auto
next
  show  $\bigcup \{ \{v \in \mathcal{V} . f v = c\} \mid c. c \in \{0.. using vertex-colouring-union
  assms by auto
next$$$$$$$ 
```

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

1.3 Monochromatic Edges

definition *mono-edge* :: ('a \Rightarrow colour) \Rightarrow 'a hyp-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** $ceq: \bigwedge v. v \in e \implies f v = c$ **by** *blast*

then have f ' e = {c} **using** *image-singleton assms blocks-empty* **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-edge \Rightarrow colour \Rightarrow bool **where**
mono-edge-col f e c $\equiv \forall v \in e. f v = c$

lemma *mono-edge-colI*: $(\bigwedge v. v \in e \implies f v = c) \implies$ *mono-edge-col* f e c

unfolding *mono-edge-col-def* **by** *simp*

lemma *mono-edge-colD*: *mono-edge-col* f e c $\implies (\bigwedge v. v \in e \implies f v = c)$

unfolding *mono-edge-col-def* **by** *simp*

lemma *mono-edge-alt-col*: *mono-edge* f e $\equiv \exists c. \text{mono-edge-col } f e c$

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 $\wedge (\forall e \in \# E. \neg \text{is-singleton } (f ' e))$

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* \longleftrightarrow *vertex-colouring f n*
 $\wedge (\forall e \in \# E. \neg \text{mono-edge } f e)$
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* $\implies (\bigwedge e . e \in \# E \implies$
 $\neg \text{is-singleton } (f \text{ ' } e)) \implies \text{is-proper-colouring } f n$
using *is-proper-colouring-alt* **by** *simp*

lemma *is-proper-colouringI2[intro]*: *vertex-colouring f n* $\implies (\bigwedge e . e \in \# E \implies \neg$
 $\text{mono-edge } f e)$
 $\implies \text{is-proper-colouring } f n$
using *is-proper-colouring-alt2* **by** *simp*

lemma *is-proper-colouring-n0*: $\mathcal{V} \neq \{\}$ $\implies \neg \text{is-proper-colouring } f 0$
unfolding *is-proper-colouring-def* **using** *vertex-colouring-n0* **by** *auto*

lemma *is-proper-colouring-empty*:
assumes $\mathcal{V} = \{\}$
shows *is-proper-colouring f n* $\longleftrightarrow f = (\lambda x . \text{undefined})$
unfolding *is-proper-colouring-def* **using** *PiE-empty-domain assms*
using *vertex-colouring-image-edge-ss* **by** *fastforce*

lemma *is-proper-colouring-n1*:
assumes $\mathcal{V} \neq \{\}$ $E \neq \{\#\}$
shows $\neg \text{is-proper-colouring } f 1$
proof (*rule ccontr*)
assume $\neg \neg \text{is-proper-colouring } f 1$
then have *vc*: *vertex-colouring f 1* **and** *em*: $(\forall e \in \# E. \neg \text{mono-edge } f e)$
using *is-proper-colouring-alt2* **by** *auto*
then obtain *e* **where** *ein*: $e \in \# E$ **using** *assms* **by** *blast*
have $f \in \mathcal{V} \rightarrow_E \{0\}$ **using** *vc* **by** *auto*
then have $\forall v \in \mathcal{V}. f v = 0$
by *simp*
then have $\forall v \in e. f v = 0$ **using** *wellformed* $\langle e \in \# E \rangle$ **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 $\mathcal{V} \neq \{\}$ $E \neq \{\#\}$
assumes $n > 1$
assumes *is-proper-colouring f n*
shows $\text{card } (f \text{ ' } \mathcal{V}) > 1$
proof (*rule ccontr*)
assume $\neg 1 < \text{card } (f \text{ ' } \mathcal{V})$
then have *a*: $\text{card } (f \text{ ' } \mathcal{V}) = 1$

using *assms* **by** (*meson card-0-eq finite-imageI finite-sets image-is-empty less-one linorder-neqE-nat*)
then obtain *c* **where** *ceq*: $f \text{ ` } \mathcal{V} = \{c\}$
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-monochromatic-is-colouring*:
assumes $\forall e \in \# E. \neg \text{mono-edge } f e$
assumes *vertex-colouring f n*
shows *is-proper-colouring f n*
using *assms mono-edge-single is-proper-colouringI* **by** (*auto*)

lemma *ex-monochromatic-not-colouring*:
assumes $\exists e \in \# E. \text{mono-edge } f e$
assumes *vertex-colouring f n*
shows $\neg \text{is-proper-colouring } f n$
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*: $f \text{ ` } e \subseteq \{0..<n\}$ **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 $f \text{ ` } e \neq \{\}$ **using** *blocks-nempty* **by** (*simp add: assms(3)*)
then have $c \in f \text{ ` } e$ **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 $f \text{ ` } \mathcal{V} = \{0..<n\}$

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

by *auto*

lemma *is-complete-proper-colouring-n0*: $\mathcal{V} \neq \{\}$ $\implies \neg$ *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* $f'\ (\text{card } (f' \mathcal{V}))$
proof (*cases* $f' \mathcal{V} = \{0..<(n::\text{nat})\}$)
case *True*
then have $\text{card } (f' \mathcal{V}) = n$ **by** *simp*
then show *thesis* **using** *is-complete-proper-colouring-def* *assms*
using *True* **that** **by** *auto*

next
case *False*
obtain $g :: \text{nat} \Rightarrow \text{nat}$ **where** *bij*: *bij-betw* $g\ (f' \mathcal{V})\ \{0..<(\text{card } (f' \mathcal{V}))\}$
using *ex-bij-betw-finite-nat* *finite-sets* **by** *blast*
let $?f' = \lambda x . \text{if } x \in \mathcal{V} \text{ then } (g \circ f)\ x \text{ else undefined}$
have *img*: $?f' \mathcal{V} = \{0..<\text{card } (f' \mathcal{V})\}$ **using** *bij* *bij-betw-imp-surj-on* *image-comp*
by (*smt* (*verit*) *image-cong*)
have *is-proper-colouring* $?f'\ (\text{card } (f' \mathcal{V}))$
proof (*intro* *is-proper-colouringI*)
show *vertex-colouring* $?f'\ (\text{card } (f' \mathcal{V}))$
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 ‘ V) > 1* **using** *is-proper-colouring-image-card assms*
using *one-less-numeral-iff semiring-norm(76)* **by** *blast*
have *f ∈ V →_E {0..<2}* **using** *is-proper-colouring-def assms(3)* **by** *auto*
then have *f ‘ V ⊆ {0..<2}* **by** *blast*
then have *card (f ‘ 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 ‘ V ⊆ {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 ((Cⁿ) [502] 500)*

lemma *all-n-vertex-colourings-alt: Cⁿ = V →_E {0..<n}*
unfolding *all-n-vertex-colourings-def* **by** *auto*

lemma *vertex-colourings-empty: V ≠ {} ⇒ 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 (Cⁿ)*
using *all-n-vertex-colourings-alt finite-PiE finite-sets* **by** (*metis finite-atLeastLessThan*)

lemma (*in fin-hypergraph*) *count-vertex-colourings: card (Cⁿ) = 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 V ≥ n*
assumes *n ≠ 0*
shows *Cⁿ ≠ {}*
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  $\{f \in \mathcal{C}^n . \text{mono-edge } f e\} = (\bigcup c \in \{0..<n\} . \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\})$ 
  proof (intro subset-antisym subsetI)
    fix  $g$  assume  $a1: g \in \{f \in \mathcal{C}^n . \text{mono-edge } f e\}$ 
    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  $g \in (\bigcup c \in \{0..<n\} . \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\})$ 
      using  $\langle \text{vertex-colouring } g n \rangle$  all-n-vertex-colourings-def by auto
  next
    fix  $h$  assume  $h \in (\bigcup c \in \{0..<n\} . \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\})$ 
    then obtain  $c$  where  $c \in \{0..<n\}$  and  $h \in \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\}$ 
      by blast
    then show  $h \in \{f \in \mathcal{C}^n . \text{mono-edge } f e\}$ 
      using mono-edge-alt-col by blast
qed

end

```

Property B set up

```

abbreviation (in hypergraph) has-property-B :: bool where
has-property-B  $\equiv$  is-n-colourable 2

```

```

abbreviation hyp-graph-order:: 'a hyp-graph  $\Rightarrow$  nat where
hyp-graph-order  $h \equiv$  card (hyp-verts h)

```

```

definition not-col-n-uni-hyps:: nat  $\Rightarrow$  'a hyp-graph set
  where not-col-n-uni-hyps  $n \equiv$   $\{ h . \text{fin-kuniform-hypergraph-nt } (\text{hyp-verts } h) (\text{hyp-edges } h) n \wedge$ 
 $\neg (\text{hypergraph.has-property-B } (\text{hyp-verts } h) (\text{hyp-edges } h)) \}$ 

```

```

definition min-edges-colouring :: nat  $\Rightarrow$  'a itself  $\Rightarrow$  enat where
min-edges-colouring  $n - \equiv$  INF  $h \in ((\text{not-col-n-uni-hyps } n) :: 'a \text{ hyp-graph set}) .$ 
enat (size (hyp-edges h))

```

```

lemma obtains-min-edge-colouring:
  fixes  $z :: 'a \text{ itself}$ 
  assumes min-edges-colouring  $n z < x$ 
  obtains  $h :: 'a \text{ hyp-graph}$  where  $h \in \text{not-col-n-uni-hyps } n$  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-monochromatic-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-monochromatic-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 $\mathcal{V} \geq n \implies n \neq 0 \implies$* *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 ($\lambda c. \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\}) \{0..<n\}$*

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 *$F :: 'a \text{ set} \Rightarrow 'b \text{ set}$*

fixes *$p :: 'b \Rightarrow \text{real}$*

assumes *$ne: F \mathcal{V} \neq \{\}$*

assumes *$fin: \text{finite } (F \mathcal{V})$*

assumes *$pgte0: \bigwedge fv . fv \in F \mathcal{V} \implies p fv \geq 0$*

assumes *$sump: (\sum x \in (F \mathcal{V}) . p x) = 1$*

begin

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

lemma *$fin\text{-}\Omega: \text{finite } \Omega$*

unfolding *$\Omega\text{-def}$* **using** *fin* **by** *auto*

lemma *$ne\text{-}\Omega: \Omega \neq \{\}$*

unfolding Ω -def using *ne* by *simp*

definition $M = \text{point-measure } \Omega \ p$

lemma *space-eq*: $\text{space } M = \Omega$
unfolding M -def Ω -def by (*simp add: space-point-measure*)

lemma *sets-eq*: $\text{sets } M = \text{Pow } (\Omega)$
unfolding M -def by (*simp add: sets-point-measure*)

lemma *finite-event*: $A \subseteq \Omega \implies \text{finite } A$
by (*simp add: finite-subset fin- Ω*)

lemma *emeasure-eq*: $\text{emeasure } M \ A = (\text{if } (A \subseteq \Omega) \text{ then } (\sum_{a \in A} p \ a) \text{ else } 0)$
proof (*cases* $A \subseteq \Omega$)
 case *True*
 then have *finite* A using *finite-event* by *auto*
 moreover have $\text{ennreal } (\text{sum } p \ A) = (\sum_{a \in A} \text{ennreal } (p \ a))$
 using *sum-ennreal pgte0 True* by (*simp add: subset-iff Ω -def*)
 ultimately have $\text{emeasure } M \ A = (\sum_{a \in A} p \ a)$
 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*]*: $\text{integrable } M \ (f::- \implies \text{real})$
using *fin- Ω* by (*simp add: integrable-point-measure-finite M-def*)

lemma *borel-measurable-M[*measurable*]*: $f \in \text{borel-measurable } M$
unfolding M -def by *simp*

lemma *prob-space-M*: $\text{prob-space } M$
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* \subseteq *prob-space* M
using *prob-space-M* .

 A uniform variation of the space

locale *vertex-fn-space-uniform* = *fin-hypersystem-vne* +
 fixes $F :: 'a \text{ set} \implies 'b \text{ set}$
 assumes *ne*: $F \ \mathcal{V} \neq \{\}$
 assumes *fin*: $\text{finite } (F \ \mathcal{V})$
begin

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

definition $MU \equiv \text{uniform-count-measure } \Omega U$

end

sublocale $\text{vertex-fn-space-uniform} \subseteq \text{vertex-fn-space } \mathcal{V} E F (\lambda x. 1 / \text{card } \Omega U)$
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: $\bigwedge fv. fv \in F \mathcal{V} \implies 0 \leq 1 / \text{real } (\text{card } (\Omega U))$ **by** *auto*

show 4: $(\sum x \in F \mathcal{V}. 1 / \text{real } (\text{card } \Omega U)) = 1$

using *sum-constant ne fin ΩU -def* **by** *auto*

interpret *vf*: $\text{vertex-fn-space } \mathcal{V} E F (\lambda x. 1 / \text{card } (\Omega U))$

using 1 2 3 4 **by** (*unfold-locales*)

show *vf*. $\Omega = \Omega U$ **unfolding** *vf*. Ω -*def* ΩU -*def* **by** *simp*

show *vf*. $M = MU$ **unfolding** *vf*. M -*def* *vf*. Ω -*def* MU -*def* *uniform-count-measure-def*

using ΩU -*def* **by** *auto*

qed

context *vertex-fn-space-uniform*

begin

lemma *emeasure-eq*: $\text{emeasure } MU A = (\text{if } (A \subseteq \Omega U) \text{ then } ((\text{card } A) / \text{card } (\Omega U)) \text{ else } 0)$

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*: $A \in \text{events} \implies \text{measure } MU A = (\text{card } A) / \text{card } (\Omega U)$

using *sets-eq* **by** (*simp add: MU -def ΩU -def fin* *measure-uniform-count-measure*)

lemma *expectation-eq*:

shows $\text{expectation } f = (\sum x \in \Omega U. f x) / \text{card } \Omega U$

proof–

have $(\bigwedge a. a \in \Omega U \implies 0 \leq 1 / \text{real } (\text{card } \Omega U))$

using *fin- Ω ne- Ω* **by** *auto*

moreover have $\bigwedge a. a \in \Omega U \implies (1 / \text{real } (\text{card } \Omega U)) *_{\mathbb{R}} f a = 1 / (\text{card } \Omega U) * f a$

using *real-scale \mathbb{R} -def* **by** *simp*

ultimately have $\text{expectation } f = (\sum x \in \Omega U. f x * (1 / (\text{card } \Omega U)))$

using *uniform-count-measure-def*[*of ΩU*] *lebesgue-integral-point-measure-finite*[*of $\Omega U (\lambda x. 1 / \text{card } \Omega U) f$*]

MU-def fin- Ω **by** *auto*

then show *?thesis* **using** *sum-distrib-right*[*symmetric, of $f 1 / (\text{card } \Omega U) \Omega U$*]

by *auto*

qed

end

A probability space over the full vertex set

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

```
sublocale vertex-space  $\subseteq$  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  $\Omega$ -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  $\subseteq$  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  $\Omega U$ -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  $\subseteq$  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  $\Omega$ -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  $\implies$  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  $\subseteq$  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  $\Omega$ -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  $\subseteq$  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  $\Omega U$ -def by simp  
qed
```

```
context vertex-prop-space  
begin
```

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


assumes $d \subseteq \mathcal{V}$
shows $\text{prob } \{f \in \Omega . (\forall v \in d . f v = b)\} = 1/((\text{card } P) \text{ powi } (\text{card } d))$
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 $\text{prob } \{f \in \Omega U . f v = b\} = 1/(\text{card } P)$
proof –
have $\text{prob } \{f \in \Omega U . f v = b\} = \text{card } \{f \in \Omega U . f v = b\} / \text{card } \Omega U$
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 $n :: \text{nat}$
assumes *n-lt-order: $n \leq \text{horder}$*
assumes *n-not-zero: $n \neq 0$*

sublocale *vertex-colour-space \subseteq 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: $\{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\} \in \text{events}$*
using *sets-eq by simp*

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

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

using *prob-uniform-vertex* **by** *simp*

lemma *prob-edge-colour*:

assumes $e \in \# E$ $c \in \{0..<n\}$

shows $\text{prob} \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\} = 1/(n \text{ powi } (\text{card } e))$

proof –

have $\text{card } \{0..<n\} = n$ **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 $\text{prob}\{f \in \mathcal{C}^n . \text{mono-edge } f e\} = 1/(n \text{ powi } (\text{int } (\text{card } e) - 1))$

proof –

have *finite* $\{0..<n\}$ **by** *auto*

then **have** $\text{prob} \{f \in \mathcal{C}^n . \text{mono-edge } f e\} = (\sum c \in \{0..<n\} . \text{prob} \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\})$

using *finite-measure-finite-Union*[of $\{0..<n\}$ $(\lambda c . \{f \in \mathcal{C}^n . \text{mono-edge-col } f e c\})$]

disjoint-family-on-colourings colour-functions-event mono-edge-set-union assms
by *auto*

also **have** $\dots = n/(n \text{ powi } (\text{int } (\text{card } e)))$ **using** *prob-edge-colour assms* **by** *simp*

also **have** $\dots = n/(n * (n \text{ powi } ((\text{int } (\text{card } e) - 1)))$ **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 $\text{prob}\{f \in \mathcal{C}^n . \text{mono-edge } f e\} = n \text{ powi } (1 - \text{int } (\text{card } e))$

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 $\bigwedge e. e \in \# E \implies \text{card } e \geq k$

assumes $k > 0$

shows $\text{prob}\{f \in \mathcal{C}^n . \text{mono-edge } f e\} \leq 1/((\text{real } n) \text{ powi } (k-1))$

proof –

have $(\text{int } (\text{card } (e)) - 1) \geq k - 1$ **using** *assms(3) assms(1)*

using *assms(2) int-ops(6) of-nat-0-less-iff of-nat-mono* **by** *fastforce*

then **have** $((n :: \text{real}) \text{ powi } (\text{int } (\text{card } (e)) - 1)) \geq (n \text{ powi } (k - 1))$

using *power-int-increasing*[of $k - 1$ (int (card (e)) - 1) n] *n-not-zero* **by** *linarith*
moreover have $\text{prob}(\{f \in \mathcal{C}^n . \text{mono-edge } f \ e\}) = 1/(n \text{ powi } (\text{int } (\text{card } (e)) - 1))$
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*: $\{f \in \mathcal{C}^n . \neg \text{is-proper-colouring } f \ n\} = (\bigcup e \in (\text{set-mset } E). \{f \in \mathcal{C}^n . \text{mono-edge } f \ e\})$

proof –

have $\{f \in \mathcal{C}^n . \neg \text{is-proper-colouring } f \ n\} = \{f \in \mathcal{C}^n . \exists e \in \text{set-mset } E . \text{mono-edge } f \ e\}$

using *ex-monochromatic-not-colouring* *no-monochromatic-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*: $\{f \in \mathcal{C}^n . \text{is-proper-colouring } f \ n\} = (\bigcap e \in (\text{set-mset } E). \{f \in \mathcal{C}^n . \neg \text{mono-edge } f \ e\})$

proof –

have $\{f \in \mathcal{C}^n . \text{is-proper-colouring } f \ n\} = \{f \in \mathcal{C}^n . \forall e \in \text{set-mset } E . \neg \text{mono-edge } f \ e\}$

using *is-proper-colouring-alt2* *all-n-vertex-colourings-alt* **by** *auto*

moreover have $\text{set-mset } E \neq \{\}$ **using** *E-not-empty* **by** *simp*

ultimately show *?thesis* **using** *Inter-forall* **by** *auto*

qed

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

using *proper-colouring-edge-mono* **by** *auto*

lemma *event-is-proper-colouring*:

assumes $g \in \mathcal{C}^n$

assumes $g \notin (\bigcup e \in (\text{set-mset } E). \{f \in \mathcal{C}^n . \text{mono-edge } f \ e\})$

shows *is-proper-colouring* $g \ n$

proof –

have $\bigwedge e. e \in \# E \implies \neg \text{mono-edge } g \ e$ **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** $A \equiv (\lambda e. \{f \in \mathcal{C}^2 . \text{mono-edge } f e\})$

– (3) Calculation 2: Have $\Pr(\text{of } A e \text{ for any } e) \leq \sum_{e \in E} (\Pr(A e)) < 1$

have $(\sum_{e \in \text{set-mset } E}. P.\text{prob } (A e)) < 1$

proof –

have $\text{int } k - 1 = \text{int } (k - 1)$ **using** *assms* **by** *linarith*

then have $\text{card } (\text{set-mset } E) < 2^{\text{powi } (\text{int } k - 1)}$ **using** *card-size-set-mset*[*of E*] *assms* **by** *simp*

then have $(\sum_{e \in (\text{set-mset } E)}. P.\text{prob } (A e)) < 2^{\text{powi } (\text{int } k - 1)} * 2^{\text{powi } (1 - \text{int } k)}$

unfolding *A-def* **using** *P.prob-monochromatic-edge uniform assms(1)* **by** *simp*

moreover have $((2 :: \text{real})^{\text{powi } ((\text{int } k) - 1)} * (2^{\text{powi } (1 - (\text{int } k))})) = 1$

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 $A \subseteq (\text{set-mset } E)$ **unfolding** *A-def* *P.sets-eq* **by** *blast*

– (4) obtain a colouring avoiding bad events

ultimately obtain *f* **where** $f \in \mathcal{C}^2$ **and** $f \notin \bigcup (A \subseteq (\text{set-mset } E))$

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 $(\text{min-edges-colouring } n z) \geq 2^{n-1}$

proof (*rule ccontr*)

```

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

end

```

3 Lovasz Local Framework Application

```

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

```

3.1 More set extras

```

lemma multiset-remove1-filter:  $a \in \# A \implies P a \implies$ 
 $\{\#b \in \# A . P b\} = \{\#b \in \# \text{remove1-mset } a A . P b\} + \{\#a\}$ 
by auto

```

lemma *card-partition-image*:

```

assumes finite C
assumes finite  $(\bigcup c \in C . f c)$ 
assumes  $(\bigwedge c. c \in C \implies \text{card } (f c) = k)$ 
assumes  $(\bigwedge c1 c2. c1 \in C \implies c2 \in C \implies c1 \neq c2 \implies f c1 \cap f c2 = \{\})$ 
shows  $k * \text{card } (f ' C) = \text{card } (\bigcup c \in C . f c)$ 
proof -
have  $\text{card } (\bigcup c \in C . f c) = \text{card } (\bigcup (f ' C))$  by simp
moreover have finite  $(f ' C)$  using assms(1) by auto
moreover have finite  $(\bigcup (f ' C))$  using assms(2) by auto
moreover have  $\bigwedge c. c \in f ' C \implies \text{card } c = k$  using assms(3) by auto
moreover have  $(\bigwedge c1 c2. c1 \in f ' C \implies c2 \in f ' C \implies c1 \neq c2 \implies c1 \cap c2 = \{\})$  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  $f (\text{mset-set } A) = B$ 
assumes  $\bigwedge a . a \in A \implies P (f a)$ 
shows  $\bigwedge b . b \in \# B \implies P b$ 
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 ($\bigwedge c. c \in C \implies card (f c) = k$)
assumes ($\bigwedge c1 c2. c1 \in C \implies c2 \in C \implies c1 \neq c2 \implies f c1 \cap f c2 = \{\}$)
shows $k * card (C) = card (\bigcup c \in C . f c)$
proof –
have $card C = card (f ' C)$ **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 $size (\sum x \in \# X . M x) = (\sum x \in \# X . size (M x))$
by (*induct X*) *auto*

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

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

lemma *mset-subset-eq-elemI*:
 $(\bigwedge a . a \in \# A \implies count A a \leq count B a) \implies A \subseteq \# B$
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 \Rightarrow nat **where**
block-intersect-count b \equiv size {# b2 \in # (B - {#b#}) . b2 \cap b \neq {} #}

lemma (in *hypergraph*) *edge-intersect-count-inc*:

assumes $e \in \# E$

shows size {# f \in # E . f \cap e \neq {} #} = *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 # *remove1-mset* e E . f \cap e \neq {} #} + {#e#}

using *multiset-remove1-filter*[of e E λ f . f \cap e \neq {}] *assms(1)* **by** *blast*

then show size {#f \in # E . f \cap e \neq {} #} = size {#b2 \in # *remove1-mset* e E . b2 \cap e \neq {} #} + 1

by (*metis size-single size-union*)

qed

lemma *disjoint-set-is-mutually-independent*:

assumes *iin*: $i \in \{0..<(size E)\}$

assumes *idffn*: $idf \in \{0..<size E\} \rightarrow_E set-mset E$

assumes *Aefn*: $\bigwedge i . i \in \{0..<size E\} \implies Ae i = \{f \in \mathcal{C}^2 . mono-edge f (idf i)\}$

shows *prob-space.mutual-indep-events* (*uniform-count-measure* (\mathcal{C}^2)) ($Ae i$) Ae ($\{j \in \{0..<(size E)\} . (idf j \cap idf i) = \{\}\}$)

proof –

interpret *P*: *vertex-colour-space* $\mathcal{V} E \mathbb{2}$

using *order-ge-two* **by** (*unfold-locales*) (*auto*)

have *P.mutual-indep-events* ($Ae i$) Ae ($\{j \in \{0..<(size E)\} . (idf j \cap idf i) = \{\}\}$)

proof (*intro P.mutual-indep-eventsI*)

show $Ae i \in P.events$ **using** *P.sets-eq iin Aefn* **by** *simp*

next

show $Ae ' \{j \in \{0..<b\} . idf j \cap idf i = \{\}\} \subseteq P.events$ **using** *P.sets-eq Aefn*

by *auto*

next

show $\bigwedge J . J \subseteq \{j \in \{0..<b\} . idf j \cap idf i = \{\}\} \implies J \neq \{\} \implies$

$P.prob (Ae i \cap \bigcap (Ae ' J)) = P.prob (Ae i) * P.prob (\bigcap (Ae ' J))$

proof –

let $?e = idf i$

fix *J* **assume** *jss*: $J \subseteq \{j \in \{0..<b\} . idf j \cap ?e = \{\}\}$ **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*: $\bigwedge j . j \in J \implies j \in \{0..<b\}$ **using** *jss* **by** *auto*

have *iedge*: $\bigwedge i . i \in \{0..<size E\} \implies idf i \in \# E$ **using** *idffn* **by** *auto*

define *P'* **where** $P' \equiv (\mathcal{V} - ?e) \rightarrow_E \{0..<2::colour\}$

then have *finP*: *finite P'* **using** *finite-PiE finite-sets* **by** (*metis P.finP*)

```

finite-Diff)
  define T where T ≡ λ p. {f ∈ C2 . ∀ v ∈ (V - ?e) . f v = p v}
  have Tss: ∧ p. T p ⊆ C2 unfolding T-def by auto
  have Pdjnt: ∧ p1 p2. p1 ∈ P' ⇒ p2 ∈ P' ⇒ p1 ≠ p2 ⇒ T p1 ∩ T p2 =
{}
  proof -
    fix p1 p2 assume p1in: p1 ∈ P' and p2in: p2 ∈ P' and pne: p1 ≠ p2
    have ∧ x. x ∈ T p1 ⇒ x ∉ T p2
    proof (rule ccontr)
      fix x assume xin: x ∈ T p1 and ¬ x ∉ T p2
      then have xin2: x ∈ T p2 by simp
      then have x ∈ C2 and ∀ v ∈ (V - ?e) . x v = p1 v and ∀ v ∈ (V - ?e)
. x v = p2 v
      using T-def xin by auto
      then have ∧ v. v ∈ (V - ?e) ⇒ 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 ∩ T p2 = {} by auto
  qed
  have cp: ∧ p. p ∈ P' ⇒ card (T p) = 2 powi (card ?e)
  proof -
    fix p assume pin: p ∈ P'
    have card (V - ?e) = card V - card ?e using iedge wellformed iin
    using block-complement-def block-complement-size by auto
    moreover have card V ≥ card ?e using iedge wellformed iin
    by (simp add: block-size-lt-order)
    ultimately have (card V - card (V - ?e)) = card ?e by simp
    then have card {0..<2::colour} ^ (card V - card (V - ?e)) = 2 powi (card
?e) by auto
    moreover have (∧ a. a ∈ (V - ?e) ⇒ p a ∈ {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 V - ?e p {0..<2::colour}
V ]
    finite-sets all-n-vertex-colourings-alt by auto
  qed
  define Ps where Ps ≡ {p ∈ P' . T p ⊆ ∩ (Ae ' J)}
  have psss: Ps ⊆ P' unfolding Ps-def P'-def by auto
  have p1: ∧ i. i ∈ {0..<b} ⇒ P.prob(Ae i) = 1/(2 powi (int (card (idf i))
- 1))
    using Aefn P.prob-monochromatic-edge-inv iedge by simp
  have bunrep: ∩ (Ae ' J) = (∪ p ∈ Ps . T p)
  proof (intro subset-antisym subsetI)
    fix x assume x ∈ ∩ (Ae ' J)
    then have xin: x ∈ C2 and xmono: ∧ j. j ∈ J ⇒ mono-edge x (idf j)
    using jin Aefn ne by auto
    define p where p = (λ v . if (v ∈ 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 $xin: x \in T p$ **unfolding** T -def p -def
by (*simp add: xin*)
have $T p \subseteq \bigcap (Ae \text{ ' } J)$
proof (*intro subsetI*)
fix y **assume** $yin: y \in T p$
have $\bigwedge j . j \in J \implies \text{mono-edge } y (idf j)$
proof –
fix j **assume** $jin: j \in J$
then have $idf j \cap idf i = \{\}$ **using** jss **by** *auto*
then have $\bigwedge v . v \in idf j \implies v \notin ?e$ **by** *auto*
then have $\bigwedge v . v \in idf j \implies v \in \mathcal{V} - ?e$ **using** jss *wellformed*
by (*metis (no-types, lifting) DiffI ‹j ∈ J› basic-trans-rules(31) iedge*
mem-Collect-eq)
then have $\bigwedge v . v \in idf j \implies y v = x v$ **using** yin T -def p -def **by** *auto*
then show $\text{mono-edge } y (idf j)$ **using** $xmono jin$
by (*simp add: mono-edge-def*)
qed
moreover have $y \in \mathcal{C}^2$
using $Tss yin$ **by** *auto*
ultimately show $y \in \bigcap (Ae \text{ ' } J)$ **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 xin **by** *auto*
next
fix x **assume** $x \in (\bigcup p \in Ps . T p)$
then show $x \in \bigcap (Ae \text{ ' } J)$ **unfolding** Ps -def **by** *auto*
qed
moreover have $dfo: \text{disjoint-family-on } (\lambda p . T p) Ps$
using $psss Pdjnt \text{disjoint-family-on-def}$ **by** *blast*
moreover have $(\lambda p . T p) \text{ ' } Ps \subseteq P.\text{events}$
unfolding T -def Ps -def **using** $P.\text{sets-eq}$ **by** *auto*
moreover have $finPs: \text{finite } Ps$ **using** $finP psss \text{finite-subset}$ **by** *auto*
ultimately have $P.\text{prob } (\bigcap (Ae \text{ ' } J)) = (\sum p \in Ps . P.\text{prob } (T p))$
using $P.\text{finite-measure-finite-Union}[of Ps \lambda p . T p]$ **by** *simp*
moreover have $\bigwedge p . p \in Ps \implies P.\text{prob } (T p) = \text{card } (T p) / \text{card } (\mathcal{C}^2)$
using $\text{measure-uniform-count-measure}[of \mathcal{C}^2]$ Tss
by (*simp add: P.MU-def P.fin-Ω*)
ultimately have $P.\text{prob } (\bigcap (Ae \text{ ' } J)) = (\sum p \in Ps . \text{real } (\text{card } (T p))) / (\text{card } (\mathcal{C}^2))$
using $\text{sum-divide-distrib}[of - Ps \text{card } (\mathcal{C}^2)]$ **by** (*simp*)
also have $\dots = (\sum p \in Ps . 2 \text{ powi } (\text{card } ?e)) / (\text{card } (\mathcal{C}^2))$
using $cp psss$ **by** (*simp add: Ps-def*)
finally have $P.\text{prob } (\bigcap (Ae \text{ ' } J)) = (\text{card } Ps * (2 \text{ powi } (\text{card } ?e))) / (\text{card } (\mathcal{C}^2))$
by *simp*
then have $P.\text{prob } (Ae i) * P.\text{prob } (\bigcap (Ae \text{ ' } J)) =$
 $1 / (2 \text{ powi } (\text{int } (\text{card } ?e)) - 1) * ((\text{card } Ps * (2 \text{ powi } (\text{card } ?e))) / (\text{card } (\mathcal{C}^2)))$

$(\mathcal{C}^2))$
using *iin p1 by simp*
also have $\dots = ((2 \text{ powi } (\text{card } ?e)) / (2 \text{ powi } (\text{int } (\text{card } (?e)) - 1))) * (\text{card } Ps / \text{card } (\mathcal{C}^2))$
by (*simp add: field-simps*)
also have $\dots = 2 * (\text{card } Ps / \text{card } (\mathcal{C}^2))$
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 ($\bigcap (Ae \text{ ' } J)$) = (card Ps * 2) / (card (\mathcal{C}^2))* **by** *simp*
have $(Ae \text{ } i) \cap (\bigcap (Ae \text{ ' } J)) = (\bigcup p \in Ps . ((Ae \text{ } i) \cap T p))$
using *bunrep by auto*
moreover have *disjoint-family-on* ($\lambda p . (Ae \text{ } i) \cap T p$) *Ps*
using *dfo disjoint-family-on-bisimulation[of T Ps ($\lambda p . (Ae \text{ } i) \cap T p$)] by auto
moreover have ($\lambda p . (Ae \text{ } i) \cap T p$) ' *Ps* $\subseteq P.\text{events}$
unfolding *T-def using P.sets-eq by auto*
ultimately have $P.\text{prob } ((Ae \text{ } i) \cap (\bigcap (Ae \text{ ' } J))) = (\sum p \in Ps . P.\text{prob } ((Ae \text{ } i) \cap T p))$
using *P.finite-measure-finite-Union[of Ps $\lambda p . (Ae \text{ } i) \cap T p$] finPs by simp*
moreover have *tss2: $\bigwedge p . (Ae \text{ } i) \cap T p \subseteq \mathcal{C}^2$ using Tss by auto*
moreover have $\bigwedge p . p \in Ps \implies P.\text{prob } ((Ae \text{ } i) \cap T p) = \text{card } ((Ae \text{ } i) \cap T p) / \text{card } (\mathcal{C}^2)$
using *measure-uniform-count-measure[of \mathcal{C}^2] tss2 P.MU-def P.fin- Ω by simp*
ultimately have $P.\text{prob } ((Ae \text{ } i) \cap (\bigcap (Ae \text{ ' } J))) = (\sum p \in Ps . \text{card } ((Ae \text{ } i) \cap T p)) / (\text{card } (\mathcal{C}^2))$
using *sum-divide-distrib[of - Ps card (\mathcal{C}^2)] by (simp)*
moreover have $\bigwedge p . p \in Ps \implies \text{card } ((Ae \text{ } i) \cap (T p)) = 2$
proof –
fix *p* **assume** $p \in Ps$
define *h* **where** $h \equiv \lambda c . (\lambda v . \text{if } (v \in \mathcal{V}) \text{ then } (\text{if } (v \in ?e) \text{ then } c \text{ else } p \ v) \text{ else } \text{undefined})$
have *hc: $\bigwedge c \ v . v \in ?e \implies h \ c \ v = c$* **unfolding** *h-def using wellformed iin iedge by auto*
have *hne: $\bigwedge c1 \ c2 . c1 \neq c2 \implies h \ c1 \neq h \ c2$*
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 $\bigwedge v . v \in ?e \implies h \ c1 \ v = c1$ **using** *hc by simp*
then have $\bigwedge v . v \in ?e \implies c1 = c2$ **using** *hc eq by auto*
then show *False using ne eq using V-nempty blocks-nempty iedge iin by blast*
qed
then have *hdjnt: $\bigwedge n . (\forall c1 \in \{0..<n\} . \forall c2 \in \{0..<n\} . c1 \neq c2 \implies \{h \ c1\} \cap \{h \ c2\} = \{\})$*
by *auto*
have *heq: $\bigwedge c . c \in \{0..<2\} \implies \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \ ?e \ c \wedge (\forall v \in (\mathcal{V} - ?e) . f \ v = p \ v)\} = \{h \ c\}$*
proof –
fix *c* **assume** $c \in \{0..<2::\text{nat}\}$*

have $\bigwedge x f. f \in \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c \wedge (\forall v \in (\mathcal{V} - ?e) . f v = p v)\} \implies f x = h c x$
unfolding *h-def* **using** *mono-edge-colD all-n-vertex-colourings-alt* **by** *auto*
then have $\bigwedge f . f \in \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c \wedge (\forall v \in (\mathcal{V} - ?e) . f v = p v)\} \implies f = h c$
by *auto*
moreover have $h c \in \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c \wedge (\forall v \in (\mathcal{V} - ?e) . f v = p v)\}$
proof –
have $h c \in \mathcal{C}^2$ **unfolding** *h-def* **using** *all-n-vertex-colourings-alt*
by (*smt (verit, ccfv-SIG) DiffI P'-def PiE-I PiE-mem (c \in \{0..<2\}) (p \in 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 $\{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c \wedge (\forall v \in (\mathcal{V} - ?e) . f v = p v)\} = \{h c\}$
by *blast*
qed
have $Ae i = (\bigcup c \in \{0..<2\} . \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c\})$
using *Aefn iedge iin mono-edge-set-union[of ?e 2]* **by** *auto*
then have $(Ae i) \cap (T p) = (\bigcup c \in \{0..<2\} . \{f \in \mathcal{C}^2 . \text{mono-edge-col } f \text{ ?}e \text{ } c \wedge (\forall v \in (\mathcal{V} - ?e) . f v = p v)\})$
unfolding *T-def* **by** *auto*
then have $\text{card} ((Ae i) \cap (T p)) = \text{card} ((\bigcup c \in \{0..<2\} . \{h c\}))$ **using** *heq* **by** *simp*
moreover have $(1:: \text{nat}) * \text{card} \{0..<2:: \text{nat}\} = \text{card} ((\bigcup c \in \{0..<2\} . \{h c\}))$
proof –
have *inj-on* $(\lambda c. \{h c\}) \{0..<2\}$ **using** *hdjnt inj-onI*
by (*metis (mono-tags, lifting) hne insertI1 singletonD*)
then show *?thesis*
using *card-partition-image-inj*[of $\{0..<2\} \lambda c . \{h c\} 1:: \text{nat}$] *hdjnt* **by** *auto*
qed
ultimately show $\text{card} ((Ae i) \cap (T p)) = 2$ **by** *auto*
qed
ultimately show $P.\text{prob} (Ae i \cap \bigcap (Ae ' J)) = P.\text{prob} (Ae i) * P.\text{prob} (\bigcap (Ae ' J))$
using *prob* **by** *simp*
qed
qed
then show *?thesis* **using** *P.MU-def* **by** *auto*
qed

lemma *intersect-empty-set-size*:

assumes $\bigwedge e . e \in \#E \implies \text{size} \{\# f \in \# (E - \{\#e\}) . f \cap e \neq \{\#\} \} \leq d$

assumes $e2 \in \# E$
shows $\text{size } \{\#e \in \# E . e \cap e2 = \{\} \# \} \geq \text{size } E - d - 1$ (**is** $\text{size } ?S' \geq \text{size } E - d - 1$)
proof –
have $a1alt: \bigwedge e . e \in \# E \implies \text{size } \{\# f \in \# E . f \cap e \neq \{\} \# \} \leq d + 1$
using *edge-intersect-count-inc* *assms(1)* *block-intersect-count-def* **by** *force*
have $E = ?S' + \{\#e \in \# E . e \cap e2 \neq \{\} \# \}$ **by** *auto*
then have $\text{size } E = \text{size } ?S' + \text{size } \{\#e \in \# E . e \cap e2 \neq \{\} \# \}$
by (*metis size-union*)
then have $\text{size } ?S' = \text{size } E - \text{size } \{\#e \in \# E . e \cap e2 \neq \{\} \# \}$ **by** *simp*
moreover have $\text{size } \{\#e \in \# E . e \cap e2 \neq \{\} \# \} \leq d + 1$ **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 $\bigwedge e . e \in \# E \implies \text{card } e \geq k$

assumes $\bigwedge e . e \in \# E \implies \text{size } \{\# f \in \# (E - \{e\}) . f \cap e \neq \{\} \# \} \leq d$

assumes $\exp(1) * (d+1) \leq (2 \text{ powi } (k - 1))$

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 $?N = \{0..<\text{size } E\}$

obtain id **where** *ideq: image-mset id (mset-set ?N) = E* **and** *idin: id ∈ ?N →_E set-mset E*

using *obtain-function-on-ext-funcset[of ?N E]* **by** *auto*

then have *iedge: $\bigwedge i . i \in ?N \implies id \ i \in \# E$* **by** *auto*

– 2 define event

define Ae **where** $Ae \equiv \lambda i . \{f \in \mathcal{C}^2 . \text{mono-edge } f \ (id \ i)\}$

– (3) Prove each event A is mutually independent of all other mono events for other edges that don't intersect.

have $0 < P.\text{prob} (\bigcap Ai \in ?N . \text{space } P.MU - Ae \ Ai)$

proof (*intro P.lovasz-local-symmetric[of ?N Ae d (1/(2 powi (k-1)))]*)

have *mis: $\bigwedge i . i \in ?N \implies P.\text{mutual-indep-events} (Ae \ i) \ Ae \ (\{j \in ?N . (id \ j \cap id \ i) = \{\}\})$*

using *disjoint-set-is-mutually-independent[of - id Ae] P.MU-def* *assms idin* **by** (*simp add: Ae-def*)

then show $\bigwedge i . i \in ?N \implies \exists S . S \subseteq ?N - \{i\} \wedge \text{card } S \geq \text{card } ?N - d - 1 \wedge$

$P.\text{mutual-indep-events} (Ae \ i) \ Ae \ S$

proof –

fix i **assume** *iin: i ∈ ?N*

define S' **where** $S' \equiv \{j \in ?N . (id \ j) \cap (id \ i) = \{\}\}$

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 ∩*
(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 $\bigwedge i. i \in ?N \implies P.prob(Ae i) \leq 1/(2 \text{ powi } (k-1))$
unfolding *Ae-def* **using** *P.prob-monochromatic-edge-bound[of - k]* *iedge*
assms(4) *assms(1)* **by** *auto*
show $exp(1) * (1 / 2 \text{ powi int } (k - 1)) * (d + 1) \leq 1$
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** $\bigwedge i. i \in ?N \implies \neg mono-edge f (id i)$
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 $\bigwedge e. e \in \# E \implies \neg mono-edge f e$
using *iedq mset-set-implies[of id ?N E λ e. ¬ 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 $k \geq 9$

lemma *exp-ineq-k9:*

fixes $k:: nat$

assumes $k \geq 9$

shows $exp(1::real) * (k * (k - 1) + 1) < 2^{\wedge(k-1)}$

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 $eq: \{\# f \in \# (E - \{\#e\#\}) . f \cap e \neq \{\#\}\} \subseteq \# (\sum v \in \# (mset-set e) . \{\# f \in \# (E - \{\#e\#\}) . v \in f \#\})$
proof (*intro mset-subset-eq-elemI*)
fix a **assume** $a \in \# \{\# f \in \# remove1-mset e E . f \cap e \neq \{\#\}\}$ (**is** $a \in \# ?E'$)
then have $ain: a \in \# remove1-mset e E$ **and** $a \cap e \neq \{\}$
using *mset-obtain-from-filter* **by** *fast+*
then obtain v **where** $v \in a$ **and** $v \in e$ **by** *blast*
then have $vvertex: v \in \mathcal{V}$ **using** *wellformed[of e] assms(1)* **by** *auto*
have $count ?E' a \leq count \{\# f \in \# E - \{\#e\#\} . v \in f \#\} a$
by (*metis* $\langle a \in \# ?E' \rangle \langle v \in a \rangle$ *count-filter-mset le-eq-less-or-eq mset-obtain-from-filter(2)*)
moreover have $count \{\# f \in \# E - \{\#e\#\} . v \in f \#\} a \leq$
 $(\sum v \in \# (mset-set e) . count \{\# f \in \# (E - \{\#e\#\}) . v \in f \#\} a)$
by (*metis* $\langle v \in e \rangle$ *assms(1) finite-blocks finite-set-mset-mset-set sum-image-mset-mono-mem*)

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

corollary *erdos-propertyB-LLL9*:
assumes $k \geq 9$
shows *has-property-B*
proof –
define d **where** $d = k*(k-1)$
have $\bigwedge e . e \in \# E \implies card e \geq k$
using *uniform* **by** *simp*
moreover have $\bigwedge e . e \in \# E \implies size \{\# f \in \# (E - \{\#e\#\}) . f \cap e \neq \{\#\}\} \leq d$
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/.