# Expander Graphs

Emin Karayel

March 17, 2025

**Abstract**

Expander Graphs are low-degree graphs that are highly connected. They have diverse applications, for example in derandomization and pseudo-randomness, error-correcting codes, as well as pure mathematical subjects such as metric embeddings. This entry formalizes the concept and derives main theorems about them such as Cheeger's inequality or tail bounds on distribution of random walks on them. It includes a strongly explicit construction for every size and spectral gap. The latter is based on the Margulis-Gabber-Galil graphs and several graph operations that preserve spectral properties. The proofs are based on the survey papers/monographs by Hoory et al. [4] and Vadhan [11], as well as results from Impagliazzo and Kabanets [5] and Murtagh et al. [9]

## Contents

**1 Introduction** **2**

**2 Preliminary Results** **2**
2.1 Constructive Chernoff Bound . . . . . 2
2.2 Congruence Method . . . . . 12
2.3 Multisets . . . . . 13

**3 Definitions** **17**

**4 Setup for Types to Sets** **35**

**5 Algebra-only Theorems** **37**

**6 Spectral Theory** **54**

**7 Cheeger Inequality** **80**

**8 Margulis Gabber Galil Construction** **92**

**9 Random Walks** **114**

**10 Graph Powers** **124**

**11 Strongly Explicit Expander Graphs** **134**

**12 Expander Walks as Pseudorandom Objects** **152**

# 1 Introduction

A good introduction into Expander Graphs can be found in the survey article by Hoory et al. [4]: An expander graph is an infinite family of undirected regular graphs[1] with increasing sizes, but contant degrees, all fulfilling a non-trivial expansion condition consistently. Most common are the following expansion conditions:

- One-sided spectral expansion – an upper-bound on the second largest eigenvalue $\lambda_2$ of the adjacency matrix,
- Two-sided spectral expansion – an upper-bound on the absolute value of both $\lambda_2$ and $\lambda_n$ the smallest eigenvalue,
- Edge expansion – a lower-bound on the relative count of edges between any subset and its complement.

There are various implications between the three types of families, most notably the Cheeger inequality, which relates edge-expansion to (one-sided) spectral expansion. (Section 7)

This entry formalizes

- definitions for the expansion conditions, as well as proofs for the relations between them,
- a construction and proofs of spectral expansion of the Margulis-Gabber-Galil expander (Section 8), and
- proofs of how expansion-properties are affected by graph operations (Sections 10 and 11).

And concludes with a consturction of strongly explicit expanders for every size and spectral gap with asymptotically optimal degree (Section 11).

It also includes a proof of the hitting property, i.e., tail-bounds for the probability that a random walk in an expander graph ramains inside a given subset, as well as Chernoff-type bounds on the number of times a given subset will be hit by a random walk. (Section 9)

The basis for the graph theory relies on the formalization by Lars Noschinski [10]. Most of the algabraic development is carried out in the type-based formalization of linear algebra in "HOL-Analysis". To achieve that I have transferred some results from the set based world into the type-based world - most notably unified diagonalization of commuting hermitian matrices by Echenim [2] (Section 6). The transfer happens using the pre-exisiting framework by Divasón et al. [1].

On the otherhand, results that are obtained using the stochastic matrix, but do not explicitly reference it are transferred back into purely graph-theoretic theorems using the Types-To-Sets mechanism by Kunčar and Popescu [7] (Section 4), i.e., the stochastic matrix is defined using a local type (isomorphic to the vertex set.)

# 2 Preliminary Results

## 2.1 Constructive Chernoff Bound

This section formalizes Theorem 5 by Impagliazzo and Kabanets [5]. It is a general result with which Chernoff-type tail bounds for various kinds of weakly dependent random variables can be obtained. The results here are general and will be applied in Section 9 to random walks in expander graphs.

**theory** *Constructive-Chernoff-Bound*
  **imports**
    *HOL−Probability.Probability-Measure*
    *Universal-Hash-Families.Universal-Hash-Families-More-Product-PMF*
    *Weighted-Arithmetic-Geometric-Mean.Weighted-Arithmetic-Geometric-Mean*
**begin**

**lemma** *powr-mono-rev*:
  **fixes** $x$ :: *real*

[1] A graph is regular if every node has the same degree.

**assumes** $a \le b$ **and** $x > 0$ $x \le 1$
**shows** $x$ *powr* $b \le x$ *powr* $a$
**proof** –
**have** $x$ *powr* $b = (1/x)$ *powr* $(-b)$
**using** *assms* **by** (*simp add*: *powr-divide powr-minus-divide*)
**also have** ... $\le (1/x)$ *powr* $(-a)$
**using** *assms* **by** (*intro powr-mono*) *auto*
**also have** ... $= x$ *powr* $a$
**using** *assms* **by** (*simp add*: *powr-divide powr-minus-divide*)
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *exp-powr*: (*exp* $x$) *powr* $y$ = *exp* $(x*y)$ **for** $x$ :: *real*
**unfolding** *powr-def* **by** *simp*

**lemma** *integrable-pmf-iff-bounded*:
**fixes** $f$ :: $'a \Rightarrow$ *real*
**assumes** $\bigwedge x.\ x \in$ *set-pmf* $p \Longrightarrow$ *abs* $(f\ x) \le C$
**shows** *integrable* (*measure-pmf* $p$) $f$
**proof** –
**obtain** $x$ **where** $x \in$ *set-pmf* $p$
**using** *set-pmf-not-empty* **by** *fast*
**hence** $C \ge 0$ **using** *assms*(*1*) **by** *fastforce*
**hence** $(\int^+ x.$ *ennreal* (*abs* $(f\ x)$) $\partial$*measure-pmf* $p) \le (\int^+ x.\ C\ \partial$*measure-pmf* $p)$
**using** *assms ennreal-le-iff*
**by** (*intro nn-integral-mono-AE AE-pmfI*) *auto*
**also have** ... $= C$
**by** *simp*
**also have** ... $<$ *Orderings.top*
**by** *simp*
**finally have** $(\int^+ x.$ *ennreal* (*abs* $(f\ x)$) $\partial$*measure-pmf* $p) <$ *Orderings.top* **by** *simp*
**thus** *?thesis*
**by** (*intro iffD2*[*OF integrable-iff-bounded*]) *auto*
**qed**

**lemma** *split-pair-pmf*:
*measure-pmf.prob* (*pair-pmf* $A$ $B$) $S$ = $integral^L$ $A$ ($\lambda a.$ *measure-pmf.prob* $B$ $\{b.\ (a,b) \in S\}$)
(**is** $?L = ?R$)
**proof** –
**have** $a$:*integrable* (*measure-pmf* $A$) ($\lambda x.$ *measure-pmf.prob* $B$ $\{b.\ (x,\ b) \in S\}$)
**by** (*intro integrable-pmf-iff-bounded*[**where** $C$=*1*]) *simp*

**have** $?L = (\int^+ x.$ *indicator* $S$ $x$ $\partial$(*measure-pmf* (*pair-pmf* $A$ $B$)))
**by** (*simp add*: *measure-pmf.emeasure-eq-measure*)
**also have** ... $= (\int^+ x. (\int^+ y.$ *indicator* $S$ $(x,y)$ $\partial B)$ $\partial A)$
**by** (*simp add*: *nn-integral-pair-pmf'*)
**also have** ... $= (\int^+ x. (\int^+ y.$ *indicator* $\{b.\ (x,b) \in S\}$ $y$ $\partial B)$ $\partial A)$
**by** (*simp add*:*indicator-def*)
**also have** ... $= (\int^+ x.$ (*measure-pmf.prob* $B$ $\{b.\ (x,b) \in S\}$) $\partial A)$
**by** (*simp add*: *measure-pmf.emeasure-eq-measure*)
**also have** ... $= ?R$
**using** $a$
**by** (*subst nn-integral-eq-integral*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *split-pair-pmf-2*:
*measure*(*pair-pmf* $A$ $B$) $S$ = $integral^L$ $B$ ($\lambda a.$ *measure-pmf.prob* $A$ $\{b.\ (b,a) \in S\}$)

```
  (is ?L = ?R)
proof -
  have ?L = measure (pair-pmf B A) {ω. (snd ω, fst ω) ∈ S}
    by (subst pair-commute-pmf) (simp add:vimage-def case-prod-beta)
  also have ... = ?R
    unfolding split-pair-pmf by simp
  finally show ?thesis by simp
qed

definition KL-div :: real ⇒ real ⇒ real
  where KL-div p q = p ∗ ln (p/q) + (1−p) ∗ ln ((1−p)/(1−q))

theorem impagliazzo-kabanets-pmf:
  fixes Y :: nat ⇒ 'a ⇒ bool
  fixes p :: 'a pmf
  assumes n > 0
  assumes ⋀i. i ∈ {..<n} ⟹ δ i ∈ {0..1}
  assumes ⋀S. S ⊆ {..<n} ⟹ measure p {ω. (∀ i ∈ S. Y i ω)} ≤ (∏ i ∈ S. δ i)
  defines δ-avg ≡ (∑ i∈ {..<n}. δ i)/n
  assumes γ ∈ {δ-avg..1}
  assumes δ-avg > 0
  shows measure p {ω. real (card {i ∈ {..<n}. Y i ω}) ≥ γ ∗ n} ≤ exp (−real n ∗ KL-div γ
δ-avg)
    (is ?L ≤ ?R)
proof -
  let ?n = real n
  define q :: real where q = (if γ = 1 then 1 else (γ−δ-avg)/(γ∗(1−δ-avg)))

  define g where g ω = card {i. i < n ∧ ¬Y i ω} for ω
  let ?E = (λω. real (card {i. i < n ∧ Y i ω}) ≥ γ ∗ n)
  let ?Ξ = prod-pmf {..<n} (λ-. bernoulli-pmf q)

  have q-range:q ∈{0..1}
  proof (cases γ < 1)
    case True
    then show ?thesis
      using assms(5,6)
      unfolding q-def by (auto intro!:divide-nonneg-pos simp add:algebra-simps)
  next
    case False
    hence γ = 1 using assms(5) by simp
    then show ?thesis unfolding q-def by simp
  qed

  have abs-pos-le-1I: abs x ≤ 1 if x ≥ 0 x ≤ 1 for x :: real
    using that by auto

  have γ-n-nonneg: γ∗?n ≥ 0
    using assms(1,5,6) by simp
  define r where r = n − nat ⌈γ∗n⌉

  have 2:(1−q) ^r ≤ (1−q)^ g ω if ?E ω for ω
  proof -
    have g ω = card ({i. i < n} − {i. i < n ∧ Y i ω})
      unfolding g-def by (intro arg-cong[where f=λx. card x]) auto
    also have ... = card {i. i < n} − card {i. i < n ∧ Y i ω}
      by (subst card-Diff-subset, auto)
    also have ... ≤ card {i. i < n} − nat ⌈γ∗n⌉
```

```
      using that γ-n-nonneg by (intro diff-le-mono2) simp
    also have ... = r
      unfolding r-def by simp
    finally have g ω ≤ r by simp
    thus (1−q) ^ r ≤ (1−q) ^(g ω)
      using q-range by (intro power-decreasing) auto
  qed

  have γ-gt-0: γ > 0
    using assms(5,6) by simp

  have q-lt-1: q < 1 if γ < 1
  proof −
    have δ-avg < 1 using assms(5) that by simp
    hence (γ − δ-avg) / (γ ∗ (1 − δ-avg)) < 1
      using γ-gt-0 assms(6) that
      by (subst pos-divide-less-eq) (auto simp add:algebra-simps)
    thus q < 1
      unfolding q-def using that by simp
  qed

  have 5: (δ-avg ∗ q + (1−q)) / (1−q) powr (1−γ) = exp (− KL-div γ δ-avg) (is ?L1 = ?R1)
    if γ < 1
  proof −
    have δ-avg-range: δ-avg ∈ {0<..<1}
      using that assms(5,6)  by simp

    have ?L1 = (1 − (1−δ-avg) ∗ q) / (1−q) powr (1−γ)
      by (simp add:algebra-simps)
    also have ... = (1 − (γ−δ-avg) / γ ) / (1−q) powr (1−γ)
      unfolding q-def using that γ-gt-0 δ-avg-range by simp
    also have ... = (δ-avg / γ) / (1−q) powr (1−γ)
      using γ-gt-0 by (simp add:divide-simps)
    also have ... = (δ-avg / γ) ∗ (1/(1−q)) powr (1−γ)
      using q-lt-1[OF that] by (subst powr-divide, simp-all)
    also have ... = (δ-avg / γ) ∗ (1/((γ∗(1−δ-avg)−(γ−δ-avg))/(γ∗(1−δ-avg)))) powr (1−γ)
      using γ-gt-0 δ-avg-range unfolding q-def by (simp add:divide-simps)
    also have ... = (δ-avg / γ) ∗ ((γ / δ-avg) ∗((1−δ-avg)/(1−γ))) powr (1−γ)
      by (simp add:algebra-simps)
    also have ... = (δ-avg / γ) ∗ (γ / δ-avg) powr (1−γ) ∗((1−δ-avg)/(1−γ)) powr (1−γ)
      using γ-gt-0  δ-avg-range that by (subst powr-mult, auto)
     also have ... = (δ-avg / γ) powr 1 ∗ (δ-avg / γ) powr −(1−γ) ∗((1−δ-avg)/(1−γ)) powr
(1−γ)
      using γ-gt-0 δ-avg-range that unfolding powr-minus-divide by (simp add:powr-divide)
    also have ... = (δ-avg / γ) powr γ ∗((1−δ-avg)/(1−γ)) powr (1−γ)
      by (subst powr-add[symmetric]) simp
    also have ... = exp ( ln ((δ-avg / γ) powr γ ∗((1−δ-avg)/(1−γ)) powr (1−γ)))
      using γ-gt-0 δ-avg-range that by (intro exp-ln[symmetric] mult-pos-pos) auto
    also have ... =  exp ((ln ((δ-avg / γ) powr γ) + ln (((1 − δ-avg) / (1 − γ)) powr (1−γ))))
      using γ-gt-0 δ-avg-range that by (subst ln-mult) auto
    also have ... =  exp ((γ ∗ ln (δ-avg / γ) + (1 − γ) ∗ ln ((1 − δ-avg) / (1 − γ))))
      using γ-gt-0 δ-avg-range that by (simp add:ln-powr algebra-simps)
    also have ... =  exp (− (γ ∗ ln (γ / δ-avg) + (1 − γ) ∗ ln ((1 − γ) / (1 − δ-avg))))
      using γ-gt-0 δ-avg-range that by (simp add: ln-div algebra-simps)
    also have ... = ?R1
      unfolding KL-div-def by simp

    finally show ?thesis by simp
```

```
qed

have 3: (δ-avg ∗ q + (1−q)) ^ n / (1−q) ^ r ≤ exp (− ?n∗ KL-div γ δ-avg) (is ?L1 ≤ ?R1)
proof (cases γ < 1)
  case True
  have γ ∗ real n ≤ 1 ∗ real n
    using True by (intro mult-right-mono) auto
  hence r = real n − real (nat ⌈γ ∗ real n⌉)
    unfolding r-def by (subst of-nat-diff) auto
  also have ... = real n − ⌈γ ∗ real n⌉
    using γ-n-nonneg by (subst of-nat-nat, auto)
  also have ... ≤ ?n − γ ∗ ?n
    by (intro diff-mono) auto
  also have ... = (1−γ) ∗?n by (simp add:algebra-simps)
  finally have r-bound: r ≤ (1−γ)∗n by simp

  have ?L1 = (δ-avg ∗ q + (1−q)) ^ n / (1−q) powr r
    using q-lt-1[OF True] assms(1) by (simp add: powr-realpow)
  also have ... = (δ-avg ∗ q + (1−q)) powr n / (1−q) powr r
    using q-lt-1[OF True] assms(6) q-range
    by (subst powr-realpow[symmetric], auto intro!:add-nonneg-pos)
  also have ... ≤ (δ-avg ∗ q + (1−q)) powr n / (1−q) powr ((1−γ)∗n)
    using q-range q-lt-1[OF True] by (intro divide-left-mono powr-mono-rev r-bound) auto
  also have ... = (δ-avg ∗ q + (1−q)) powr n / ((1−q) powr (1−γ)) powr n
    unfolding powr-powr by simp
  also have ... = ((δ-avg ∗ q + (1−q)) / (1−q) powr (1−γ)) powr n
    using assms(6) q-range by (subst powr-divide) auto
  also have ... = exp (− KL-div γ δ-avg) powr real n
    unfolding 5[OF True] by simp
  also have ... = ?R1
    unfolding exp-powr by simp
  finally show ?thesis by simp
next
  case False
  hence γ-eq-1: γ=1 using assms(5) by simp
  have ?L1 = δ-avg ^ n
    using γ-eq-1 r-def q-def by simp
  also have ... = exp( − KL-div 1 δ-avg) ^ n
    unfolding KL-div-def using assms(6) by (simp add:ln-div)
  also have ... = ?R1
    using γ-eq-1 by (simp add: powr-realpow[symmetric] exp-powr)
  finally show ?thesis by simp
qed

have 4: (1 − q) ^ r > 0
proof (cases γ < 1)
  case True
  then show ?thesis using q-lt-1[OF True] by simp
next
  case False
  hence γ=1 using assms(5) by simp
  hence r=0 unfolding r-def by simp
  then show ?thesis by simp
qed

have (1−q) ^ r ∗ ?L = (∫ω. indicator {ω. ?E ω} ω ∗ (1−q) ^ r ∂p)
  by simp
also have ... ≤ (∫ω. indicator {ω. ?E ω} ω ∗ (1−q) ^ g ω ∂p)
```

**using** *q-range 2* **by** (*intro integral-mono-AE integrable-pmf-iff-bounded*[**where** $C=1$] *abs-pos-le-1I mult-le-one power-le-one AE-pmfI*) (*simp-all split*:*split-indicator*)
**also have** ... $= (\int \omega.\ indicator\ \{\omega.\ ?E\ \omega\}\ \omega * (\prod i \in \{i.\ i < n \wedge \neg Y\ i\ \omega\}.\ (1-q))\ \partial p)$
**unfolding** *g-def* **using** *q-range*
**by** (*intro integral-cong-AE AE-pmfI*, *simp-all add*:*powr-realpow*)
**also have** ... $= (\int \omega.\ indicator\ \{\omega.\ ?E\ \omega\}\ \omega * measure\ ?\Xi\ (\{j.\ j < n \wedge \neg Y\ j\ \omega\} \rightarrow \{False\})\ \partial p)$
**using** *q-range* **by** (*subst prob-prod-pmf'*) (*auto simp add*:*measure-pmf-single*)
**also have** ... $= (\int \omega.\ measure\ ?\Xi\ \{\xi.\ ?E\ \omega \wedge (\forall i \in \{j.\ j < n \wedge \neg Y\ j\ \omega\}.\ \neg \xi\ i)\}\ \partial p)$
**by** (*intro integral-cong-AE AE-pmfI*, *simp-all add*:*Pi-def split*:*split-indicator*)
**also have** ... $= (\int \omega.\ measure\ ?\Xi\ \{\xi.\ ?E\ \omega \wedge (\forall i \in \{..<n\}.\ \xi\ i \longrightarrow Y\ i\ \omega)\}\ \partial p)$
**by** (*intro integral-cong-AE AE-pmfI measure-eq-AE*) *auto*
**also have** ... $= measure\ (pair\text{-}pmf\ p\ ?\Xi)\ \{\varphi.?E\ (fst\ \varphi) \wedge (\forall i \in \{..<n\}.\ snd\ \varphi\ i \longrightarrow Y\ i\ (fst\ \varphi))\}$
**unfolding** *split-pair-pmf* **by** *simp*
**also have** ... $\leq measure\ (pair\text{-}pmf\ p\ ?\Xi)\ \{\varphi.\ (\forall i \in \{j.\ j < n \wedge snd\ \varphi\ j\}.\ Y\ i\ (fst\ \varphi))\}$
**by** (*intro pmf-mono*, *auto*)
**also have** ... $= (\int \xi.\ measure\ p\ \{\omega.\ \forall i \in \{j.\ j < n \wedge \xi\ j\}.\ Y\ i\ \omega\}\ \partial\ ?\Xi)$
**unfolding** *split-pair-pmf-2* **by** *simp*
**also have** ... $\leq (\int a.\ (\prod i \in \{j.\ j < n \wedge a\ j\}.\ \delta\ i)\ \partial\ ?\Xi)$
**using** *assms*(*2*) **by** (*intro integral-mono-AE AE-pmfI assms*(*3*) *subsetI prod-le-1 prod-nonneg integrable-pmf-iff-bounded*[**where** $C=1$] *abs-pos-le-1I*) *auto*
**also have** ... $= (\int a.\ (\prod i \in \{..<n\}.\ \delta\ i\hat{}\ of\text{-}bool(a\ i))\ \partial\ ?\Xi)$
**unfolding** *of-bool-def* **by** (*intro integral-cong-AE AE-pmfI*)
(*auto simp add*:*if-distrib prod.If-cases Int-def*)
**also have** ... $= (\prod i<n.\ (\int a.\ (\delta\ i\ \hat{}\ of\text{-}bool\ a)\ \partial(bernoulli\text{-}pmf\ q)))$
**using** *assms*(*2*) **by** (*intro expectation-prod-Pi-pmf integrable-pmf-iff-bounded*[**where** $C=1$]) *auto*
**also have** ... $= (\prod i<n.\ \delta\ i * q + (1-q))$
**using** *q-range* **by** *simp*
**also have** ... $= (root\ (card\ \{..<n\})\ (\prod i<n.\ \delta\ i * q + (1-q)))\ \hat{}\ (card\ \{..<n\})$
**using** *assms*(*1*,*2*) *q-range* **by** (*intro real-root-pow-pos2*[*symmetric*] *prod-nonneg*) *auto*
**also have** ... $\leq ((\sum i<n.\ \delta\ i * q + (1-q))/card\{..<n\})\hat{}(card\ \{..<n\})$
**using** *assms*(*1*,*2*) *q-range* **by** (*intro power-mono arithmetic-geometric-mean*)
(*auto intro*: *prod-nonneg*)
**also have** ... $= ((\sum i<n.\ \delta\ i * q)/n + (1-q))\hat{}n$
**using** *assms*(*1*) **by** (*simp add*:*sum.distrib divide-simps mult.commute*)
**also have** ... $= (\delta\text{-}avg * q + (1-q))\hat{}n$
**unfolding** *δ-avg-def* **by** (*simp add*: *sum-distrib-right*[*symmetric*])
**finally have** $(1-q)\ \hat{}\ r * ?L \leq (\delta\text{-}avg * q + (1-q))\ \hat{}\ n$ **by** *simp*
**hence** $?L \leq (\delta\text{-}avg * q + (1-q))\ \hat{}\ n\ /\ (1-q)\ \hat{}\ r$
**using** *4* **by** (*subst pos-le-divide-eq*) (*auto simp add*:*algebra-simps*)
**also have** ... $\leq ?R$
**by** (*intro 3*)
**finally show** *?thesis* **by** *simp*
**qed**

The distribution of a random variable with a countable range is a discrete probability space, i.e., induces a PMF. Using this it is possible to generalize the previous result to arbitrary probability spaces.

**lemma** (**in** *prob-space*) *establish-pmf*:
**fixes** $f :: 'a \Rightarrow 'b$
**assumes** *rv*: *random-variable discrete f*
**assumes** *countable* (*f* ‘ *space M*)
**shows** $distr\ M\ discrete\ f \in \{M.\ prob\text{-}space\ M \wedge sets\ M = UNIV \wedge (AE\ x\ in\ M.\ measure\ M\ \{x\} \neq 0)\}$
**proof** −
**define** $N$ **where** $N = \{x \in space\ M. \neg\ prob\ (f -`\ \{f\ x\} \cap space\ M) \neq 0\}$
**define** $I$ **where** $I = \{z \in (f\ `\ space\ M).\ prob\ (f -`\ \{z\} \cap space\ M) = 0\}$

**have** *countable-I*: *countable I*
  **unfolding** *I-def* **by** (*intro countable-subset*[*OF - assms*(*2*)]) *auto*

**have** *disj*: *disjoint-family-on* (λ*y*. *f* −‘ {*y*} ∩ *space M*) *I*
  **unfolding** *disjoint-family-on-def* **by** *auto*

**have** *N-alt-def*: *N* = (⋃ *y* ∈ *I*. *f* −‘ {*y*} ∩ *space M*)
  **unfolding** *N-def I-def* **by** (*auto simp add*:*set-eq-iff*)
**have** *emeasure M N* = ∫⁺ *y*. *emeasure M* (*f* −‘ {*y*} ∩ *space M*) ∂*count-space I*
  **using** *rv countable-I* **unfolding** *N-alt-def*
  **by** (*subst emeasure-UN-countable*) (*auto simp add*:*disjoint-family-on-def*)
**also have** ... = ∫⁺ *y*. *0* ∂*count-space I*
  **unfolding** *I-def* **using** *emeasure-eq-measure ennreal-0*
  **by** (*intro nn-integral-cong*) *auto*
**also have** ... = *0* **by** *simp*
**finally have** *0*:*emeasure M N* = *0* **by** *simp*

**have** *1*:*N* ∈ *events*
  **unfolding** *N-alt-def* **using** *rv*
  **by** (*intro sets.countable-UN″ countable-I*) *simp*

**have** *AE x in M*. *prob* (*f* −‘ {*f x*} ∩ *space M*) ≠ *0*
  **using** *0 1* **by** (*subst AE-iff-measurable*[*OF - N-def*[*symmetric*]])
**hence** *AE x in M*. *measure* (*distr M discrete f*) {*f x*} ≠ *0*
  **by** (*subst measure-distr*[*OF rv*], *auto*)
**hence** *AE x in distr M discrete f*. *measure* (*distr M discrete f*) {*x*} ≠ *0*
  **by** (*subst AE-distr-iff*[*OF rv*], *auto*)
**thus** *?thesis*
  **using** *prob-space-distr rv* **by** *auto*
**qed**

**lemma** *singletons-image-eq*:
  (λ*x*. {*x*}) ‘ *T* ⊆ *Pow T*
  **by** *auto*

**theorem** (**in** *prob-space*) *impagliazzo-kabanets*:
  **fixes** *Y* :: *nat* ⇒ ′*a* ⇒ *bool*
  **assumes** *n* > *0*
  **assumes** ⋀*i*. *i* ∈ {..<*n*} ⟹ *random-variable discrete* (*Y i*)
  **assumes** ⋀*i*. *i* ∈ {..<*n*} ⟹ δ *i* ∈ {*0*..*1*}
  **assumes** ⋀*S*. *S* ⊆ {..<*n*} ⟹ 𝒫(ω *in M*. (∀ *i* ∈ *S*. *Y i* ω)) ≤ (∏ *i* ∈ *S*. δ *i*)
  **defines** δ*-avg* ≡ (∑ *i*∈ {..<*n*}. δ *i*)/*n*
  **assumes** γ ∈ {δ*-avg*..*1*} δ*-avg* > *0*
  **shows** 𝒫(ω *in M*. *real* (*card* {*i* ∈ {..<*n*}. *Y i* ω}) ≥ γ ∗ *n*) ≤ *exp* (−*real n* ∗ *KL-div* γ δ*-avg*)
    (**is** *?L* ≤ *?R*)
**proof** −
  **define** *f* **where** *f* = (λω *i*. *if i* < *n then Y i* ω *else False*)
  **define** *g* **where** *g* = (λω *i*. *if i* < *n then* ω *i else False*)
  **define** *T* **where** *T* = {ω. (∀ *i*. ω *i* ⟶ *i* < *n*)}

  **have** *g-idem*: *g* ∘ *f* = *f* **unfolding** *f-def g-def* **by** (*simp add*:*comp-def*)

  **have** *f-range*: *f* ∈ *space M* → *T*
    **unfolding** *T-def f-def* **by** *simp*

  **have** *T* = *PiE-dflt* {..<*n*} *False* (λ-. *UNIV*)
    **unfolding** *T-def PiE-dflt-def* **by** *auto*

**hence** *finite T*
  **using** *finite-PiE-dflt* **by** *auto*
**hence** *countable-T*: *countable T*
  **by** (*intro countable-finite*)
**moreover have** *f ‘ space M* $\subseteq$ *T*
  **using** *f-range* **by** *auto*
**ultimately have** *countable-f*: *countable* (*f ‘ space M*)
  **using** *countable-subset* **by** *auto*

**have** *f −‘ y* $\cap$ *space M* $\in$ *events* **if** *t*:*y* $\in$ ($\lambda x.$ $\{x\}$) *‘ T* **for** *y*
**proof** −
  **obtain** *t* **where** $y = \{t\}$ **and** *t-range*: $t \in T$ **using** *t* **by** *auto*
  **hence** *f −‘ y* $\cap$ *space M* = $\{\omega \in$ *space M*. $f\ \omega = t\}$
    **by** (*auto simp add*:*vimage-def*)
  **also have** ... = $\{\omega \in$ *space M*. $(\forall i < n.\ Y\ i\ \omega = t\ i)\}$
    **using** *t-range* **unfolding** *f-def T-def* **by** *auto*
  **also have** ... = $(\bigcap i \in \{..<n\}.\ \{\omega \in$ *space M*. $Y\ i\ \omega = t\ i\})$
    **using** *assms*(*1*) **by** *auto*
  **also have** ... $\in$ *events*
    **using** *assms*(*1*,*2*)
    **by** (*intro sets.countable-INT*) *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**hence** *random-variable* (*count-space T*) *f*
  **using** *sigma-sets-singletons*[*OF countable-T*] *singletons-image-eq f-range*
  **by** (*intro measurable-sigma-sets*[**where** $\Omega$=*T* **and** *A*= ($\lambda x.$ $\{x\}$) *‘ T*]) *simp-all*
**moreover have** *g* $\in$ *measurable discrete* (*count-space T*)
  **unfolding** *g-def T-def* **by** *simp*
**ultimately have** *random-variable discrete* ($g \circ f$)
  **by** *simp*
**hence** *rv*:*random-variable discrete f*
  **unfolding** *g-idem* **by** *simp*

**define** *M′* :: (*nat* $\Rightarrow$ *bool*) *measure*
  **where** *M′* = *distr M discrete f*

**define** $\Omega$ **where** $\Omega$ = *Abs-pmf M′*
**have** *a*:*measure-pmf* (*Abs-pmf M′*) = *M′*
  **unfolding** *M′-def*
  **by** (*intro Abs-pmf-inverse*[*OF establish-pmf*] *rv countable-f*)

**have** *b*:$\{i.\ (i < n \longrightarrow Y\ i\ x) \wedge i < n\} = \{i.\ i < n \wedge Y\ i\ x\}$ **for** *x*
  **by** *auto*

**have** *c*: *measure* $\Omega$ $\{\omega.\ \forall i{\in}S.\ \omega\ i\} \le$ *prod* $\delta$ *S* (**is** *?L1* $\le$ *?R1*) **if** $S \subseteq \{..<n\}$ **for** *S*
**proof** −
  **have** *d*: $i \in S \Longrightarrow i < n$ **for** *i*
    **using** *that* **by** *auto*
  **have** *?L1* = *measure M′* $\{\omega.\ \forall i{\in}S.\ \omega\ i\}$
    **unfolding** $\Omega$*-def a* **by** *simp*
  **also have** ... = $\mathcal{P}(\omega$ *in M*. $(\forall i \in S.\ Y\ i\ \omega))$
    **unfolding** *M′-def* **using** *that d*
    **by** (*subst measure-distr*[*OF rv*]) (*auto simp add*:*f-def Int-commute Int-def*)
  **also have** ... $\le$ *?R1*
    **using** *that assms*(*4*) **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

```
  have ?L = measure M′ {ω. real (card {i. i < n ∧ ω i}) ≥ γ ∗ n}
    unfolding M′-def by (subst measure-distr[OF rv])
      (auto simp add:f-def algebra-simps Int-commute Int-def b)
  also have ... = measure-pmf.prob Ω {ω. real (card {i ∈ {..<n}. ω i}) ≥ γ ∗ n}
    unfolding Ω-def a by simp
  also have ... ≤ ?R
    using assms(1,3,6,7) c unfolding δ-avg-def
    by (intro impagliazzo-kabanets-pmf) auto
  finally show ?thesis by simp
qed
```

Bounds and properties of *KL-div*

```
lemma KL-div-mono-right-aux-1:
  assumes 0 ≤ p p ≤ q q ≤ q′ q′ < 1
  shows KL-div p q−2∗(p−q)^2 ≤ KL-div p q′−2∗(p−q′)^2
proof (cases p = 0)
  case True
  define f′ :: real ⇒ real where f′ = (λx. 1/(1−x) − 4 ∗ x)

  have deriv: ((λq. ln (1/(1−q)) − 2∗q^2) has-real-derivative (f′ x)) (at x)
    if x ∈ {q..q′} for x
  proof −
    have x ∈ {0..<1} using assms that by auto
    thus ?thesis unfolding f′-def by (auto intro!: derivative-eq-intros)
  qed

  have deriv-nonneg: f′ x ≥ 0 if x ∈ {q..q′} for x
  proof −
    have 0:x ∈ {0..<1} using assms that by auto
    have 4 ∗ x∗(1−x) = 1 − 4∗(x−1/2)^2 by (simp add:power2-eq-square field-simps)
    also have ... ≤ 1 by simp
    finally have 4∗x∗(1−x) ≤ 1 by simp
    hence 1/(1−x) ≥ 4∗x using 0 by (simp add: pos-le-divide-eq)
    thus ?thesis unfolding f′-def by auto
  qed

  have ln (1 / (1 − q)) − 2 ∗ q^2 ≤ ln (1 / (1 − q′)) − 2 ∗ q′^2
    using deriv deriv-nonneg by (intro DERIV-nonneg-imp-nondecreasing[OF assms(3)]) auto
  thus ?thesis using True unfolding KL-div-def by simp
next
  case False
  hence p-gt-0: p > 0 using assms by auto

  define f′ :: real ⇒ real where f′ = (λx. (1−p)/(1−x) − p/x + 4 ∗ (p−x))

  have deriv: ((λq. KL-div p q − 2∗(p−q)^2) has-real-derivative (f′ x)) (at x) if x ∈ {q..q′}
    for x
  proof −
    have 0 < p /x  0 < (1 − p) / (1 − x) using that assms p-gt-0 by auto
    thus ?thesis unfolding KL-div-def f′-def by (auto intro!: derivative-eq-intros)
  qed

  have f′-part-nonneg: (1/(x∗(1−x)) − 4) ≥ 0 if x ∈ {0<..<1} for x :: real
  proof −
    have 4 ∗ x ∗ (1−x) = 1 − 4 ∗ (x−1/2)^2 by (simp add:power2-eq-square algebra-simps)
    also have ... ≤ 1 by simp
    finally have 4 ∗ x ∗ (1−x) ≤ 1 by simp
```

**hence** $1/(x*(1-x)) \geq 4$ **using** *that* **by** (*subst pos-le-divide-eq*) *auto*
**thus** *?thesis* **by** *simp*
**qed**

**have** *f'-alt*: $f'\ x = (x-p)*(1/(x*(1-x)) - 4)$ **if** $x \in \{0<..<1\}$ **for** $x$
**proof** −
**have** $f'\ x = (x-p)/(x*(1-x)) + 4 * (p-x)$ **using** *that* **unfolding** *f'-def* **by** (*simp add*:*field-simps*)
**also have** ... $= (x-p)*(1/(x*(1-x)) - 4)$ **by** (*simp add*:*algebra-simps*)
**finally show** *?thesis* **by** *simp*
**qed**

**have** *deriv-nonneg*: $f'\ x \geq 0$ **if** $x \in \{q..q'\}$ **for** $x$
**proof** −
**have** $x \in \{0<..<1\}$ **using** *assms that p-gt-0* **by** *auto*
**have** $f'\ x = (x-p)*(1/(x*(1-x)) - 4)$ **using** *that assms p-gt-0* **by** (*subst f'-alt*) *auto*
**also have** ... $\geq 0$ **using** *that f'-part-nonneg assms p-gt-0* **by** (*intro mult-nonneg-nonneg*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**show** *?thesis* **using** *deriv deriv-nonneg*
**by** (*intro DERIV-nonneg-imp-nondecreasing*[*OF assms*(*3*)]) *auto*
**qed**

**lemma** *KL-div-swap*: *KL-div* $(1-p)$ $(1-q)$ = *KL-div p q*
**unfolding** *KL-div-def* **by** *auto*

**lemma** *KL-div-mono-right-aux-2*:
**assumes** $0 < q'$ $q' \leq q$ $q \leq p$ $p \leq 1$
**shows** *KL-div p q*$-2*(p-q)$^$2 \leq$ *KL-div p q'*$-2*(p-q')$^$2$
**proof** −
**have** *KL-div* $(1-p)$ $(1-q)-2*((1-p)-(1-q))$^$2 \leq$ *KL-div* $(1-p)$ $(1-q')-2*((1-p)-(1-q'))$^$2$
**using** *assms* **by** (*intro KL-div-mono-right-aux-1*) *auto*
**thus** *?thesis* **unfolding** *KL-div-swap* **by** (*auto simp*:*algebra-simps power2-commute*)
**qed**

**lemma** *KL-div-mono-right-aux*:
**assumes** $(0 \leq p \wedge p \leq q \wedge q \leq q' \wedge q' < 1) \vee (0 < q' \wedge q' \leq q \wedge q \leq p \wedge p \leq 1)$
**shows** *KL-div p q*$-2*(p-q)$^$2 \leq$ *KL-div p q'*$-2*(p-q')$^$2$
**using** *KL-div-mono-right-aux-1 KL-div-mono-right-aux-2 assms* **by** *auto*

**lemma** *KL-div-mono-right*:
**assumes** $(0 \leq p \wedge p \leq q \wedge q \leq q' \wedge q' < 1) \vee (0 < q' \wedge q' \leq q \wedge q \leq p \wedge p \leq 1)$
**shows** *KL-div p q* $\leq$ *KL-div p q'* (**is** *?L* $\leq$ *?R*)
**proof** −
**consider** (*a*) $0 \leq p$ $p \leq q$ $q \leq q'$ $q' < 1$ | (*b*) $0 < q'$ $q' \leq q$ $q \leq p$ $p \leq 1$
**using** *assms* **by** *auto*
**hence** *0*: $(p - q)^2 \leq (p - q')^2$
**proof** (*cases*)
**case** *a*
**hence** $(q-p)$^$2 \leq (q' - p)$^$2$ **by** *auto*
**thus** *?thesis* **by** (*simp add*: *power2-commute*)
**next**
**case** *b* **thus** *?thesis* **by** *simp*
**qed**
**have** *?L* = (*KL-div p q* $- 2*(p-q)$^$2) + 2 * (p-q)$^$2$ **by** *simp*
**also have** ... $\leq$ (*KL-div p q'* $- 2*(p-q')$^$2) + 2 * (p-q')$^$2$
**by** (*intro add-mono KL-div-mono-right-aux assms mult-left-mono 0*) *auto*
**also have** ... = *?R* **by** *simp*

**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *KL-div-lower-bound*:
  **assumes** $p \in \{0..1\}$ $q \in \{0<..<1\}$
  **shows** $2*(p-q)\hat{}2 \le$ *KL-div p q*
**proof** −
  **have** $0 \le$ *KL-div p p* $-\ 2 * (p-p)\hat{}2$ **unfolding** *KL-div-def* **by** *simp*
  **also have** ... $\le$ *KL-div p q* $-\ 2 * (p-q)\hat{}2$ **using** *assms* **by** (*intro KL-div-mono-right-aux*) *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**end**

## 2.2 Congruence Method

The following is a method for proving equalities of large terms by checking the equivalence of subterms. It is possible to precisely control which operators to split by.

**theory** *Extra-Congruence-Method*
  **imports**
    *Main*
    *HOL−Eisbach.Eisbach*
**begin**

**datatype** *cong-tag-type* = *CongTag*

**definition** *cong-tag-1* :: $('a \Rightarrow 'b) \Rightarrow$ *cong-tag-type*
  **where** *cong-tag-1 x* = *CongTag*
**definition** *cong-tag-2* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow$ *cong-tag-type*
  **where** *cong-tag-2 x* = *CongTag*
**definition** *cong-tag-3* :: $('a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd) \Rightarrow$ *cong-tag-type*
  **where** *cong-tag-3 x* = *CongTag*

**lemma** *arg-cong3*:
  **assumes** *x1* = *x2 y1* = *y2 z1* = *z2*
  **shows** *f x1 y1 z1* = *f x2 y2 z2*
  **using** *assms* **by** *auto*

**method** *intro-cong* **for** *A* :: *cong-tag-type list* **uses** *more* =
  (*match* (*A*) **in**
      *cong-tag-1 f#h* (*multi*) **for** $f$ :: $'a \Rightarrow 'b$ **and** $h$
        ⇒ ‹*intro-cong h more*:*more arg-cong*[*where f*=*f*]›
    | *cong-tag-2 f#h* (*multi*) **for** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c$ **and** $h$
        ⇒ ‹*intro-cong h more*:*more arg-cong2*[*where f*=*f*]›
    | *cong-tag-3 f#h* (*multi*) **for** $f$ :: $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd$ **and** $h$
        ⇒ ‹*intro-cong h more*:*more arg-cong3*[*where f*=*f*]›
    | - ⇒ ‹*intro more refl*›)

**bundle** *intro-cong-syntax*
**begin**
**notation** *cong-tag-1* (‹$\sigma_1$›)
**notation** *cong-tag-2* (‹$\sigma_2$›)
**notation** *cong-tag-3* (‹$\sigma_3$›)
**end**

**lemma** *restr-Collect-cong*:
  **assumes** $\bigwedge x.\ x \in A \Longrightarrow P\ x = Q\ x$

**shows** $\{x \in A.\ P\ x\} = \{x \in A.\ Q\ x\}$
**using** *assms* **by** *auto*

**end**

## 2.3 Multisets

Some preliminary results about multisets.

**theory** *Expander-Graphs-Multiset-Extras*
**imports**
*HOL−Library.Multiset*
*Extra-Congruence-Method*
**begin**

**unbundle** *intro-cong-syntax*

This is an induction scheme over the distinct elements of a multisets: We can represent each multiset as a sum like: *replicate-mset* $n_1$ $x_1$ + *replicate-mset* $n_2$ $x_2$ + ... + *replicate-mset* $n_k$ $x_k$ where the $x_i$ are distinct.

**lemma** *disj-induct-mset*:
**assumes** $P\ \{\#\}$
**assumes** $\bigwedge n\ M\ x.\ P\ M \Longrightarrow \neg(x \in\#\ M) \Longrightarrow n > 0 \Longrightarrow P\ (M +$ *replicate-mset* $n\ x)$
**shows** $P\ M$
**proof** (*induction size M arbitrary*: *M rule*:*nat-less-induct*)
**case** *1*
**show** *?case*
**proof** (*cases* $M = \{\#\}$)
**case** *True*
**then show** *?thesis* **using** *assms* **by** *simp*
**next**
**case** *False*
**then obtain** *x* **where** *x-def*: $x \in\#\ M$ **using** *multiset-nonemptyE* **by** *auto*
**define** *M1* **where** *M1* = *M* − *replicate-mset* (*count M x*) *x*
**then have** *M-def*: *M* = *M1* + *replicate-mset* (*count M x*) *x*
**by** (*metis count-le-replicate-mset-subset-eq dual-order.refl subset-mset.diff-add*)
**have** *size M1* < *size M*
**by** (*metis M-def x-def count-greater-zero-iff less-add-same-cancel1 size-replicate-mset size-union*)
**hence** *P M1* **using** *1* **by** *blast*
**then show** *P M*
**apply** (*subst M-def*, *rule assms*(*2*), *simp*)
**by** (*simp add*:*M1-def x-def count-eq-zero-iff*[*symmetric*])+
**qed**
**qed**

**lemma** *sum-mset-conv*:
**fixes** $f :: 'a \Rightarrow 'b$::{*semiring-1*}
**shows** *sum-mset* (*image-mset f A*) = *sum* ($\lambda x.$ *of-nat* (*count A x*) $*$ *f x*) (*set-mset A*)
**proof** (*induction A rule*: *disj-induct-mset*)
**case** *1*
**then show** *?case* **by** *simp*
**next**
**case** (*2 n M x*)
**moreover have** *count M x* = *0* **using** *2* **by** (*simp add*: *count-eq-zero-iff*)
**moreover have** $\bigwedge y.\ y \in$ *set-mset M* $\Longrightarrow y \neq x$ **using** *2* **by** *blast*
**ultimately show** *?case* **by** (*simp add*:*algebra-simps*)
**qed**

**lemma** *sum-mset-conv-2*:
  **fixes** $f$ :: $'a \Rightarrow 'b$::{*semiring-1*}
  **assumes** *set-mset* $A \subseteq B$ *finite* $B$
  **shows** *sum-mset* (*image-mset* $f$ $A$) = *sum* ($\lambda x.$ *of-nat* (*count* $A$ $x$) $*$ $f$ $x$) $B$ (**is** *?L* = *?R*)
**proof** −
  **have** *?L* = *sum* ($\lambda x.$ *of-nat* (*count* $A$ $x$) $*$ $f$ $x$) (*set-mset* $A$)
    **unfolding** *sum-mset-conv* **by** *simp*
  **also have** ... = *?R*
    **by** (*intro sum.mono-neutral-left assms*) (*simp-all add*: *iffD2*[*OF count-eq-zero-iff*])
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *count-mset-exp*: *count* $A$ $x$ = *size* (*filter-mset* ($\lambda y.$ $y = x$) $A$)
  **by** (*induction* $A$, *simp*, *simp*)

**lemma** *mset-repl*: *mset* (*replicate* $k$ $x$) = *replicate-mset* $k$ $x$
  **by** (*induction* $k$, *auto*)

**lemma** *count-image-mset-inj*:
  **assumes** *inj* $f$
  **shows** *count* (*image-mset* $f$ $A$) ($f$ $x$) = *count* $A$ $x$
**proof** (*cases* $x \in$ *set-mset* $A$)
  **case** *True*
  **hence** $f$ −' {$f$ $x$} ∩ *set-mset* $A$ = {$x$}
    **using** *assms* **by** (*auto simp add*:*vimage-def inj-def*)
  **then show** *?thesis* **by** (*simp add*:*count-image-mset*)
**next**
  **case** *False*
  **hence** $f$ −' {$f$ $x$} ∩ *set-mset* $A$ = {}
    **using** *assms* **by** (*auto simp add*:*vimage-def inj-def*)
  **thus** *?thesis* **using** *False* **by** (*simp add*:*count-image-mset count-eq-zero-iff*)
**qed**

**lemma** *count-image-mset-0-triv*:
  **assumes** $x \notin$ *range* $f$
  **shows** *count* (*image-mset* $f$ $A$) $x$ = *0*
**proof** −
  **have** $x \notin$ *set-mset* (*image-mset* $f$ $A$)
    **using** *assms* **by** *auto*
  **thus** *?thesis*
    **by** (*meson count-inI*)
**qed**

**lemma** *filter-mset-ex-predicates*:
  **assumes** $\bigwedge x.$ $\neg$ $P$ $x$ $\vee$ $\neg$ $Q$ $x$
  **shows** *filter-mset* $P$ $M$ + *filter-mset* $Q$ $M$ = *filter-mset* ($\lambda x.$ $P$ $x$ $\vee$ $Q$ $x$) $M$
  **using** *assms* **by** (*induction* $M$, *auto*)

**lemma** *sum-count-2*:
  **assumes** *finite* $F$
  **shows** *sum* (*count* $M$) $F$ = *size* (*filter-mset* ($\lambda x.$ $x \in F$) $M$)
  **using** *assms*
**proof** (*induction* $F$ *rule*:*finite-induct*)
  **case** *empty*
  **then show** *?case* **by** *simp*
**next**
  **case** (*insert* $x$ $F$)
  **have** *sum* (*count* $M$) (*insert* $x$ $F$) = *size* ({#$y$ ∈# $M$. $y = x$#} + {#$x$ ∈# $M$. $x \in F$#})

**using** *insert*(*1*,*2*,*3*) **by** (*simp add*:*count-mset-exp*)
**also have** ... = *size* ({#*y* ∈# *M*. *y* = *x* ∨ *y* ∈ *F*#})
**using** *insert*(*2*)
**by** (*intro arg-cong*[**where** *f*=*size*] *filter-mset-ex-predicates*) *simp*
**also have** ... = *size* (*filter-mset* (λ*y*. *y* ∈ *insert x F*) *M*)
**by** *simp*
**finally show** *?case* **by** *simp*
**qed**

**definition** *concat-mset* :: ('*a multiset*) *multiset* ⇒ '*a multiset*
**where** *concat-mset xss* = *fold-mset* (λ*xs ys*. *xs* + *ys*) {#} *xss*

**lemma** *image-concat-mset*:
*image-mset f* (*concat-mset xss*) = *concat-mset* (*image-mset* (*image-mset f*) *xss*)
**unfolding** *concat-mset-def* **by** (*induction xss*, *auto*)

**lemma** *concat-add-mset*:
*concat-mset* (*image-mset* (λ*x*. *f x* + *g x*) *xs*) = *concat-mset* (*image-mset f xs*) + *concat-mset* (*image-mset g xs*)
**unfolding** *concat-mset-def* **by** (*induction xs*) *auto*

**lemma** *concat-add-mset-2*:
*concat-mset* (*xs* + *ys*) = *concat-mset xs* + *concat-mset ys*
**unfolding** *concat-mset-def* **by** (*induction xs*, *auto*)

**lemma** *size-concat-mset*:
*size* (*concat-mset xss*) = *sum-mset* (*image-mset size xss*)
**unfolding** *concat-mset-def* **by** (*induction xss*, *auto*)

**lemma** *filter-concat-mset*:
*filter-mset P* (*concat-mset xss*) = *concat-mset* (*image-mset* (*filter-mset P*) *xss*)
**unfolding** *concat-mset-def* **by** (*induction xss*, *auto*)

**lemma** *count-concat-mset*:
*count* (*concat-mset xss*) *xs* = *sum-mset* (*image-mset* (λ*x*. *count x xs*) *xss*)
**unfolding** *concat-mset-def* **by** (*induction xss*, *auto*)

**lemma** *set-mset-concat-mset*:
*set-mset* (*concat-mset xss*) = ⋃ (*set-mset* ‘ (*set-mset xss*))
**unfolding** *concat-mset-def* **by** (*induction xss*, *auto*)

**lemma** *concat-mset-empty*: *concat-mset* {#} = {#}
**unfolding** *concat-mset-def* **by** *simp*

**lemma** *concat-mset-single*: *concat-mset* {#*x*#} = *x*
**unfolding** *concat-mset-def* **by** *simp*

**lemma** *concat-disjoint-union-mset*:
**assumes** *finite I*
**assumes** ⋀*i*. *i* ∈ *I* ⟹ *finite* (*A i*)
**assumes** ⋀*i j*. *i* ∈ *I* ⟹ *j* ∈ *I* ⟹ *i* ≠ *j* ⟹ *A i* ∩ *A j* = {}
**shows** *mset-set* (⋃ (*A* ‘ *I*)) = *concat-mset* (*image-mset* (*mset-set* ∘ *A*) (*mset-set I*))
**using** *assms*
**proof** (*induction I rule*:*finite-induct*)
**case** *empty*
**then show** *?case* **by** (*simp add*:*concat-mset-empty*)
**next**
**case** (*insert x F*)

**have** *mset-set* ($\bigcup$ (*A* ' *insert x F*)) = *mset-set* (*A x* ∪ ($\bigcup$ (*A* ' *F*)))
  **by** *simp*
**also have** ... = *mset-set* (*A x*) + *mset-set* ($\bigcup$ (*A* ' *F*))
  **using** *insert* **by** (*intro mset-set-Union*) *auto*
**also have** ... = *mset-set* (*A x*) + *concat-mset* (*image-mset* (*mset-set* ∘ *A*) (*mset-set F*))
  **using** *insert* **by** (*intro arg-cong2*[**where** *f*=(+)] *insert*(*3*)) *auto*
**also have** ... = *concat-mset* (*image-mset* (*mset-set* ∘ *A*) ({#*x*#} + *mset-set F*))
  **by** (*simp add*:*concat-mset-def*)
**also have** ... = *concat-mset* (*image-mset* (*mset-set* ∘ *A*) (*mset-set* (*insert x F*)))
  **using** *insert* **by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*]) *auto*
**finally show** *?case* **by** *blast*
**qed**

**lemma** *size-filter-mset-conv*:
  *size* (*filter-mset f A*) = *sum-mset* (*image-mset* (λ*x*. *of-bool* (*f x*) :: *nat*) *A*)
  **by** (*induction A*, *auto*)

**lemma** *filter-mset-const*: *filter-mset* (λ-. *c*) *xs* = (*if c then xs else* {#})
  **by** *simp*

**lemma** *repeat-image-concat-mset*:
  *repeat-mset n* (*image-mset f A*) = *concat-mset* (*image-mset* (λ*x*. *replicate-mset n* (*f x*)) *A*)
  **unfolding** *concat-mset-def* **by** (*induction A*, *auto*)

**lemma** *mset-prod-eq*:
  **assumes** *finite A finite B*
  **shows**
    *mset-set* (*A* × *B*) = *concat-mset* {# {# (*x*,*y*). *y* ∈# *mset-set B* #} .*x* ∈# *mset-set A* #}
  **using** *assms*(*1*)
**proof** (*induction rule*:*finite-induct*)
  **case** *empty*
  **then show** *?case* **unfolding** *concat-mset-def* **by** *simp*
**next**
  **case** (*insert x F*)
  **have** *mset-set* (*insert x F* × *B*) = *mset-set* (*F* × *B* ∪ (λ*y*. (*x*,*y*)) ' *B*)
    **by** (*intro arg-cong*[**where** *f*=*mset-set*]) *auto*
  **also have** ... = *mset-set* (*F* × *B*) + *mset-set* ((λ*y*. (*x*,*y*)) ' *B*)
    **using** *insert*(*1*,*2*) *assms*(*2*) **by** (*intro mset-set-Union finite-cartesian-product*) *auto*
  **also have** ... = *mset-set* (*F* × *B*) + {# (*x*,*y*). *y* ∈# *mset-set B* #}
    **by** (*intro arg-cong2*[**where** *f*=(+)] *image-mset-mset-set*[*symmetric*] *inj-onI*) *auto*
  **also have** ... = *concat-mset* {#*image-mset* (*Pair x*) (*mset-set B*). *x* ∈# {#*x*#} + (*mset-set F*)#}
    **unfolding** *insert image-mset-union concat-add-mset-2* **by** (*simp add*:*concat-mset-single*)
  **also have** ... = *concat-mset* {#*image-mset* (*Pair x*) (*mset-set B*). *x* ∈# *mset-set* (*insert x F*)#}
    **using** *insert*(*1*,*2*) **by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*]) *auto*
  **finally show** *?case* **by** *simp*
**qed**

**lemma** *sum-mset-repeat*:
  **fixes** *f* :: *'a* ⇒ *'b* :: {*comm-monoid-add*,*semiring-1*}
  **shows** *sum-mset* (*image-mset f* (*repeat-mset n A*)) = *of-nat n* ∗ *sum-mset* (*image-mset f A*)
  **by** (*induction n*, *auto simp add*:*sum-mset.distrib algebra-simps*)

**unbundle** *no intro-cong-syntax*

**end**

## 3 Definitions

This section introduces regular graphs as a sublocale in the graph theory developed by Lars Noschinski [10] and introduces various expansion coefficients.

**theory** *Expander-Graphs-Definition*
  **imports**
    *Graph-Theory.Digraph-Isomorphism*
    *HOL−Analysis.L2-Norm*
    *Extra-Congruence-Method*
    *Expander-Graphs-Multiset-Extras*
    *Jordan-Normal-Form.Conjugate*
    *Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities*
**begin**

**unbundle** *intro-cong-syntax*

**definition** *arcs-betw* **where** *arcs-betw G u v* = $\{a.\ a \in$ *arcs G* $\wedge$ *head G a* $= v \wedge$ *tail G a* $= u\}$

The following is a stronger notion than the notion of symmetry defined in *Graph-Theory.Digraph*, it requires that the number of edges from $v$ to $w$ must be equal to the number of edges from $w$ to $v$ for any pair of vertices $v\ w \in$ *verts G*.

**definition** *symmetric-multi-graph* **where** *symmetric-multi-graph G* =
  (*fin-digraph G* $\wedge$ ($\forall v\ w.\ \{v, w\} \subseteq$ *verts G* $\longrightarrow$ *card* (*arcs-betw G w v*) = *card* (*arcs-betw G v w*)))

**lemma** *symmetric-multi-graphI*:
  **assumes** *fin-digraph G*
  **assumes** *bij-betw f* (*arcs G*) (*arcs G*)
  **assumes** $\bigwedge e.\ e \in$ *arcs G* $\Longrightarrow$ *head G* (*f e*) = *tail G e* $\wedge$ *tail G* (*f e*) = *head G e*
  **shows** *symmetric-multi-graph G*
**proof** −
  **have** *card* (*arcs-betw G w v*) = *card* (*arcs-betw G v w*)
    (**is** *?L* = *?R*) **if** $v \in$ *verts G* $w \in$ *verts G* **for** *v w*
  **proof** −
    **have** *a*:*f x* $\in$ *arcs G* **if** $x \in$ *arcs G* **for** *x*
      **using** *assms*(*2*) *that* **unfolding** *bij-betw-def* **by** *auto*
    **have** *b*:$\exists y.\ y \in$ *arcs G* $\wedge$ *f y* = *x* **if** $x \in$ *arcs G* **for** *x*
      **using** *bij-betw-imp-surj-on*[*OF assms*(*2*)] *that* **by** *force*

    **have** *inj-on f* (*arcs G*)
      **using** *assms*(*2*) **unfolding** *bij-betw-def* **by** *simp*
    **hence** *inj-on f* $\{e \in$ *arcs G*. *head G e* $= v \wedge$ *tail G e* $= w\}$
      **by** (*rule inj-on-subset*, *auto*)
    **hence** *?L* = *card* (*f* ‘ $\{e \in$ *arcs G*. *head G e* $= v \wedge$ *tail G e* $= w\}$)
      **unfolding** *arcs-betw-def*
      **by** (*intro card-image*[*symmetric*])
    **also have** ... = *?R*
      **unfolding** *arcs-betw-def* **using** *a b assms*(*3*)
      **by** (*intro arg-cong*[**where** *f*=*card*] *order-antisym image-subsetI subsetI*) *fastforce*+
    **finally show** *?thesis* **by** *simp*
  **qed**
  **thus** *?thesis*
    **using** *assms*(*1*) **unfolding** *symmetric-multi-graph-def* **by** *simp*
**qed**

**lemma** *symmetric-multi-graphD2*:
  **assumes** *symmetric-multi-graph G*

```
  shows fin-digraph G
  using assms unfolding symmetric-multi-graph-def by simp

lemma symmetric-multi-graphD:
  assumes symmetric-multi-graph G
  shows card {e ∈ arcs G. head G e=v ∧ tail G e=w} = card {e ∈ arcs G. head G e=w ∧ tail
G e=v}
    (is card ?L = card ?R)
proof (cases v ∈ verts G ∧ w ∈ verts G)
  case True
  then show ?thesis
  using assms unfolding symmetric-multi-graph-def arcs-betw-def by simp
next
  case False
  interpret fin-digraph G
    using symmetric-multi-graphD2[OF assms(1)] by simp
  have 0:?L = {} ?R = {}
    using False wellformed by auto
  show ?thesis unfolding 0 by simp
qed

lemma symmetric-multi-graphD3:
  assumes symmetric-multi-graph G
  shows
    card {e∈arcs G. tail G e=v ∧ head G e=w}=card {e∈arcs G. tail G e=w∧head G e=v}
  using symmetric-multi-graphD[OF assms] by (simp add:conj.commute)

lemma symmetric-multi-graphD4:
  assumes symmetric-multi-graph G
  shows card (arcs-betw G v w) = card (arcs-betw G w v)
  using symmetric-multi-graphD[OF assms] unfolding arcs-betw-def by simp

lemma symmetric-degree-eq:
  assumes symmetric-multi-graph G
  assumes v ∈ verts G
  shows out-degree G v = in-degree G v (is ?L = ?R)
proof -
  interpret fin-digraph G
    using assms(1) symmetric-multi-graph-def by auto

  have ?L = card {e ∈ arcs G. tail G e = v}
    unfolding out-degree-def out-arcs-def by simp
  also have ... = card (⋃w ∈ verts G. {e ∈ arcs G. head G e = w ∧ tail G e = v})
    by (intro arg-cong[where f=card]) (auto simp add:set-eq-iff)
  also have ... = (∑w ∈ verts G. card {e ∈ arcs G. head G e = w ∧ tail G e = v})
    by (intro card-UN-disjoint) auto
  also have ... = (∑w ∈ verts G. card {e ∈ arcs G. head G e = v ∧ tail G e = w})
    by (intro sum.cong refl symmetric-multi-graphD assms)
  also have ... = card (⋃w ∈ verts G. {e ∈ arcs G. head G e = v ∧ tail G e = w})
    by (intro card-UN-disjoint[symmetric]) auto
  also have ... = card (in-arcs G v)
    by (intro arg-cong[where f=card]) (auto simp add:set-eq-iff)
  also have ... = ?R
    unfolding in-degree-def by simp
  finally show ?thesis by simp
qed

definition edges where edges G = image-mset (arc-to-ends G) (mset-set (arcs G))
```

**lemma** (**in** *fin-digraph*) *count-edges*:
*count* (*edges G*) (*u*,*v*) = *card* (*arcs-betw G u v*) (**is** *?L* = *?R*)
**proof** −
**have** *?L* = *card* {*x* ∈ *arcs G*. *arc-to-ends G x* = (*u*, *v*)}
**unfolding** *edges-def count-mset-exp image-mset-filter-mset-swap*[*symmetric*] **by** *simp*
**also have** ... = *?R*
**unfolding** *arcs-betw-def arc-to-ends-def*
**by** (*intro arg-cong*[**where** *f*=*card*]) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** (**in** *fin-digraph*) *count-edges-sym*:
**assumes** *symmetric-multi-graph G*
**shows** *count* (*edges G*) (*v*, *w*) = *count* (*edges G*) (*w*, *v*)
**unfolding** *count-edges* **using** *symmetric-multi-graphD4*[*OF assms*] **by** *simp*

**lemma** (**in** *fin-digraph*) *edges-sym*:
**assumes** *symmetric-multi-graph G*
**shows** {# (*y*,*x*). (*x*,*y*) ∈# (*edges G*) #} = *edges G*
**proof** −
**have** *count* {#(*y*, *x*). (*x*, *y*) ∈# *edges G*#} *x* = *count* (*edges G*) *x* (**is** *?L* = *?R*) **for** *x*
**proof** −
**have** *?L* = *count* (*edges G*) (*snd x*, *fst x*)
**unfolding** *count-mset-exp*
**by** (*simp add*:*image-mset-filter-mset-swap*[*symmetric*] *case-prod-beta prod-eq-iff ac-simps*)
**also have** ... = *count* (*edges G*) (*fst x*, *snd x*)
**unfolding** *count-edges-sym*[*OF assms*] **by** *simp*
**also have** ... = *count* (*edges G*) *x* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**thus** *?thesis*
**by** (*intro multiset-eqI*) *simp*
**qed**

**definition** *vertices-from G v* = {# *snd e* | *e* ∈# *edges G*. *fst e* = *v* #}
**definition** *vertices-to G v* = {# *fst e* | *e* ∈# *edges G*. *snd e* = *v* #}

**context** *fin-digraph*
**begin**

**lemma** *edge-set*:
**assumes** *x* ∈# *edges G*
**shows** *fst x* ∈ *verts G snd x* ∈ *verts G*
**using** *assms* **unfolding** *edges-def arc-to-ends-def* **by** *auto*

**lemma** *verts-from-alt*:
*vertices-from G v* = *image-mset* (*head G*) (*mset-set* (*out-arcs G v*))
**proof** −
**have** {#*x* ∈# *mset-set* (*arcs G*). *tail G x* = *v*#} = *mset-set* {*a* ∈ *arcs G*. *tail G a* = *v*}
**by** (*intro filter-mset-mset-set*) *simp*
**thus** *?thesis*
**unfolding** *vertices-from-def out-arcs-def edges-def arc-to-ends-def*
**by** (*simp add*:*image-mset.compositionality image-mset-filter-mset-swap*[*symmetric*] *comp-def*)
**qed**

**lemma** *verts-to-alt*:

*vertices-to G v = image-mset (tail G) (mset-set (in-arcs G v))*
**proof** −
**have** {#*x* ∈# *mset-set* (*arcs G*). *head G x* = *v*#} = *mset-set* {*a* ∈ *arcs G*. *head G a* = *v*}
**by** (*intro filter-mset-mset-set*) *simp*
**thus** *?thesis*
**unfolding** *vertices-to-def in-arcs-def edges-def arc-to-ends-def*
**by** (*simp add:image-mset.compositionality image-mset-filter-mset-swap*[*symmetric*] *comp-def*)
**qed**

**lemma** *set-mset-vertices-from*:
*set-mset* (*vertices-from G x*) ⊆ *verts G*
**unfolding** *vertices-from-def* **using** *edge-set* **by** *auto*

**lemma** *set-mset-vertices-to*:
*set-mset* (*vertices-to G x*) ⊆ *verts G*
**unfolding** *vertices-to-def* **using** *edge-set* **by** *auto*

**end**

A symmetric multigraph is regular if every node has the same degree. This is the context in which the expansion conditions are introduced.

**locale** *regular-graph* = *fin-digraph* +
**assumes** *sym*: *symmetric-multi-graph G*
**assumes** *verts-non-empty*: *verts G* ≠ {}
**assumes** *arcs-non-empty*: *arcs G* ≠ {}
**assumes** *reg'*: ⋀*v w*. *v* ∈ *verts G* ⟹ *w* ∈ *verts G* ⟹ *out-degree G v* = *out-degree G w*
**begin**

**definition** *d* **where** *d* = *out-degree G* (*SOME v*. *v* ∈ *verts G*)

**lemmas** *count-sym* = *count-edges-sym*[*OF sym*]

**lemma** *reg*:
**assumes** *v* ∈ *verts G*
**shows** *out-degree G v* = *d in-degree G v* = *d*
**proof** −
**define** *w* **where** *w* = (*SOME v*. *v* ∈ *verts G*)
**have** *w* ∈ *verts G*
**unfolding** *w-def* **using** *assms*(*1*) **by** (*rule someI*)
**hence** *out-degree G v* = *out-degree G w*
**by** (*intro reg' assms*(*1*))
**also have** ... = *d*
**unfolding** *d-def w-def* **by** *simp*
**finally show** *a*:*out-degree G v* = *d* **by** *simp*

**show** *in-degree G v* = *d*
**using** *a symmetric-degree-eq*[*OF sym assms*(*1*)] **by** *simp*
**qed**

**definition** *n* **where** *n* = *card* (*verts G*)

**lemma** *n-gt-0*: *n* > *0*
**unfolding** *n-def* **using** *verts-non-empty* **by** *auto*

**lemma** *d-gt-0*: *d* > *0*
**proof** −
**obtain** *a* **where** *a*:*a* ∈ *arcs G*
**using** *arcs-non-empty* **by** *auto*

**hence** *a* ∈ *in-arcs G* (*head G a*)
  **unfolding** *in-arcs-def* **by** *simp*
**hence** *0* < *in-degree G* (*head G a*)
  **unfolding** *in-degree-def card-gt-0-iff* **by** *blast*
**also have** ... = *d*
  **using** *a* **by** (*intro reg*) *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**definition** *g-inner* :: ('*a* ⇒ ('*c* :: *conjugatable-field*)) ⇒ ('*a* ⇒ '*c*) ⇒ '*c*
  **where** *g-inner f g* = ($\sum$ *x* ∈ *verts G*. (*f x*) ∗ *conjugate* (*g x*))

**lemma** *conjugate-divide*[*simp*]:
  **fixes** *x y* :: '*c* :: *conjugatable-field*
  **shows** *conjugate* (*x* / *y*) = *conjugate x* / *conjugate y*
**proof** (*cases y* = *0*)
  **case** *True*
  **then show** *?thesis* **by** *simp*
**next**
  **case** *False*
  **have** *conjugate* (*x*/*y*) ∗ *conjugate y* = *conjugate x*
    **using** *False* **by** (*simp add*:*conjugate-dist-mul*[*symmetric*])
  **thus** *?thesis*
    **by** (*simp add*:*divide-simps*)
**qed**

**lemma** *g-inner-simps*:
  *g-inner* (λ*x*. *0*) *g* = *0*
  *g-inner f* (λ*x*. *0*) = *0*
  *g-inner* (λ*x*. *c* ∗ *f x*) *g* = *c* ∗ *g-inner f g*
  *g-inner f* (λ*x*. *c* ∗ *g x*) = *conjugate c* ∗ *g-inner f g*
  *g-inner* (λ*x*. *f x* − *g x*) *h* = *g-inner f h* − *g-inner g h*
  *g-inner* (λ*x*. *f x* + *g x*) *h* = *g-inner f h* + *g-inner g h*
  *g-inner f* (λ*x*. *g x* + *h x*) = *g-inner f g* + *g-inner f h*
  *g-inner f* (λ*x*. *g x* / *c*) = *g-inner f g* / *conjugate c*
  *g-inner* (λ*x*. *f x* / *c*) *g* = *g-inner f g* / *c*
  **unfolding** *g-inner-def*
  **by** (*auto simp add*:*sum.distrib algebra-simps sum-distrib-left sum-subtractf sum-divide-distrib*
    *conjugate-dist-mul conjugate-dist-add*)

**definition** *g-norm f* = *sqrt* (*g-inner f f*)

**lemma** *g-norm-eq*: *g-norm f* = *L2-set f* (*verts G*)
  **unfolding** *g-norm-def g-inner-def L2-set-def*
  **by** (*intro arg-cong*[**where** *f*=*sqrt*] *sum.cong refl*) (*simp add*:*power2-eq-square*)

**lemma** *g-inner-cauchy-schwartz*:
  **fixes** *f g* :: '*a* ⇒ *real*
  **shows** |*g-inner f g*| ≤ *g-norm f* ∗ *g-norm g*
**proof** −
  **have** |*g-inner f g*| ≤ ($\sum$ *v* ∈ *verts G*. |*f v* ∗ *g v*|)
    **unfolding** *g-inner-def conjugate-real-def* **by** (*intro sum-abs*)
  **also have** ... ≤ *g-norm f* ∗ *g-norm g*
    **unfolding** *g-norm-eq abs-mult* **by** (*intro L2-set-mult-ineq*)
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *g-inner-cong*:

**assumes** ⋀*x*. *x* ∈ *verts G* ⟹ *f1 x* = *f2 x*
**assumes** ⋀*x*. *x* ∈ *verts G* ⟹ *g1 x* = *g2 x*
**shows** *g-inner f1 g1* = *g-inner f2 g2*
**unfolding** *g-inner-def* **using** *assms*
**by** (*intro sum.cong refl*) *auto*

**lemma** *g-norm-cong*:
**assumes** ⋀*x*. *x* ∈ *verts G* ⟹ *f x* = *g x*
**shows** *g-norm f* = *g-norm g*
**unfolding** *g-norm-def*
**by** (*intro arg-cong*[**where** *f*=*sqrt*] *g-inner-cong assms*)

**lemma** *g-norm-nonneg*: *g-norm f* ≥ *0*
**unfolding** *g-norm-def g-inner-def*
**by** (*intro real-sqrt-ge-zero sum-nonneg*) *auto*

**lemma** *g-norm-sq*:
*g-norm f^2* = *g-inner f f*
**using** *g-norm-nonneg g-norm-def* **by** *simp*

**definition** *g-step* :: (′*a* ⇒ *real*) ⇒ (′*a* ⇒ *real*)
**where** *g-step f v* = (∑ *x* ∈ *in-arcs G v*. *f* (*tail G x*) / *real d*)

**lemma** *g-step-simps*:
*g-step* (λ*x*. *f x* + *g x*) *y* = *g-step f y* + *g-step g y*
*g-step* (λ*x*. *f x* / *c*) *y* = *g-step f y* / *c*
**unfolding** *g-step-def sum-divide-distrib*[*symmetric*] **using** *finite-in-arcs d-gt-0*
**by** (*auto intro*:*sum.cong simp add*:*sum.distrib field-simps sum-distrib-left sum-subtractf*)

**lemma** *g-inner-step-eq*:
*g-inner f* (*g-step f*) = (∑ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*)) / *d* (**is** *?L* = *?R*)
**proof** −
**have** *?L* = (∑ *v*∈*verts G*. *f v* ∗ (∑ *a*∈*in-arcs G v*. *f* (*tail G a*) / *d*))
**unfolding** *g-inner-def g-step-def* **by** *simp*
**also have** ... = (∑ *v*∈*verts G*. (∑ *a*∈*in-arcs G v*. *f v* ∗ *f* (*tail G a*) / *d*))
**by** (*subst sum-distrib-left*) *simp*
**also have** ... = (∑ *v*∈*verts G*. (∑ *a*∈*in-arcs G v*. *f* (*head G a*) ∗ *f* (*tail G a*) / *d*))
**unfolding** *in-arcs-def* **by** (*intro sum.cong refl*) *simp*
**also have** ... = (∑ *a* ∈ (⋃ (*in-arcs G* ‘ *verts G*)). *f* (*head G a*) ∗ *f* (*tail G a*) / *d*)
**using** *finite-verts* **by** (*intro sum.UNION-disjoint*[*symmetric*] *ballI*)
(*auto simp add*:*in-arcs-def*)
**also have** ... = (∑ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*) / *d*)
**unfolding** *in-arcs-def* **using** *wellformed* **by** (*intro sum.cong*) *auto*
**also have** ... = *?R*
**by** (*intro sum-divide-distrib*[*symmetric*])
**finally show** *?thesis* **by** *simp*
**qed**

**definition** Λ-*test*
**where** Λ-*test* = {*f*. *g-norm f^2* ≠ *0* ∧ *g-inner f* (λ-. *1*) = *0*}

**lemma** Λ-*test-ne*:
**assumes** *n* > *1*
**shows** Λ-*test* ≠ {}
**proof** −
**obtain** *v* **where** *v-def*: *v* ∈ *verts G* **using** *verts-non-empty* **by** *auto*
**have** *False* **if** ⋀*w*. *w* ∈ *verts G* ⟹ *w* = *v*
**proof** −

**have** *verts G* = {*v*} **using** *that v-def*
**by** (*intro iffD2*[*OF set-eq-iff*] *allI*) *blast*
**thus** *False*
**using** *assms n-def* **by** *simp*
**qed**
**then obtain** *w* **where** *w-def*: *w* ∈ *verts G v* ≠ *w*
**by** *auto*
**define** *f* **where** *f x*= (*if x* = *v then 1 else* (*if x* = *w then* (−*1*) *else* (*0*::*real*))) **for** *x*

**have** *g-norm f*ˆ*2* = ($\sum$ *x*∈*verts G*. (*if x* = *v then 1 else if x* = *w then* − *1 else 0*)$^2$)
**unfolding** *g-norm-sq g-inner-def conjugate-real-def power2-eq-square*[*symmetric*]
**by** (*simp add*:*f-def*)
**also have** ... = ($\sum$ *x* ∈ {*v*,*w*}. (*if x* = *v then 1 else if x* = *w then* −*1 else 0*)$^2$)
**using** *v-def*(*1*) *w-def*(*1*) **by** (*intro sum.mono-neutral-cong refl*) *auto*
**also have** ... = ($\sum$ *x* ∈ {*v*,*w*}. (*if x* = *v then 1 else* − *1*)$^2$)
**by** (*intro sum.cong*) *auto*
**also have** ... = *2*
**using** *w-def*(*2*) **by** (*simp add*:*if-distrib if-distribR sum.If-cases*)
**finally have** *g-norm f*ˆ*2* = *2* **by** *simp*
**hence** *g-norm f* ≠ *0* **by** *auto*

**moreover have** *g-inner f* (λ-.*1*) = *0*
**unfolding** *g-inner-def f-def* **using** *v-def w-def* **by** (*simp add*:*sum.If-cases*)
**ultimately have** *f* ∈ Λ-*test*
**unfolding** Λ-*test-def* **by** *simp*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** Λ-*test-empty*:
**assumes** *n* = *1*
**shows** Λ-*test* = {}
**proof** −
**obtain** *v* **where** *v-def*: *verts G* = {*v*}
**using** *assms card-1-singletonE* **unfolding** *n-def* **by** *auto*
**have** *False* **if** *f* ∈ Λ-*test* **for** *f*
**proof** −
**have** *0* = (*g-inner f* (λ-.*1*))ˆ*2*
**using** *that* Λ-*test-def* **by** *simp*
**also have** ... = (*f v*)ˆ*2*
**unfolding** *g-inner-def v-def* **by** *simp*
**also have** ... = *g-norm f*ˆ*2*
**unfolding** *g-norm-sq g-inner-def v-def*
**by** (*simp add*:*power2-eq-square*)
**also have** ... ≠ *0*
**using** *that* Λ-*test-def* **by** *simp*
**finally show** *False* **by** *simp*
**qed**
**thus** *?thesis* **by** *auto*
**qed**

The following are variational definitions for the maxiumum of the spectrum (resp. maximum modulus of the spectrum) of the stochastic matrix (excluding the Perron eigenvalue 1). Note that both values can still obtain the value one 1 (if the multiplicity of the eigenvalue 1 is larger than 1 in the stochastic matrix, or in the modulus case if −1 is an eigenvalue).

The definition relies on the supremum of the Rayleigh-Quotient for vectors orthogonal to the stationary distribution). In Section 6, the equivalence of this value with the algebraic definition will be shown. The definition here has the advantage that it is (obviously) independent of the matrix representation (ordering of the vertices) used.

**definition** $\Lambda_2$ :: *real*
  **where** $\Lambda_2 = (\mathit{if}\ n > 1\ \mathit{then}\ (\mathit{SUP}\ f \in \Lambda\text{-}\mathit{test}.\ \mathit{g\text{-}inner}\ f\ (\mathit{g\text{-}step}\ f)/\mathit{g\text{-}inner}\ f\ f)\ \mathit{else}\ 0)$

**definition** $\Lambda_a$ :: *real*
  **where** $\Lambda_a = (\mathit{if}\ n > 1\ \mathit{then}\ (\mathit{SUP}\ f \in \Lambda\text{-}\mathit{test}.\ |\mathit{g\text{-}inner}\ f\ (\mathit{g\text{-}step}\ f)|/\mathit{g\text{-}inner}\ f\ f)\ \mathit{else}\ 0)$

**lemma** *sum-arcs-tail*:
  **fixes** $f$ :: $'a \Rightarrow ('c$ :: *semiring-1*)
  **shows** $(\sum a \in \mathit{arcs}\ G.\ f\ (\mathit{tail}\ G\ a)) = \mathit{of\text{-}nat}\ d * (\sum v \in \mathit{verts}\ G.\ f\ v)$ (**is** *?L* = *?R*)
**proof** −
  **have** $?L = (\sum a{\in}(\bigcup(\mathit{out\text{-}arcs}\ G\ `\ \mathit{verts}\ G)).\ f\ (\mathit{tail}\ G\ a))$
    **by** (*intro sum.cong*) *auto*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ (\sum a \in \mathit{out\text{-}arcs}\ G\ v.\ f\ (\mathit{tail}\ G\ a)))$
    **by** (*intro sum.UNION-disjoint*) *auto*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ \mathit{of\text{-}nat}\ (\mathit{out\text{-}degree}\ G\ v) * f\ v)$
    **unfolding** *out-degree-def* **by** *simp*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ \mathit{of\text{-}nat}\ d * f\ v)$
    **by** (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *arg-cong*[**where** *f*=*of-nat*] *reg*) *auto*
  **also have** ... = *?R* **by** (*simp add:sum-distrib-left*)
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *sum-arcs-head*:
  **fixes** $f$ :: $'a \Rightarrow ('c$ :: *semiring-1*)
  **shows** $(\sum a \in \mathit{arcs}\ G.\ f\ (\mathit{head}\ G\ a)) = \mathit{of\text{-}nat}\ d * (\sum v \in \mathit{verts}\ G.\ f\ v)$ (**is** *?L* = *?R*)
**proof** −
  **have** $?L = (\sum a{\in}(\bigcup(\mathit{in\text{-}arcs}\ G\ `\ \mathit{verts}\ G)).\ f\ (\mathit{head}\ G\ a))$
    **by** (*intro sum.cong*) *auto*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ (\sum a \in \mathit{in\text{-}arcs}\ G\ v.\ f\ (\mathit{head}\ G\ a)))$
    **by** (*intro sum.UNION-disjoint*) *auto*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ \mathit{of\text{-}nat}\ (\mathit{in\text{-}degree}\ G\ v) * f\ v)$
    **unfolding** *in-degree-def* **by** *simp*
  **also have** ... = $(\sum v \in \mathit{verts}\ G.\ \mathit{of\text{-}nat}\ d * f\ v)$
    **by** (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *arg-cong*[**where** *f*=*of-nat*] *reg*) *auto*
  **also have** ... = *?R* **by** (*simp add:sum-distrib-left*)
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *bdd-above-aux*:
  $|\sum a{\in}\mathit{arcs}\ G.\ f(\mathit{head}\ G\ a){*}f(\mathit{tail}\ G\ a)| \le d{*}\ \mathit{g\text{-}norm}\ f\hat{\ }2$ (**is** *?L* ≤ *?R*)
**proof** −
  **have** $(\sum a{\in}\mathit{arcs}\ G.\ f\ (\mathit{head}\ G\ a)\hat{\ }2) = d * \mathit{g\text{-}norm}\ f\hat{\ }2$
    **unfolding** *sum-arcs-head*[**where** $f{=}\lambda x.\ f\ x\hat{\ }2$] *g-norm-sq g-inner-def*
    **by** (*simp add:power2-eq-square*)
  **hence** $0{:}\mathit{L2\text{-}set}\ (\lambda a.\ f\ (\mathit{head}\ G\ a))\ (\mathit{arcs}\ G) \le \mathit{sqrt}\ (d * \mathit{g\text{-}norm}\ f\hat{\ }2)$
    **using** *g-norm-nonneg* **unfolding** *L2-set-def* **by** *simp*

  **have** $(\sum a{\in}\mathit{arcs}\ G.\ f\ (\mathit{tail}\ G\ a)\hat{\ }2) = d * \mathit{g\text{-}norm}\ f\hat{\ }2$
   **unfolding** *sum-arcs-tail*[**where** $f{=}\lambda x.\ f\ x\hat{\ }2$] *sum-distrib-left*[*symmetric*] *g-norm-sq g-inner-def*
    **by** (*simp add:power2-eq-square*)
  **hence** $1{:}\mathit{L2\text{-}set}\ (\lambda a.\ f\ (\mathit{tail}\ G\ a))\ (\mathit{arcs}\ G) \le \mathit{sqrt}\ (d * \mathit{g\text{-}norm}\ f\hat{\ }2)$
    **unfolding** *L2-set-def* **by** *simp*

  **have** $?L \le (\sum a \in \mathit{arcs}\ G.\ |f\ (\mathit{head}\ G\ a)| * |f(\mathit{tail}\ G\ a)|)$
    **unfolding** *abs-mult*[*symmetric*] **by** (*intro divide-right-mono sum-abs*)

**also have** ... ≤ (*L2-set* (λ*a*. *f* (*head G a*)) (*arcs G*) ∗ *L2-set* (λ*a*. *f* (*tail G a*)) (*arcs G*))
  **by** (*intro L2-set-mult-ineq*)
**also have** ... ≤ (*sqrt* (*d* ∗ *g-norm f*^*2*) ∗ *sqrt* (*d* ∗ *g-norm f*^*2*))
  **by** (*intro mult-mono 0 1*) *auto*
**also have** ... = *d* ∗ *g-norm f*^*2*
  **using** *d-gt-0 g-norm-nonneg* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *bdd-above-aux-2*:
  **assumes** *f* ∈ Λ-*test*
  **shows** |*g-inner f* (*g-step f*)| / *g-inner f f* ≤ *1*
**proof** −
  **have** *0*:*g-inner f f* > *0*
    **using** *assms* **unfolding** Λ-*test-def g-norm-sq*[*symmetric*] **by** *auto*

  **have** |*g-inner f* (*g-step f*)| = |∑ *a*∈*arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*)| / *real d*
    **unfolding** *g-inner-step-eq* **by** *simp*
  **also have** ... ≤ *d* ∗ *g-norm f*^*2* / *d*
    **by** (*intro divide-right-mono bdd-above-aux assms*) *auto*
  **also have** ... = *g-inner f f*
    **using** *d-gt-0* **unfolding** *g-norm-sq* **by** *simp*
  **finally have** |*g-inner f* (*g-step f*)| ≤ *g-inner f f*
    **by** *simp*

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

**lemma** *bdd-above-aux-3*:
  **assumes** *f* ∈ Λ-*test*
  **shows** *g-inner f* (*g-step f*) / *g-inner f f* ≤ *1* (**is** *?L* ≤ *?R*)
**proof** −
  **have** *?L* ≤ |*g-inner f* (*g-step f*)| / *g-inner f f*
    **unfolding** *g-norm-sq*[*symmetric*]
    **by** (*intro divide-right-mono*) *auto*
  **also have** ... ≤ *1*
    **using** *bdd-above-aux-2*[*OF assms*] **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *bdd-above*-Λ: *bdd-above* ((λ*f*. |*g-inner f* (*g-step f*)| / *g-inner f f*) ‘ Λ-*test*)
  **using** *bdd-above-aux-2*
  **by** (*intro bdd-aboveI*[**where** *M*=*1*]) *auto*

**lemma** *bdd-above*-$\Lambda_2$: *bdd-above* ((λ*f*. *g-inner f* (*g-step f*) / *g-inner f f*) ‘ Λ-*test*)
  **using** *bdd-above-aux-3*
  **by** (*intro bdd-aboveI*[**where** *M*=*1*]) *auto*

**lemma** Λ-*le-1*: $\Lambda_a$ ≤ *1*
**proof** (*cases n* > *1*)
  **case** *True*
  **have** (*SUP f*∈Λ-*test*. |*g-inner f* (*g-step f*)| / *g-inner f f*) ≤ *1*
    **using** *bdd-above-aux-2* Λ-*test-ne*[*OF True*] **by** (*intro cSup-least*) *auto*
  **thus** $\Lambda_a$ ≤ *1*
    **unfolding** $\Lambda_a$-*def* **using** *True* **by** *simp*
**next**
  **case** *False*

**thus** *?thesis* **unfolding** $\Lambda_a$*-def* **by** *simp*
**qed**

**lemma** $\Lambda_2$*-le-1*: $\Lambda_2 \le 1$
**proof** (*cases* $n > 1$)
**case** *True*
**have** ($SUP\ f{\in}\Lambda\text{-}test.\ g\text{-}inner\ f\ (g\text{-}step\ f)\ /\ g\text{-}inner\ f\ f) \le 1$
**using** *bdd-above-aux-3* $\Lambda$*-test-ne*[*OF True*] **by** (*intro cSup-least*) *auto*
**thus** $\Lambda_2 \le 1$
**unfolding** $\Lambda_2$*-def* **using** *True* **by** *simp*
**next**
**case** *False*
**thus** *?thesis* **unfolding** $\Lambda_2$*-def* **by** *simp*
**qed**

**lemma** $\Lambda$*-ge-0*: $\Lambda_a \ge 0$
**proof** (*cases* $n > 1$)
**case** *True*
**obtain** *f* **where** *f-def*: $f \in \Lambda$*-test*
**using** $\Lambda$*-test-ne*[*OF True*] **by** *auto*
**have** $0 \le |g\text{-}inner\ f\ (g\text{-}step\ f)|\ /\ g\text{-}inner\ f\ f$
**unfolding** *g-norm-sq*[*symmetric*] **by** (*intro divide-nonneg-nonneg*) *auto*
**also have** ... $\le (SUP\ f{\in}\Lambda\text{-}test.\ |g\text{-}inner\ f\ (g\text{-}step\ f)|\ /\ g\text{-}inner\ f\ f)$
**using** *f-def* **by** (*intro cSup-upper bdd-above-*$\Lambda$) *auto*
**finally have** $(SUP\ f{\in}\Lambda\text{-}test.\ |g\text{-}inner\ f\ (g\text{-}step\ f)|\ /\ g\text{-}inner\ f\ f) \ge 0$
**by** *simp*
**thus** *?thesis*
**unfolding** $\Lambda_a$*-def* **using** *True* **by** *simp*
**next**
**case** *False*
**thus** *?thesis* **unfolding** $\Lambda_a$*-def* **by** *simp*
**qed**

**lemma** *os-expanderI*:
**assumes** $n > 1$
**assumes** $\bigwedge f.\ g\text{-}inner\ f\ (\lambda\text{-}.\ 1){=}0 \Longrightarrow g\text{-}inner\ f\ (g\text{-}step\ f) \le C{*}g\text{-}norm\ f\hat{}2$
**shows** $\Lambda_2 \le C$
**proof** −
**have** $g\text{-}inner\ f\ (g\text{-}step\ f)\ /\ g\text{-}inner\ f\ f \le C$ **if** $f \in \Lambda$*-test* **for** *f*
**proof** −
**have** $g\text{-}inner\ f\ (g\text{-}step\ f) \le C{*}g\text{-}inner\ f\ f$
**using** *that* $\Lambda$*-test-def assms*(*2*) **unfolding** *g-norm-sq* **by** *auto*
**moreover have** $g\text{-}inner\ f\ f > 0$
**using** *that* **unfolding** $\Lambda$*-test-def g-norm-sq*[*symmetric*] **by** *auto*
**ultimately show** *?thesis*
**by** (*simp add*:*divide-simps*)
**qed**
**hence** $(SUP\ f{\in}\Lambda\text{-}test.\ g\text{-}inner\ f\ (g\text{-}step\ f)\ /\ g\text{-}inner\ f\ f) \le C$
**using** $\Lambda$*-test-ne*[*OF assms*(*1*)] **by** (*intro cSup-least*) *auto*
**thus** *?thesis*
**unfolding** $\Lambda_2$*-def* **using** *assms* **by** *simp*
**qed**

**lemma** *os-expanderD*:
**assumes** $g\text{-}inner\ f\ (\lambda\text{-}.\ 1) = 0$
**shows** $g\text{-}inner\ f\ (g\text{-}step\ f) \le \Lambda_2 * g\text{-}norm\ f\hat{}2$ (**is** $?L \le ?R$)
**proof** (*cases* $g\text{-}norm\ f \ne 0$)
**case** *True*

**have** *0*:$f \in \Lambda$-*test*
  **unfolding** $\Lambda$-*test-def* **using** *assms True* **by** *auto*

**hence** *1*:$n > 1$
  **using** $\Lambda$-*test-empty n-gt-0* **by** *fastforce*

**have** *g-inner f* (*g-step f*)/ *g-norm f*^*2* = *g-inner f* (*g-step f*)/*g-inner f f*
  **unfolding** *g-norm-sq* **by** *simp*
**also have** ... ≤ (*SUP f*∈$\Lambda$-*test*. *g-inner f* (*g-step f*) / *g-inner f f*)
  **by** (*intro cSup-upper bdd-above*-$\Lambda_2$ *imageI 0*)
**also have** ... = $\Lambda_2$
  **using** *1* **unfolding** $\Lambda_2$-*def* **by** *simp*
**finally have** *g-inner f* (*g-step f*)/ *g-norm f*^*2* ≤ $\Lambda_2$ **by** *simp*
**thus** *?thesis*
  **using** *True* **by** (*simp add*:*divide-simps*)
**next**
**case** *False*
**hence** *g-inner f f* = *0*
  **unfolding** *g-norm-sq*[*symmetric*] **by** *simp*
**hence** *0*:$\bigwedge$*v*. *v* ∈ *verts G* ⟹ *f v* = *0*
  **unfolding** *g-inner-def* **by** (*subst* (*asm*) *sum-nonneg-eq-0-iff*) *auto*
**hence** *?L* = *0*
  **unfolding** *g-step-def g-inner-def* **by** *simp*
**also have** ... ≤ $\Lambda_2$ ∗ *g-norm f*^*2*
  **using** *False* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expander-intro-1*:
  **assumes** *C* ≥ *0*
  **assumes** $\bigwedge$*f*. *g-inner f* (λ-. *1*)=*0* ⟹ |*g-inner f* (*g-step f*)| ≤ *C*∗*g-norm f*^*2*
  **shows** $\Lambda_a$ ≤ *C*
**proof** (*cases n* > *1*)
  **case** *True*
  **have** |*g-inner f* (*g-step f*)| / *g-inner f f* ≤ *C* **if** *f* ∈ $\Lambda$-*test* **for** *f*
  **proof** −
    **have** |*g-inner f* (*g-step f*)| ≤ *C*∗*g-inner f f*
      **using** *that* $\Lambda$-*test-def assms*(*2*) **unfolding** *g-norm-sq* **by** *auto*
    **moreover have** *g-inner f f* > *0*
      **using** *that* **unfolding** $\Lambda$-*test-def g-norm-sq*[*symmetric*] **by** *auto*
    **ultimately show** *?thesis*
      **by** (*simp add*:*divide-simps*)
  **qed**

  **hence** (*SUP f*∈$\Lambda$-*test*. |*g-inner f* (*g-step f*)| / *g-inner f f*) ≤ *C*
    **using** $\Lambda$-*test-ne*[*OF True*] **by** (*intro cSup-least*) *auto*
  **thus** *?thesis* **using** *True* **unfolding** $\Lambda_a$-*def* **by** *auto*
**next**
  **case** *False*
  **then show** *?thesis* **using** *assms* **unfolding** $\Lambda_a$-*def* **by** *simp*
**qed**

**lemma** *expander-intro*:
  **assumes** *C* ≥ *0*
  **assumes** $\bigwedge$*f*. *g-inner f* (λ-. *1*)=*0* ⟹ |$\sum$ *a* ∈ *arcs G*. *f*(*head G a*) ∗ *f*(*tail G a*)| ≤ *C*∗*g-norm*
*f*^*2*
  **shows** $\Lambda_a$ ≤ *C*/*d*

**proof** −
  **have** $|g\text{-}inner\ f\ (g\text{-}step\ f)| \leq C\ /\ real\ d * (g\text{-}norm\ f)^2$ (**is** $?L \leq ?R$)
    **if** $g\text{-}inner\ f\ (\lambda\text{-}.\ 1) = 0$ **for** $f$
  **proof** −
    **have** $?L = |\sum a{\in}arcs\ G.\ f\ (head\ G\ a) * f\ (tail\ G\ a)|\ /\ real\ d$
      **unfolding** *g-inner-step-eq* **by** *simp*
    **also have** ... $\leq C{*}g\text{-}norm\ f\hat{\ }2\ /\ real\ d$
      **by** (*intro divide-right-mono assms*(*2*)[*OF that*]) *auto*
    **also have** ... $= ?R$ **by** *simp*
    **finally show** *?thesis* **by** *simp*
  **qed**
  **thus** *?thesis*
    **by** (*intro expander-intro-1 divide-nonneg-nonneg assms*) *auto*
**qed**

**lemma** *expansionD1*:
  **assumes** $g\text{-}inner\ f\ (\lambda\text{-}.\ 1) = 0$
  **shows** $|g\text{-}inner\ f\ (g\text{-}step\ f)| \leq \Lambda_a * g\text{-}norm\ f\hat{\ }2$ (**is** $?L \leq ?R$)
**proof** (*cases g-norm f* $\neq 0$)
  **case** *True*

  **have** $0{:}f \in \Lambda\text{-}test$
    **unfolding** $\Lambda$*-test-def* **using** *assms True* **by** *auto*

  **hence** $1{:}n > 1$
    **using** $\Lambda$*-test-empty n-gt-0* **by** *fastforce*

  **have** $|g\text{-}inner\ f\ (g\text{-}step\ f)|/\ g\text{-}norm\ f\hat{\ }2 = |g\text{-}inner\ f\ (g\text{-}step\ f)|/g\text{-}inner\ f\ f$
    **unfolding** *g-norm-sq* **by** *simp*
  **also have** ... $\leq (SUP\ f{\in}\Lambda\text{-}test.\ |g\text{-}inner\ f\ (g\text{-}step\ f)|\ /\ g\text{-}inner\ f\ f)$
    **by** (*intro cSup-upper bdd-above-*$\Lambda$ *imageI 0*)
  **also have** ... $= \Lambda_a$
    **using** *1* **unfolding** $\Lambda_a$*-def* **by** *simp*
  **finally have** $|g\text{-}inner\ f\ (g\text{-}step\ f)|/\ g\text{-}norm\ f\hat{\ }2 \leq \Lambda_a$ **by** *simp*
  **thus** *?thesis*
    **using** *True* **by** (*simp add*:*divide-simps*)
**next**
  **case** *False*
  **hence** $g\text{-}inner\ f\ f = 0$
    **unfolding** *g-norm-sq*[*symmetric*] **by** *simp*
  **hence** $0{:}\bigwedge v.\ v \in verts\ G \Longrightarrow f\ v = 0$
    **unfolding** *g-inner-def* **by** (*subst* (*asm*) *sum-nonneg-eq-0-iff*) *auto*
  **hence** $?L = 0$
    **unfolding** *g-step-def g-inner-def* **by** *simp*
  **also have** ... $\leq \Lambda_a * g\text{-}norm\ f\hat{\ }2$
    **using** *False* **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expansionD*:
  **assumes** $g\text{-}inner\ f\ (\lambda\text{-}.\ 1) = 0$
  **shows** $|\sum a \in arcs\ G.\ f\ (head\ G\ a) * f\ (tail\ G\ a)| \leq d * \Lambda_a * g\text{-}norm\ f\hat{\ }2$ (**is** $?L \leq ?R$)
**proof** −
  **have** $?L = |g\text{-}inner\ f\ (g\text{-}step\ f) * d|$
    **unfolding** *g-inner-step-eq* **using** *d-gt-0* **by** *simp*
  **also have** ... $\leq |g\text{-}inner\ f\ (g\text{-}step\ f)| * d$
    **by** (*simp add*:*abs-mult*)
  **also have** ... $\leq (\Lambda_a * g\text{-}norm\ f\hat{\ }2) * d$

**by** (*intro expansionD1 mult-right-mono assms*(*1*)) *auto*
**also have** ... = *?R* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**definition** *edges-betw* **where** *edges-betw S T* = $\{a \in arcs\ G.\ tail\ G\ a \in S \wedge head\ G\ a \in T\}$

This parameter is the edge expansion. It is usually denoted by the symbol $h$ or $h(G)$ in text books. Contrary to the previous definitions it doesn't have a spectral theoretic counter part.

**definition** $\Lambda_e$ **where** $\Lambda_e = (\mathit{if}\ n > 1\ \mathit{then}$
$(MIN\ S \in \{S.\ S \subseteq verts\ G \wedge 2 * card\ S \leq n \wedge S \neq \{\}\}.\ real\ (card\ (edges\text{-}betw\ S\ (-S)))/card\ S)\ \mathit{else}\ 0)$

**lemma** *edge-expansionD*:
**assumes** $S \subseteq verts\ G\ 2 * card\ S \leq n$
**shows** $\Lambda_e * card\ S \leq real\ (card\ (edges\text{-}betw\ S\ (-S)))$
**proof** (*cases* $S \neq \{\}$)
**case** *True*
**moreover have** *finite S*
**using** *finite-subset*[*OF assms*(*1*)] **by** *simp*
**ultimately have** *card S* > *0* **by** *auto*
**hence** *1*: *real* (*card S*) > *0* **by** *simp*
**hence** *2*: *n* > *1* **using** *assms*(*2*) **by** *simp*

**let** $?St = \{S.\ S \subseteq verts\ G \wedge 2 * card\ S \leq n \wedge S \neq \{\}\}$

**have** *0*: *finite ?St*
**by** (*rule finite-subset*[**where** *B*=*Pow* (*verts G*)]) *auto*
**have** $\Lambda_e = (MIN\ S \in ?St.\ real\ (card\ (edges\text{-}betw\ S\ (-S)))/card\ S)$
**using** *2* **unfolding** $\Lambda_e$*-def* **by** *simp*

**also have** ... $\leq real\ (card\ (edges\text{-}betw\ S\ (-S)))\ /\ card\ S$
**using** *assms True* **by** (*intro Min-le finite-imageI imageI*) *auto*
**finally have** $\Lambda_e \leq real\ (card\ (edges\text{-}betw\ S\ (-S)))\ /\ card\ S$ **by** *simp*
**thus** *?thesis* **using** *1* **by** (*simp add*:*divide-simps*)
**next**
**case** *False*
**hence** *card S* = *0* **by** *simp*
**thus** *?thesis* **by** *simp*
**qed**

**lemma** *edge-expansionI*:
**fixes** $\alpha$ :: *real*
**assumes** *n* > *1*
**assumes** $\bigwedge S.\ S \subseteq verts\ G \Longrightarrow 2 * card\ S \leq n \Longrightarrow S \neq \{\} \Longrightarrow card\ (edges\text{-}betw\ S\ (-S)) \geq \alpha *$
*card S*
**shows** $\Lambda_e \geq \alpha$
**proof** −
**define** *St* **where** $St = \{S.\ S \subseteq verts\ G \wedge 2 * card\ S \leq n \wedge S \neq \{\}\}$
**have** *0*: *finite St*
**unfolding** *St-def*
**by** (*rule finite-subset*[**where** *B*=*Pow* (*verts G*)]) *auto*

**obtain** *v* **where** *v-def*: $v \in verts\ G$ **using** *verts-non-empty* **by** *auto*

**have** $\{v\} \in St$
**using** *assms v-def* **unfolding** *St-def n-def* **by** *auto*
**hence** *1*: $St \neq \{\}$ **by** *auto*

**have** *2*: $\alpha \le$ *real* (*card* (*edges-betw* *S* (− *S*))) / *real* (*card* *S*) **if** $S \in St$ **for** *S*
**proof** −
  **have** *real* (*card* (*edges-betw* *S* (− *S*))) $\ge \alpha *$ *card* *S*
    **using** *assms*(*2*) *that* **unfolding** *St-def* **by** *simp*
  **moreover have** *finite* *S*
    **using** *that* **unfolding** *St-def*
    **by** (*intro* *finite-subset*[*OF* - *finite-verts*]) *auto*
  **hence** *card* *S* > *0*
    **using** *that* **unfolding** *St-def* **by** *auto*
  **ultimately show** *?thesis*
    **by** (*simp* *add*:*divide-simps*)
**qed**

**have** $\alpha \le$ (*MIN* *S*∈*St*. *real* (*card* (*edges-betw* *S* (− *S*))) / *real* (*card* *S*))
  **using** *0* *1* *2*
  **by** (*intro* *Min.boundedI* *finite-imageI*) *auto*

**thus** *?thesis*
  **unfolding** $\Lambda_e$-*def* *St-def*[*symmetric*] **using** *assms* **by** *auto*
**qed**

**end**

**lemma** *regular-graphI*:
  **assumes** *symmetric-multi-graph* *G*
  **assumes** *verts* *G* ≠ {} *d* > *0*
  **assumes** ⋀*v*. *v* ∈ *verts* *G* ⟹ *out-degree* *G* *v* = *d*
  **shows** *regular-graph* *G*
**proof** −
  **obtain** *v* **where** *v-def*: *v* ∈ *verts* *G*
    **using** *assms*(*2*) **by** *auto*
  **have** *arcs* *G* ≠ {}
  **proof** (*rule* *ccontr*)
    **assume** ¬*arcs* *G* ≠ {}
    **hence** *arcs* *G* = {} **by** *simp*
    **hence** *out-degree* *G* *v* = *0*
      **unfolding** *out-degree-def* *out-arcs-def* **by** *simp*
    **hence** *d* = *0*
      **using** *v-def* *assms*(*4*) **by** *simp*
    **thus** *False*
      **using** *assms*(*3*) **by** *simp*
  **qed**

  **thus** *?thesis*
    **using** *assms* *symmetric-multi-graphD2*[*OF* *assms*(*1*)]
    **unfolding** *regular-graph-def* *regular-graph-axioms-def*
    **by** *simp*
**qed**

The following theorems verify that a graph isomorphisms preserve symmetry, regularity and all the expansion coefficients.

**lemma** (**in** *fin-digraph*) *symmetric-graph-iso*:
  **assumes** *digraph-iso* *G* *H*
  **assumes** *symmetric-multi-graph* *G*
  **shows** *symmetric-multi-graph* *H*
**proof** −
  **obtain** *h* **where** *hom-iso*: *digraph-isomorphism* *h* **and** *H-alt*: *H* = *app-iso* *h* *G*

**using** *assms* **unfolding** *digraph-iso-def* **by** *auto*

**have** *0*:*fin-digraph H*
**unfolding** *H-alt*
**by** (*intro fin-digraphI-app-iso hom-iso*)

**interpret** *H*:*fin-digraph H*
**using** *0* **by** *auto*

**have** *1*:*arcs-betw H* (*iso-verts h v*) (*iso-verts h w*) = *iso-arcs h* ‘ *arcs-betw G v w*
(**is** *?L* = *?R*) **if** *v* ∈ *verts G w* ∈ *verts G* **for** *v w*
**proof** −
**have** *?L* = {*a* ∈ *iso-arcs h* ‘ *arcs G*. *iso-head h a*=*iso-verts h w* ∧ *iso-tail h a*=*iso-verts h v*}
**unfolding** *arcs-betw-def H-alt arcs-app-iso head-app-iso tail-app-iso* **by** *simp*
**also have** ... = {*a*. (∃ *b* ∈ *arcs G*. *a* = *iso-arcs h b* ∧ *iso-verts h* (*head G b*) = *iso-verts h w* ∧
*iso-verts h* (*tail G b*) = *iso-verts h v*)}
**using** *iso-verts-head*[*OF hom-iso*] *iso-verts-tail*[*OF hom-iso*] **by** *auto*
**also have** ... = {*a*. (∃ *b* ∈ *arcs G*. *a* = *iso-arcs h b* ∧ *head G b* = *w* ∧ *tail G b* = *v*)}
**using** *that iso-verts-eq-iff*[*OF hom-iso*] **by** *auto*
**also have** ... = *?R*
**unfolding** *arcs-betw-def* **by** (*auto simp add*:*image-iff set-eq-iff*)
**finally show** *?thesis* **by** *simp*
**qed**

**have** *card* (*arcs-betw H w v*) = *card* (*arcs-betw H v w*) (**is** *?L* = *?R*)
**if** *v-range*: *v* ∈ *verts H* **and** *w-range*: *w* ∈ *verts H* **for** *v w*
**proof** −
**obtain** *v′* **where** *v′*: *v* = *iso-verts h v′ v′* ∈ *verts G*
**using** *that v-range verts-app-iso* **unfolding** *H-alt* **by** *auto*
**obtain** *w′* **where** *w′*: *w* = *iso-verts h w′ w′* ∈ *verts G*
**using** *that w-range verts-app-iso* **unfolding** *H-alt* **by** *auto*
**have** *?L* = *card* (*arcs-betw H* (*iso-verts h w′*) (*iso-verts h v′*))
**unfolding** *v′ w′* **by** *simp*
**also have** ... = *card* (*iso-arcs h* ‘ *arcs-betw G w′ v′*)
**by** (*intro arg-cong*[**where** *f*=*card*] *1 v′ w′*)
**also have** ... = *card* (*arcs-betw G w′ v′*)
**using** *iso-arcs-eq-iff*[*OF hom-iso*] **unfolding** *arcs-betw-def*
**by** (*intro card-image inj-onI*) *auto*
**also have** ... = *card* (*arcs-betw G v′ w′*)
**by** (*intro symmetric-multi-graphD4 assms*(*2*))
**also have** ... = *card* (*iso-arcs h* ‘ *arcs-betw G v′ w′*)
**using** *iso-arcs-eq-iff*[*OF hom-iso*] **unfolding** *arcs-betw-def*
**by** (*intro card-image*[*symmetric*] *inj-onI*) *auto*
**also have** ... = *card* (*arcs-betw H* (*iso-verts h v′*) (*iso-verts h w′*))
**by** (*intro arg-cong*[**where** *f*=*card*] *1*[*symmetric*] *v′ w′*)
**also have** ... = *?R*
**unfolding** *v′ w′* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**thus** *?thesis*
**using** *0* **unfolding** *symmetric-multi-graph-def* **by** *auto*
**qed**

**lemma** (**in** *regular-graph*)
**assumes** *digraph-iso G H*
**shows** *regular-graph-iso*: *regular-graph H*
**and** *regular-graph-iso-size*: *regular-graph.n H* = *n*

**and** *regular-graph-iso-degree*: *regular-graph.d H* = *d*
**and** *regular-graph-iso-expansion-le*: *regular-graph*.$\Lambda_a$ *H* $\leq \Lambda_a$
**and** *regular-graph-iso-os-expansion-le*: *regular-graph*.$\Lambda_2$ *H* $\leq \Lambda_2$
**and** *regular-graph-iso-edge-expansion-ge*: *regular-graph*.$\Lambda_e$ *H* $\geq \Lambda_e$
**proof** −
**obtain** *h* **where** *hom-iso*: *digraph-isomorphism h* **and** *H-alt*: *H* = *app-iso h G*
**using** *assms* **unfolding** *digraph-iso-def* **by** *auto*

**have** *0*:*symmetric-multi-graph H*
**by** (*intro symmetric-graph-iso*[*OF assms*(*1*)] *sym*)

**have** *1*:*verts H* $\neq$ {}
**unfolding** *H-alt verts-app-iso* **using** *verts-non-empty* **by** *simp*

**then obtain** *h-wit* **where** *h-wit*: *h-wit* $\in$ *verts H*
**by** *auto*

**have** *3*:*out-degree H v* = *d* **if** *v-range*: *v* $\in$ *verts H* **for** *v*
**proof** −
**obtain** *v′* **where** *v′*: *v* = *iso-verts h v′ v′* $\in$ *verts G*
**using** *that v-range verts-app-iso* **unfolding** *H-alt* **by** *auto*
**have** *out-degree H v* = *out-degree G v′*
**unfolding** *v′ H-alt* **by** (*intro out-degree-app-iso-eq*[*OF hom-iso*] *v′*)
**also have** ... = *d*
**by** (*intro reg v′*)
**finally show** *?thesis* **by** *simp*
**qed**

**thus** *2*:*regular-graph H*
**by** (*intro regular-graphI*[**where** *d*=*d*] *0 d-gt-0 1*) *auto*
**interpret** *H*:*regular-graph H*
**using** *2* **by** *auto*

**have** *H.n* = *card* (*iso-verts h* ‘ *verts G*)
**unfolding** *H.n-def* **unfolding** *H-alt verts-app-iso* **by** *simp*
**also have** ... = *card* (*verts G*)
**by** (*intro card-image digraph-isomorphism-inj-on-verts hom-iso*)
**also have** ... = *n*
**unfolding** *n-def* **by** *simp*
**finally show** *n-eq*: *H.n* = *n* **by** *simp*

**have** *H.d* = *out-degree H h-wit*
**by** (*intro H.reg*[*symmetric*] *h-wit*)
**also have** ... = *d*
**by** (*intro 3 h-wit*)
**finally show** *4*:*H.d* = *d* **by** *simp*

**have** *bij-betw* (*iso-verts h*) (*verts G*) (*verts H*)
**unfolding** *H-alt* **using** *hom-iso*
**by** (*simp add*: *bij-betw-def digraph-isomorphism-inj-on-verts*)

**hence** *g-inner-conv*:
*H.g-inner f g* = *g-inner* ($\lambda x$. *f* (*iso-verts h x*)) ($\lambda x$. *g* (*iso-verts h x*))
**for** *f g* :: *′c* $\Rightarrow$ *real*
**unfolding** *g-inner-def H.g-inner-def* **by** (*intro sum.reindex-bij-betw*[*symmetric*])

**have** *g-step-conv*:
*H.g-step f* (*iso-verts h x*) = *g-step* ($\lambda x$. *f* (*iso-verts h x*)) *x* **if** *x* $\in$ *verts G*

```
    for f :: 'c ⇒ real and x
  proof -
    have inj-on (iso-arcs h) (in-arcs G x)
      using inj-on-subset[OF digraph-isomorphism-inj-on-arcs[OF hom-iso]]
      by (simp add:in-arcs-def)
    moreover have in-arcs H (iso-verts h x) = iso-arcs h ` in-arcs G x
      unfolding H-alt by (intro in-arcs-app-iso-eq[OF hom-iso] that)
    moreover have tail H (iso-arcs h a) = iso-verts h (tail G a) if a ∈ in-arcs G x for a
      unfolding H-alt using that by (simp add: hom-iso iso-verts-tail)
    ultimately show ?thesis
      unfolding g-step-def H.g-step-def
      by (intro-cong [σ₂(/), σ₁ f, σ₁ of-nat] more: 4 sum.reindex-cong[where l=iso-arcs h])
  qed

  show H.Λa ≤ Λa
    using expansionD1 by (intro H.expander-intro-1 Λ-ge-0)
      (simp add:g-inner-conv g-step-conv H.g-norm-sq g-norm-sq cong:g-inner-cong)

  show H.Λ₂ ≤ Λ₂
  proof (cases n > 1)
    case True
    hence H.n > 1
      by (simp add:n-eq)
    thus ?thesis
      using os-expanderD by (intro H.os-expanderI)
        (simp-all add:g-inner-conv g-step-conv H.g-norm-sq g-norm-sq cong:g-inner-cong)
  next
    case False
    thus ?thesis
      unfolding H.Λ₂-def Λ₂-def by (simp add:n-eq)
  qed

  show H.Λe ≥ Λe
  proof (cases n > 1)
    case True
    hence n-gt-1: H.n > 1
      by (simp add:n-eq)
    have Λe * real (card S) ≤ real (card (H.edges-betw S (- S)))
      if S ⊆ verts H 2 * card S ≤ H.n S ≠ {} for S
    proof -
      define T where T = iso-verts h -` S ∩ verts G
      have 4:card T = card S
        using that(1) unfolding T-def H-alt verts-app-iso
        by (intro card-vimage-inj-on digraph-isomorphism-inj-on-verts[OF hom-iso]) auto

      have card (H.edges-betw S (-S))=card {a∈iso-arcs h`arcs G. iso-tail h a∈S∧iso-head h a∈
-S}
        unfolding H.edges-betw-def unfolding H-alt tail-app-iso head-app-iso arcs-app-iso
        by simp
      also have ...=
        card(iso-arcs h` {a ∈ arcs G. iso-tail h (iso-arcs h a)∈S∧ iso-head h (iso-arcs h a)∈-S})
        by (intro arg-cong[where f=card]) auto
     also have ... = card {a ∈ arcs G. iso-tail h (iso-arcs h a)∈S∧ iso-head h (iso-arcs h a)∈-S}
       by (intro card-image inj-on-subset[OF digraph-isomorphism-inj-on-arcs[OF hom-iso]]) auto
     also have ... = card {a ∈ arcs G. iso-verts h (tail G a) ∈ S ∧ iso-verts h (head G a) ∈ -S}
       by (intro restr-Collect-cong arg-cong[where f=card])
        (simp add: iso-verts-tail[OF hom-iso] iso-verts-head[OF hom-iso])
      also have ... = card {a ∈ arcs G. tail G a ∈ T ∧ head G a ∈ -T }
```

**unfolding** *T-def* **by** (*intro-cong* [$\sigma_1$(*card*),$\sigma_2$ ($\wedge$)] *more*: *restr-Collect-cong*) *auto*
**also have** ... = *card* (*edges-betw* $T$ ($-T$))
**unfolding** *edges-betw-def* **by** *simp*
**finally have** *5*:*card* (*edges-betw* $T$ ($-T$)) = *card* ($H$.*edges-betw* $S$ ($-S$))
**by** *simp*

**have** *6*: $T \subseteq$ *verts* $G$ **unfolding** *T-def* **by** *simp*

**have** $\Lambda_e$ $*$ *real* (*card* $S$) = $\Lambda_e$ $*$ *real* (*card* $T$)
**unfolding** *4* **by** *simp*
**also have** ... $\leq$ *real* (*card* (*edges-betw* $T$ ($-T$)))
**using** *that*(*2*) **by** (*intro* *edge-expansionD 6*) (*simp add*:*4 n-eq*)
**also have** ... = *real* (*card* ($H$.*edges-betw* $S$ ($-S$)))
**unfolding** *5* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**thus** *?thesis*
**by** (*intro* $H$.*edge-expansionI n-gt-1*) *auto*
**next**
**case** *False*
**thus** *?thesis*
**unfolding** $H$.$\Lambda_e$-*def* $\Lambda_e$-*def* **by** (*simp add*:*n-eq*)
**qed**

**qed**

**lemma** (**in** *regular-graph*)
**assumes** *digraph-iso* $G$ $H$
**shows** *regular-graph-iso-expansion*: *regular-graph*.$\Lambda_a$ $H$ = $\Lambda_a$
**and** *regular-graph-iso-os-expansion*: *regular-graph*.$\Lambda_2$ $H$ = $\Lambda_2$
**and** *regular-graph-iso-edge-expansion*: *regular-graph*.$\Lambda_e$ $H$ = $\Lambda_e$
**proof** −
**interpret** $H$:*regular-graph* $H$
**by** (*intro* *regular-graph-iso assms*)

**have** *iso*:*digraph-iso* $H$ $G$
**using** *digraph-iso-swap assms wf-digraph-axioms* **by** *blast*

**hence** $\Lambda_a \leq H.\Lambda_a$
**by** (*intro* $H$.*regular-graph-iso-expansion-le*)
**moreover have** $H.\Lambda_a \leq \Lambda_a$
**using** *regular-graph-iso-expansion-le*[*OF assms*] **by** *auto*
**ultimately show** $H.\Lambda_a = \Lambda_a$
**by** *auto*

**have** $\Lambda_2 \leq H.\Lambda_2$ **using** *iso*
**by** (*intro* $H$.*regular-graph-iso-os-expansion-le*)
**moreover have** $H.\Lambda_2 \leq \Lambda_2$
**using** *regular-graph-iso-os-expansion-le*[*OF assms*] **by** *auto*
**ultimately show** $H.\Lambda_2 = \Lambda_2$
**by** *auto*

**have** $\Lambda_e \geq H.\Lambda_e$ **using** *iso*
**by** (*intro* $H$.*regular-graph-iso-edge-expansion-ge*)
**moreover have** $H.\Lambda_e \geq \Lambda_e$
**using** *regular-graph-iso-edge-expansion-ge*[*OF assms*] **by** *auto*
**ultimately show** $H.\Lambda_e = \Lambda_e$

**by** *auto*
**qed**

**unbundle** *no intro-cong-syntax*

**end**

# 4 Setup for Types to Sets

**theory** *Expander-Graphs-TTS*
**imports**
*Expander-Graphs-Definition*
*HOL−Analysis.Cartesian-Space*
*HOL−Types-To-Sets.Types-To-Sets*
**begin**

This section sets up a sublocale with the assumption that there is a finite type with the same cardinality as the vertex set of a regular graph. This allows defining the adjacency matrix for the graph using type-based linear algebra.

Theorems shown in the sublocale that do not refer to the local type are then lifted to the *regular-graph* locale using the Types-To-Sets mechanism.

**locale** *regular-graph-tts* = *regular-graph* +
**fixes** *n-itself* :: (*'n* :: *finite*) *itself*
**assumes** *td*: ∃(*f* :: (*'n* ⇒ *'a*)) *g*. *type-definition f g* (*verts G*)
**begin**

**definition** *td-components* :: (*'n* ⇒ *'a*) × (*'a* ⇒ *'n*)
**where** *td-components* = (*SOME q*. *type-definition* (*fst q*) (*snd q*) (*verts G*))

**definition** *enum-verts* **where** *enum-verts* = *fst td-components*
**definition** *enum-verts-inv* **where** *enum-verts-inv* = *snd td-components*

**sublocale** *type-definition enum-verts enum-verts-inv verts G*
**proof** −
**have** *0*:∃ *q*. *type-definition* ((*fst q*)::(*'n* ⇒ *'a*)) (*snd q*) (*verts G*)
**using** *td* **by** *simp*
**show** *type-definition enum-verts enum-verts-inv* (*verts G*)
**unfolding** *td-components-def enum-verts-def enum-verts-inv-def* **using** *someI-ex*[*OF 0*] **by** *simp*
**qed**

**lemma** *enum-verts*: *bij-betw enum-verts UNIV* (*verts G*)
**unfolding** *bij-betw-def* **by** (*simp add*: *Rep-inject Rep-range inj-on-def*)

The stochastic matrix associated to the graph.

**definition** *A* :: (*'c::field*)^*'n*^*'n* **where**
*A* = (χ *i j*. *of-nat* (*count* (*edges G*) (*enum-verts j*,*enum-verts i*))/*of-nat d*)

**lemma** *card-n*: *CARD*(*'n*) = *n*
**unfolding** *n-def card* **by** *simp*

**lemma** *symmetric-A*: *transpose A* = *A*
**proof** −
**have** *A* $ *i* $ *j* = *A* $ *j* $ *i* **for** *i j*
**unfolding** *A-def count-edges arcs-betw-def* **using** *symmetric-multi-graphD*[*OF sym*]
**by** *auto*

**thus** *?thesis*
  **unfolding** *transpose-def*
  **by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) *auto*
**qed**

**lemma** *g-step-conv*:
  $(\chi\ i.\ \textit{g-step}\ f\ (\textit{enum-verts}\ i)) = A *v\ (\chi\ i.\ f\ (\textit{enum-verts}\ i))$
**proof** −
  **have** $\textit{g-step}\ f\ (\textit{enum-verts}\ i) = (\sum j{\in}\textit{UNIV}.\ A\ \$\ i\ \$\ j * f\ (\textit{enum-verts}\ j))$ (**is** *?L* = *?R*) **for** *i*
  **proof** −
    **have** $?L = (\sum x{\in}\textit{in-arcs}\ G\ (\textit{enum-verts}\ i).\ f\ (\textit{tail}\ G\ x)\ /\ d)$
      **unfolding** *g-step-def* **by** *simp*
    **also have** ... $= (\sum x{\in}\#\textit{vertices-to}\ G\ (\textit{enum-verts}\ i).\ f\ x/d)$
        **unfolding** *verts-to-alt sum-unfold-sum-mset* **by** (*simp add*:*image-mset.compositionality comp-def*)
    **also have** ... $= (\sum j{\in}\textit{verts}\ G.\ (\textit{count}\ (\textit{vertices-to}\ G\ (\textit{enum-verts}\ i))\ j) * (f\ j\ /\ \textit{real}\ d))$
      **by** (*intro sum-mset-conv-2 set-mset-vertices-to*) *auto*
    **also have** ... $= (\sum j{\in}\textit{verts}\ G.\ (\textit{count}\ (\textit{edges}\ G)\ (j,\textit{enum-verts}\ i)) * (f\ j\ /\ \textit{real}\ d))$
      **unfolding** *vertices-to-def count-mset-exp*
      **by** (*intro sum.cong arg-cong*[**where** *f*=*real*] *arg-cong2*[**where** *f*=(∗)])
      (*auto simp add*:*filter-filter-mset image-mset-filter-mset-swap*[*symmetric*] *prod-eq-iff ac-simps*)
    **also have** ...$=(\sum j{\in}\textit{UNIV}.(\textit{count}(\textit{edges}\ G)(\textit{enum-verts}\ j,\textit{enum-verts}\ i))*(f(\textit{enum-verts}\ j)/\textit{real}\ d))$
      **by** (*intro sum.reindex-bij-betw*[*symmetric*] *enum-verts*)
    **also have** ... = *?R*
      **unfolding** *A-def* **by** *simp*
    **finally show** *?thesis* **by** *simp*
  **qed**
  **thus** *?thesis*
    **unfolding** *matrix-vector-mult-def* **by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) *simp*
**qed**

**lemma** *g-inner-conv*:
  $\textit{g-inner}\ f\ g = (\chi\ i.\ f\ (\textit{enum-verts}\ i)) \cdot (\chi\ i.\ g\ (\textit{enum-verts}\ i))$
  **unfolding** *inner-vec-def g-inner-def vec-lambda-beta inner-real-def conjugate-real-def*
  **by** (*intro sum.reindex-bij-betw*[*symmetric*] *enum-verts*)

**lemma** *g-norm-conv*:
  $\textit{g-norm}\ f = \textit{norm}\ (\chi\ i.\ f\ (\textit{enum-verts}\ i))$
**proof** −
  **have** $\textit{g-norm}\ f\hat{\ }2 = \textit{norm}\ (\chi\ i.\ f\ (\textit{enum-verts}\ i))\hat{\ }2$
    **unfolding** *g-norm-sq power2-norm-eq-inner g-inner-conv* **by** *simp*
  **thus** *?thesis*
    **using** *g-norm-nonneg norm-ge-zero* **by** *simp*
**qed**

**end**

**lemma** *eg-tts-1*:
  **assumes** *regular-graph G*
  **assumes** $\exists (f{::}('n{::}\textit{finite}) \Rightarrow 'a)\ g.\ \textit{type-definition}\ f\ g\ (\textit{verts}\ G)$
  **shows** *regular-graph-tts TYPE*('*n*) *G*
  **using** *assms*
  **unfolding** *regular-graph-tts-def regular-graph-tts-axioms-def* **by** *auto*

**context** *regular-graph*
**begin**

```
lemma remove-finite-premise-aux:
  assumes ∃(Rep :: 'n ⇒ 'a) Abs. type-definition Rep Abs (verts G)
  shows class.finite TYPE('n)
proof −
  obtain Rep :: 'n ⇒ 'a and Abs where d:type-definition Rep Abs (verts G)
    using assms by auto
  interpret type-definition Rep Abs verts G
    using d by simp

  have finite (verts G) by simp
  thus ?thesis
    unfolding class.finite-def univ by auto
qed

lemma remove-finite-premise:
  (class.finite TYPE('n) ⟹ ∃(Rep :: 'n ⇒ 'a) Abs. type-definition Rep Abs (verts G) ⟹ PROP
Q)
  ≡ (∃(Rep :: 'n ⇒ 'a) Abs. type-definition Rep Abs (verts G) ⟹ PROP Q)
  (is ?L ≡ ?R)
proof (intro Pure.equal-intr-rule)
  assume e:∃(Rep :: 'n ⇒ 'a) Abs. type-definition Rep Abs (verts G) and l:PROP ?L
  hence f: class.finite TYPE('n)
    using remove-finite-premise-aux[OF e] by simp

  show PROP ?R
    using l[OF f] by auto
next
  assume ∃(Rep :: 'n ⇒ 'a) Abs. type-definition Rep Abs (verts G) and l:PROP ?R
  show PROP ?L
    using l by auto
qed

end

end
```

## 5 Algebra-only Theorems

This section verifies the linear algebraic counter-parts of the graph-theoretic theorems about Random walks. The graph-theoretic results are then derived in Section 9.

```
theory Expander-Graphs-Algebra
  imports
    HOL−Library.Monad-Syntax
    Expander-Graphs-TTS
begin

lemma pythagoras:
  fixes v w :: 'a::real-inner
  assumes v · w = 0
  shows norm (v+w)^2 = norm v^2 + norm w^2
  using assms by (simp add:power2-norm-eq-inner algebra-simps inner-commute)

definition diag :: ('a :: zero)^'n ⇒ 'a^'n^'n
  where diag v = (χ i j. if i = j then (v $ i) else 0)

definition ind-vec :: 'n set ⇒ real^'n
  where ind-vec S = (χ i. of-bool( i ∈ S))
```

```
lemma diag-mult-eq: diag x ** diag y = diag (x * y)
  unfolding diag-def
  by (vector matrix-matrix-mult-def)
   (auto simp add:if-distrib if-distribR sum.If-cases)

lemma diag-vec-mult-eq: diag x *v y = x * y
  unfolding diag-def matrix-vector-mult-def
  by (simp add:if-distrib if-distribR sum.If-cases times-vec-def)

definition matrix-norm-bound :: real^'n^'m ⇒ real ⇒ bool
  where matrix-norm-bound A l = (∀ x. norm (A *v x) ≤ l * norm x)

lemma matrix-norm-boundI:
  assumes ⋀x. norm (A *v x) ≤ l * norm x
  shows matrix-norm-bound A l
  using assms unfolding matrix-norm-bound-def by simp

lemma matrix-norm-boundD:
  assumes matrix-norm-bound A l
  shows norm (A *v x) ≤ l * norm x
  using assms unfolding matrix-norm-bound-def by simp

lemma matrix-norm-bound-nonneg:
  fixes A :: real^'n^'m
  assumes matrix-norm-bound A l
  shows l ≥ 0
proof −
  have 0 ≤ norm (A *v 1) by simp
  also have ... ≤ l * norm (1::real^'n)
    using assms(1) unfolding matrix-norm-bound-def by simp
  finally have 0 ≤ l * norm (1::real^'n)
    by simp
  moreover have norm (1::real^'n) > 0
    by simp
  ultimately show ?thesis
    by (simp add: zero-le-mult-iff)
qed

lemma matrix-norm-bound-0:
  assumes matrix-norm-bound A 0
  shows A = (0::real^'n^'m)
proof (intro iffD2[OF matrix-eq] allI)
  fix x :: real^'n
  have norm (A *v x) = 0
    using assms unfolding matrix-norm-bound-def by simp
  thus A *v x = 0 *v x
    by simp
qed

lemma matrix-norm-bound-diag:
  fixes x :: real^'n
  assumes ⋀i. |x $ i| ≤ l
  shows matrix-norm-bound (diag x) l
proof (rule matrix-norm-boundI)
  fix y :: real^'n

  have l-ge-0: l ≥ 0 using assms by fastforce
```

```
  have a: |x $ i * v| ≤ |l * v| for v i
    using l-ge-0 assms by (simp add:abs-mult mult-right-mono)

  have norm (diag x *v y) = sqrt (∑ i ∈ UNIV. (x $ i * y $ i)^2)
    unfolding matrix-vector-mult-def diag-def norm-vec-def L2-set-def
    by (auto simp add:if-distrib if-distribR sum.If-cases)
  also have ... ≤ sqrt (∑ i ∈ UNIV. (l * y $ i)^2)
    by (intro real-sqrt-le-mono sum-mono iffD1[OF abs-le-square-iff] a)
  also have ... = l * norm y
    using l-ge-0 by (simp add:norm-vec-def L2-set-def algebra-simps
        sum-distrib-left[symmetric] real-sqrt-mult)
  finally show norm (diag x *v y) ≤ l * norm y by simp
qed

lemma vector-scaleR-matrix-ac-2: b *R (A::real^'n^'m) *v x = b *R (A *v x)
  unfolding vector-transpose-matrix[symmetric]  transpose-scalar
  by (intro vector-scaleR-matrix-ac)

lemma  matrix-norm-bound-scale:
  assumes matrix-norm-bound A l
  shows matrix-norm-bound (b *R A) (|b| * l)
proof (intro matrix-norm-boundI)
  fix x
  have norm (b *R A *v x) = norm (b *R (A *v x))
    by (metis transpose-scalar vector-scaleR-matrix-ac vector-transpose-matrix)
  also have ... = |b| * norm (A *v x)
    by simp
  also have ... ≤ |b| * (l * norm x)
    using assms matrix-norm-bound-def by (intro mult-left-mono) auto
  also have ... ≤ (|b| * l) * norm x by simp
  finally show norm (b *R A *v x) ≤ (|b| * l) * norm x by simp
qed

definition nonneg-mat :: real^'n^'m ⇒ bool
  where nonneg-mat A = (∀ i j. A $ i $ j ≥ 0)

lemma nonneg-mat-1:
  shows nonneg-mat (mat 1)
  unfolding nonneg-mat-def mat-def by auto

lemma nonneg-mat-prod:
  assumes nonneg-mat A nonneg-mat B
  shows nonneg-mat (A ** B)
  using assms unfolding nonneg-mat-def matrix-matrix-mult-def
  by (auto intro:sum-nonneg)

lemma nonneg-mat-transpose:
  nonneg-mat (transpose A) = nonneg-mat A
  unfolding nonneg-mat-def transpose-def
  by auto

definition spec-bound :: real^'n^'n ⇒ real ⇒ bool
  where spec-bound M l = (l ≥ 0 ∧ (∀ v. v · 1 = 0 ⟶ norm (M *v v) ≤ l * norm v))

lemma spec-boundD1:
  assumes spec-bound M l
  shows 0 ≤ l
```

using *assms* **unfolding** *spec-bound-def* **by** *simp*

**lemma** *spec-boundD2*:
  **assumes** *spec-bound M l*
  **assumes** $v \cdot 1 = 0$
  **shows** *norm* $(M *v\ v) \leq l * norm\ v$
  **using** *assms* **unfolding** *spec-bound-def* **by** *simp*

**lemma** *spec-bound-mono*:
  **assumes** *spec-bound M* $\alpha$ $\alpha \leq \beta$
  **shows** *spec-bound M* $\beta$
**proof** −
  **have** *norm* $(M *v\ v) \leq \beta * norm\ v$ **if** *inner v 1* $= 0$ **for** *v*
  **proof** −
    **have** *norm* $(M *v\ v) \leq \alpha * norm\ v$
      **by** (*intro spec-boundD2*[*OF assms*(*1*)] *that*)
    **also have** ... $\leq \beta * norm\ v$
      **by** (*intro mult-right-mono assms*(*2*)) *auto*
    **finally show** *?thesis* **by** *simp*
  **qed**
  **moreover have** $\beta \geq 0$
    **using** *assms*(*2*) *spec-boundD1*[*OF assms*(*1*)] **by** *simp*
  **ultimately show** *?thesis*
    **unfolding** *spec-bound-def* **by** *simp*
**qed**

**definition** *markov* :: *real^'n^'n* $\Rightarrow$ *bool*
  **where** *markov M* = (*nonneg-mat M* $\wedge$ *M *v 1 = 1* $\wedge$ *1 v* M = 1*)

**lemma** *markov-symI*:
  **assumes** *nonneg-mat A transpose A = A A *v 1 = 1*
  **shows** *markov A*
**proof** −
  **have** *1 v* A = transpose A *v 1*
    **unfolding** *vector-transpose-matrix*[*symmetric*] **by** *simp*
  **also have** ... = *1* **unfolding** *assms*(*2*,*3*) **by** *simp*
  **finally have** *1 v* A = 1* **by** *simp*
  **thus** *?thesis*
    **unfolding** *markov-def* **using** *assms* **by** *auto*
**qed**

**lemma** *markov-apply*:
  **assumes** *markov M*
  **shows** *M *v 1 = 1 1 v* M = 1*
  **using** *assms* **unfolding** *markov-def* **by** *auto*

**lemma** *markov-transpose*:
  *markov A = markov* (*transpose A*)
  **unfolding** *markov-def nonneg-mat-transpose* **by** *auto*
**fun** *matrix-pow* **where**
  *matrix-pow M 0 = mat 1* |
  *matrix-pow M* (*Suc n*) = *M *** (*matrix-pow M n*)

**lemma** *markov-orth-inv*:
  **assumes** *markov A*
  **shows** *inner* (*A *v x*) *1 = inner x 1*
**proof** −
  **have** *inner* (*A *v x*) *1 = inner x* (*1 v* A*)

    using *dot-lmul-matrix inner-commute* **by** *metis*
  **also have** ... = *inner x 1*
    **using** *markov-apply*[*OF assms*(*1*)] **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *markov-id*:
  *markov* (*mat 1*)
  **unfolding** *markov-def* **using** *nonneg-mat-1* **by** *simp*

**lemma** *markov-mult*:
  **assumes** *markov A markov B*
  **shows** *markov* (*A* ** *B*)
**proof** −
  **have** *nonneg-mat* (*A* ** *B*)
    **using** *assms* **unfolding** *markov-def* **by** (*intro nonneg-mat-prod*) *auto*
  **moreover have** (*A* ** *B*) *v *1* = *1*
    **using** *assms* **unfolding** *markov-def*
    **unfolding** *matrix-vector-mul-assoc*[*symmetric*] **by** *simp*
  **moreover have** *1 v** (*A* ** *B*) = *1*
    **using** *assms* **unfolding** *markov-def*
    **unfolding** *vector-matrix-mul-assoc*[*symmetric*] **by** *simp*
  **ultimately show** *?thesis*
    **unfolding** *markov-def* **by** *simp*
**qed**

**lemma** *markov-matrix-pow*:
  **assumes** *markov A*
  **shows** *markov* (*matrix-pow A k*)
  **using** *markov-id assms markov-mult*
  **by** (*induction k*, *auto*)

**lemma** *spec-bound-prod*:
  **assumes** *markov A markov B*
  **assumes** *spec-bound A la spec-bound B lb*
  **shows** *spec-bound* (*A* ** *B*) (*la*lb*)
**proof** −
  **have** *la-ge-0*: *la* ≥ *0* **using** *spec-boundD1*[*OF assms*(*3*)] **by** *simp*

  **have** *norm* ((*A* ** *B*) *v *x*) ≤ (*la* * *lb*) * *norm x* **if** *inner x 1* = *0* **for** *x*
  **proof** −
    **have** *norm* ((*A* ** *B*) *v *x*) = *norm* (*A* *v (*B* *v *x*))
      **by** (*simp add*:*matrix-vector-mul-assoc*)
    **also have** ... ≤ *la* * *norm* (*B* *v *x*)
      **by** (*intro spec-boundD2*[*OF assms*(*3*)]) (*simp add*:*markov-orth-inv that assms*(*2*))
    **also have** ... ≤ *la* * (*lb* * *norm x*)
      **by** (*intro spec-boundD2*[*OF assms*(*4*)] *mult-left-mono that la-ge-0*)
    **finally show** *?thesis* **by** *simp*
  **qed**
  **moreover have** *la* * *lb* ≥ *0*
    **using** *la-ge-0 spec-boundD1*[*OF assms*(*4*)] **by** *simp*
  **ultimately show** *?thesis*
    **using** *spec-bound-def* **by** *auto*
**qed**

**lemma** *spec-bound-pow*:
  **assumes** *markov A*
  **assumes** *spec-bound A l*

**shows** *spec-bound* (*matrix-pow A k*) ($l$^$k$)
**proof** (*induction k*)
**case** *0*
**then show** *?case* **unfolding** *spec-bound-def* **by** *simp*
**next**
**case** (*Suc k*)
**have** *spec-bound* (*A* ∗∗ *matrix-pow A k*) ($l * l$ ^ $k$)
**by** (*intro spec-bound-prod assms Suc markov-matrix-pow*)
**thus** *?case* **by** *simp*
**qed**

**fun** *intersperse* :: $'a \Rightarrow 'a$ *list* $\Rightarrow 'a$ *list*
**where**
*intersperse x* [] = [] |
*intersperse x* (*y*#[]) = *y*#[] |
*intersperse x* (*y*#*z*#*zs*) = *y*#*x*#*intersperse x* (*z*#*zs*)

**lemma** *intersperse-snoc*:
**assumes** *xs* ≠ []
**shows** *intersperse z* (*xs*@[*y*]) = *intersperse z xs*@[*z*,*y*]
**using** *assms*
**proof** (*induction xs rule*:*list-nonempty-induct*)
**case** (*single x*)
**then show** *?case* **by** *simp*
**next**
**case** (*cons x xs*)
**then obtain** *xsh xst* **where** *t*:*xs* = *xsh*#*xst*
**by** (*metis neq-Nil-conv*)
**have** *intersperse z* ((*x* # *xs*) @ [*y*]) = *x*#*z*#*intersperse z* (*xs*@[*y*])
**unfolding** *t* **by** *simp*
**also have** ... = *x*#*z*#*intersperse z xs*@[*z*,*y*]
**using** *cons* **by** *simp*
**also have** ... = *intersperse z* (*x*#*xs*)@[*z*,*y*]
**unfolding** *t* **by** *simp*
**finally show** *?case* **by** *simp*
**qed**

**lemma** *foldl-intersperse*:
**assumes** *xs* ≠ []
**shows** *foldl f a* ((*intersperse x xs*)@[*x*]) = *foldl* (λ*y z*. *f* (*f y z*) *x*) *a xs*
**using** *assms* **by** (*induction xs rule*:*rev-nonempty-induct*) (*auto simp add*:*intersperse-snoc*)

**lemma** *foldl-intersperse-2*:
**shows** *foldl f a* (*intersperse y* (*x*#*xs*)) = *foldl* (λ*x z*. *f* (*f x y*) *z*) (*f a x*) *xs*
**proof** (*induction xs rule*:*rev-induct*)
**case** *Nil*
**then show** *?case* **by** *simp*
**next**
**case** (*snoc xst xs*)
**have** *foldl f a* (*intersperse y* ((*x* # *xs*) @ [*xst*])) = *foldl* (λ*x*. *f* (*f x y*)) (*f a x*) (*xs* @ [*xst*])
**by** (*subst intersperse-snoc*, *auto simp add*:*snoc*)
**then show** *?case* **by** *simp*
**qed**

**context** *regular-graph-tts*
**begin**

**definition** *stat* :: *real*^'*n*
  **where** *stat* = (*1* / *real* *CARD*('*n*)) $*_R$ *1*

**definition** *J* :: ('*c* :: *field*)^'*n*^'*n*
  **where** *J* = (χ *i j*. *of-nat* *1* / *of-nat* *CARD*('*n*))

**lemma** *inner-1-1*: *1* · (*1*::*real*^'*n*) = *CARD*('*n*)
  **unfolding** *inner-vec-def* **by** *simp*

**definition** *proj-unit* :: *real*^'*n* ⇒ *real*^'*n*
  **where** *proj-unit* *v* = (*1* · *v*) $*_R$ *stat*

**definition** *proj-rem* :: *real*^'*n* ⇒ *real*^'*n*
  **where** *proj-rem* *v* = *v* − *proj-unit* *v*

**lemma** *proj-rem-orth*: *1* · (*proj-rem* *v*) = *0*
  **unfolding** *proj-rem-def* *proj-unit-def* *inner-diff-right* *stat-def*
  **by** (*simp* *add*:*inner-1-1*)

**lemma** *split-vec*: *v* = *proj-unit* *v* + *proj-rem* *v*
  **unfolding** *proj-rem-def* **by** *simp*

**lemma** *apply-J*: *J* *v *x* = *proj-unit* *x*
**proof** (*intro* *iffD2*[*OF* *vec-eq-iff*] *allI*)
  **fix** *i*
  **have** (*J* *v *x*) $ *i* = *inner* (χ *j*. *1* / *real* *CARD*('*n*)) *x*
    **unfolding** *matrix-vector-mul-component* *J-def* **by** *simp*
  **also have** ... = *inner* *stat* *x*
    **unfolding** *stat-def* *scaleR-vec-def* **by** *auto*
  **also have** ... = (*proj-unit* *x*) $ *i*
    **unfolding** *proj-unit-def* *stat-def* **by** *simp*
  **finally show** (*J* *v *x*) $ *i* = (*proj-unit* *x*) $ *i* **by** *simp*
**qed**

**lemma** *spec-bound-J*: *spec-bound* (*J* :: *real*^'*n*^'*n*) *0*
**proof** −
  **have** *norm* (*J* *v *v*) = *0* **if** *inner* *v* *1* = *0* **for** *v* :: *real*^'*n*
  **proof** −
    **have** *inner* (*proj-unit* *v* + *proj-rem* *v*) *1* = *0*
      **using** *that* **by** (*subst* (*asm*) *split-vec*[*of* *v*], *simp*)
    **hence** *inner* (*proj-unit* *v*) *1* = *0*
      **using** *proj-rem-orth* *inner-commute* **unfolding** *inner-add-left*
      **by** (*metis* *add-cancel-left-right*)
    **hence** *proj-unit* *v* = *0*
      **unfolding** *proj-unit-def* *stat-def* **by** *simp*
    **hence** *J* *v *v* = *0*
      **unfolding** *apply-J* **by** *simp*
    **thus** *?thesis* **by** *simp*
  **qed**
  **thus** *?thesis*
    **unfolding** *spec-bound-def* **by** *simp*
**qed**

**lemma** *matrix-decomposition-lemma-aux*:
  **fixes** *A* :: *real*^'*n*^'*n*
  **assumes** *markov* *A*
  **shows** *spec-bound* *A* *l* ⟷ *matrix-norm-bound* (*A* − (*1*−*l*) $*_R$ *J*) *l* (**is** *?L* ⟷ *?R*)
**proof**

```
  assume a:?L
  hence l-ge-0: l ≥ 0 using spec-boundD1 by auto
  show ?R
  proof (rule matrix-norm-boundI)
    fix x :: real^'n
    have (A − (1−l) *R J) *v x = A *v x − (1−l) *R (proj-unit x)
      by (simp add:algebra-simps vector-scaleR-matrix-ac-2 apply-J)
    also have ... = A *v proj-unit x + A *v proj-rem x − (1−l) *R (proj-unit x)
      by (subst split-vec[of x], simp add:algebra-simps)
    also have ... = proj-unit x + A *v proj-rem x − (1−l) *R (proj-unit x)
      using markov-apply[OF assms(1)]
      unfolding proj-unit-def stat-def by (simp add:algebra-simps)
    also have ... = A *v proj-rem x + l *R proj-unit x (is - = ?R1)
      by (simp add:algebra-simps)
    finally have d:(A − (1−l) *R J) *v x = ?R1 by simp

    have inner (l *R proj-unit x) (A *v proj-rem x) =
      inner ((l * inner 1 x / real CARD('n)) *R 1 v* A) (proj-rem x)
      by (subst dot-lmul-matrix[symmetric]) (simp add:proj-unit-def stat-def)
    also have ... = (l * inner 1 x / real CARD('n)) * inner 1 (proj-rem x)
      unfolding scaleR-vector-matrix-assoc markov-apply[OF assms] by simp
    also have ... = 0
      unfolding proj-rem-orth by simp
    finally have b:inner (l *R proj-unit x) (A *v proj-rem x) = 0 by simp

    have c: inner (proj-rem x) (proj-unit x) = 0
      using proj-rem-orth[of x]
      unfolding proj-unit-def stat-def by (simp add:inner-commute)

    have norm (?R1)^2 = norm (A *v proj-rem x)^2 + norm (l *R proj-unit x)^2
      using b by (intro pythagoras) (simp add:inner-commute)
    also have ... ≤ (l * norm (proj-rem x))^2 + norm (l *R proj-unit x)^2
      using proj-rem-orth[of x]
      by (intro add-mono power-mono spec-boundD2 a) (auto simp add:inner-commute)
    also have ... = l^2 * (norm (proj-rem x)^2 + norm (proj-unit x)^2)
      by (simp add:algebra-simps)
    also have ... = l^2 * (norm (proj-rem x + proj-unit x)^2)
      using c by (subst pythagoras) auto
    also have ... = l^2 * norm x^2
      by (subst (3) split-vec[of x]) (simp add:algebra-simps)
    also have ... = (l * norm x)^2
      by (simp add:algebra-simps)
    finally have norm (?R1)^2 ≤ (l * norm x)^2 by simp
    hence norm (?R1) ≤ l * norm x
      using l-ge-0 by (subst (asm) power-mono-iff) auto

    thus norm ((A − (1−l) *R J) *v x) ≤ l * norm x
      unfolding d by simp
  qed
next
  assume a:?R
  have norm (A *v x) ≤ l * norm x if inner x 1 = 0 for x
  proof −
    have (1 − l) *R J *v x = (1 − l) *R (proj-unit x)
      by (simp add:vector-scaleR-matrix-ac-2 apply-J)
    also have ... = 0
      unfolding proj-unit-def using that by (simp add:inner-commute)
    finally have b: (1 − l) *R J *v x = 0 by simp
```

```
    have norm (A *v x) = norm ((A − (1−l) *R J) *v x  + ((1−l) *R J) *v x)
      by (simp add:algebra-simps)
    also have ... ≤ norm ((A − (1−l) *R J) *v x) + norm (((1−l) *R J) *v x)
      by (intro norm-triangle-ineq)
    also have ... ≤ l * norm x + 0
      using a b unfolding  matrix-norm-bound-def by (intro add-mono, auto)
    also have ... = l * norm x
      by simp
    finally show ?thesis by simp
  qed

  moreover have l ≥ 0
    using a matrix-norm-bound-nonneg by blast

  ultimately show ?L
    unfolding spec-bound-def by simp
qed

lemma matrix-decomposition-lemma:
  fixes A :: real^'n^'n
  assumes markov A
  shows spec-bound A l ⟷ (∃ E. A = (1−l) *R J + l *R E ∧ matrix-norm-bound E 1 ∧ l ≥ 0)
    (is ?L ⟷ ?R)
proof −
  have ?L ⟷ matrix-norm-bound (A − (1−l) *R J) l
    using matrix-decomposition-lemma-aux[OF assms] by simp
  also have ... ⟷ ?R
  proof
    assume a:matrix-norm-bound (A − (1 − l) *R J) l
    hence l-ge-0: l ≥ 0 using matrix-norm-bound-nonneg by auto
    define E where E = (1/l) *R (A − (1−l) *R J)
    have A = J if l = 0
    proof −
      have matrix-norm-bound (A − J) 0
        using a that by simp
      hence A − J = 0 using matrix-norm-bound-0 by blast
      thus A = J by simp
    qed
    hence A = (1−l) *R J + l *R E
      unfolding E-def by simp
    moreover have matrix-norm-bound E 1
    proof (cases l = 0)
      case True
      hence E = 0 if l = 0
        unfolding E-def by simp
      thus matrix-norm-bound E 1 if l = 0
        using that unfolding matrix-norm-bound-def by auto
    next
      case False
      hence l > 0 using l-ge-0 by simp
      moreover have matrix-norm-bound E (|1 / l|* l)
        unfolding E-def
        by (intro matrix-norm-bound-scale a)
      ultimately show ?thesis by auto
    qed
    ultimately show ?R using l-ge-0 by auto
  next
```

**assume** *a*:*?R*
**then obtain** *E* **where** *E-def*: $A = (1 - l) *_R J + l *_R E$ *matrix-norm-bound E 1* $l \geq 0$
**by** *auto*
**have** *matrix-norm-bound* $(l *_R E)$ (*abs l*∗*1*)
**by** (*intro matrix-norm-bound-scale E-def*(*2*))
**moreover have** $l \geq 0$ **using** *E-def* **by** *simp*
**moreover have** $l *_R E = (A - (1 - l) *_R J)$
**using** *E-def*(*1*) **by** *simp*
**ultimately show** *matrix-norm-bound* $(A - (1 - l) *_R J)$ *l*
**by** *simp*
**qed**
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *hitting-property-alg*:
**fixes** *S* :: ($'n$ :: *finite*) *set*
**assumes** *l-range*: $l \in \{0..1\}$
**defines** $P \equiv$ *diag* (*ind-vec S*)
**defines** $\mu \equiv$ *card S* / *CARD*($'n$)
**assumes** $\bigwedge M.\ M \in$ *set Ms* $\Longrightarrow$ *spec-bound M l* $\wedge$ *markov M*
**shows** *foldl* ($\lambda x\ M.\ P *v\ (M *v\ x)$) ($P *v$ *stat*) *Ms* $\cdot$ *1* $\leq (\mu + l * (1-\mu))$^(*length Ms*+*1*)
**proof** −
**define** *t* :: *real*^$'n$ **where** $t = (\chi\ i.$ *of-bool* $(i \in S))$
**define** *r* **where** *r* = *foldl* ($\lambda x\ M.\ P *v\ (M *v\ x)$) ($P *v$ *stat*) *Ms*
**have** *P-proj*: $P ** P = P$
**unfolding** *P-def diag-mult-eq ind-vec-def* **by** (*intro arg-cong*[**where** *f*=*diag*]) (*vector*)

**have** *P-1-left*: *1 v*∗ *P* = *t*
**unfolding** *P-def diag-def ind-vec-def vector-matrix-mult-def t-def* **by** *simp*

**have** *P-1-right*: *P* ∗*v 1* = *t*
**unfolding** *P-def diag-def ind-vec-def matrix-vector-mult-def t-def* **by** *simp*

**have** *P-norm* :*matrix-norm-bound P 1*
**unfolding** *P-def ind-vec-def* **by** (*intro matrix-norm-bound-diag*) *simp*

**have** *norm-t*: *norm t* = *sqrt* (*real* (*card S*))
**unfolding** *t-def norm-vec-def L2-set-def of-bool-def*
**by** (*simp add*:*sum.If-cases if-distrib if-distribR*)

**have** $\mu$*-range*: $\mu \geq 0$ $\mu \leq 1$
**unfolding** $\mu$*-def* **by** (*auto simp add*:*card-mono*)

**define** *condition* :: *real*^$'n \Rightarrow$ *nat* $\Rightarrow$ *bool*
**where** *condition* = ($\lambda x\ n.$ *norm x* $\leq (\mu + l * (1-\mu))$^*n* ∗ *sqrt* (*card S*)/*CARD*($'n$) $\wedge$ *P* ∗*v x*
= *x*)

**have** *a*:*condition r* (*length Ms*)
**unfolding** *r-def* **using** *assms*(*4*)
**proof** (*induction Ms rule*:*rev-induct*)
**case** *Nil*
**have** *norm* (*P* ∗*v stat*) = (*1* / *real CARD*($'n$)) ∗ *norm t*
**unfolding** *stat-def matrix-vector-mult-scaleR P-1-right* **by** *simp*
**also have** ... $\leq$ (*1* / *real CARD*($'n$)) ∗ *sqrt* (*real* (*card S*))
**using** *norm-t* **by** (*intro mult-left-mono*) *auto*
**also have** ... = *sqrt* (*card S*)/*CARD*($'n$) **by** *simp*
**finally have** *norm* (*P* ∗*v stat*) $\leq$ *sqrt* (*card S*)/*CARD*($'n$) **by** *simp*
**moreover have** *P* ∗*v* (*P* ∗*v stat*) = *P* ∗*v stat*

**unfolding** *matrix-vector-mul-assoc P-proj* **by** *simp*
**ultimately show** *?case* **unfolding** *condition-def* **by** *simp*
**next**
**case** (*snoc M xs*)
**hence** *spec-bound M l* $\wedge$ *markov M*
**using** *snoc*(*2*) **by** *simp*
**then obtain** *E* **where** *E-def*: $M = (1-l) *_R J + l *_R E$ *matrix-norm-bound E 1*
**using** *iffD1*[*OF matrix-decomposition-lemma*] **by** *auto*

**define** *y* **where** $y = foldl\ (\lambda x\ M.\ P *v\ (M *v\ x))\ (P *v\ stat)\ xs$
**have** *b*:*condition y* (*length xs*)
**using** *snoc* **unfolding** *y-def* **by** *simp*
**hence** *a*:$P *v\ y = y$ **using** *condition-def* **by** *simp*

**have** $norm\ (P *v\ (M *v\ y)) = norm\ (P *v\ ((1-l) *_R J *v\ y) + P *v\ (l *_R E *v\ y))$
**by** (*simp add*:*E-def algebra-simps*)
**also have** ... $\leq norm\ (P *v\ ((1-l) *_R J *v\ y)) + norm\ (P *v\ (l *_R E *v\ y))$
**by** (*intro norm-triangle-ineq*)
**also have** ... $= (1 - l) * norm\ (P *v\ (J *v\ y)) + l * norm\ (P *v\ (E *v\ y))$
**using** *l-range*
**by** (*simp add*:*vector-scaleR-matrix-ac-2 matrix-vector-mult-scaleR*)
**also have** ... $= (1-l) * |1 \cdot (P *v\ y)/real\ CARD('n)| * norm\ t + l * norm\ (P *v\ (E *v\ y))$
**by** (*subst a*[*symmetric*])
(*simp add*:*apply-J proj-unit-def stat-def P-1-right matrix-vector-mult-scaleR*)
**also have** ... $= (1-l) * |t \cdot y|/real\ CARD('n) * norm\ t + l * norm\ (P *v\ (E *v\ y))$
**by** (*subst dot-lmul-matrix*[*symmetric*]) (*simp add*:*P-1-left*)
**also have** ... $\leq (1-l) * (norm\ t * norm\ y)\ /\ real\ CARD('n) * norm\ t + l * (1 * norm\ (E *v\ y))$
**using** *P-norm Cauchy-Schwarz-ineq2 l-range*
**by** (*intro add-mono mult-right-mono mult-left-mono divide-right-mono matrix-norm-boundD*) *auto*
**also have** ... $= (1-l) * \mu * norm\ y + l * norm\ (E *v\ y)$
**unfolding** *$\mu$-def norm-t* **by** *simp*
**also have** ... $\leq (1-l) * \mu * norm\ y + l * (1 * norm\ y)$
**using** *$\mu$-range l-range*
**by** (*intro add-mono matrix-norm-boundD mult-left-mono E-def*) *auto*
**also have** ... $= (\mu + l * (1-\mu)) * norm\ y$
**by** (*simp add*:*algebra-simps*)
**also have** ... $\leq (\mu + l * (1-\mu)) * ((\mu + l * (1-\mu))\hat{}length\ xs * sqrt\ (card\ S)/CARD('n))$
**using** *b $\mu$-range l-range* **unfolding** *condition-def*
**by** (*intro mult-left-mono*) *auto*
**also have** ... $= (\mu + l * (1-\mu))\hat{}(length\ xs + 1) * sqrt\ (card\ S)/CARD('n)$
**by** *simp*
**finally have** $norm\ (P *v\ (M *v\ y)) \leq (\mu + l * (1-\mu))\hat{}(length\ xs + 1) * sqrt\ (card\ S)/CARD('n)$
**by** *simp*

**moreover have** $P *v\ (P *v\ (M *v\ y)) = P *v\ (M *v\ y)$
**unfolding** *matrix-vector-mul-assoc matrix-mul-assoc P-proj*
**by** *simp*

**ultimately have** *condition* $(P *v\ (M *v\ y))$ (*length* (*xs*@[*M*]))
**unfolding** *condition-def* **by** *simp*

**then show** *?case*
**unfolding** *y-def* **by** *simp*
**qed**

```
  have inner r 1 = inner (P *v r) 1
    using a condition-def by simp
  also have ... = inner (1 v* P) r
    unfolding dot-lmul-matrix by (simp add:inner-commute)
  also have ... = inner t r
    unfolding P-1-left by simp
  also have ... ≤ norm t * norm r
    by (intro norm-cauchy-schwarz)
  also have ... ≤ sqrt (card S) * ((μ + l * (1−μ))^(length Ms) * sqrt(card S)/CARD('n))
    using a unfolding condition-def norm-t
    by (intro mult-mono) auto
  also have ... = (μ + 0) * ((μ + l * (1−μ))^(length Ms))
    by (simp add:μ-def)
  also have ... ≤ (μ + l * (1−μ)) * (μ + l * (1−μ))^(length Ms)
    using μ-range l-range
    by (intro mult-right-mono zero-le-power add-mono) auto
  also have ... = (μ + l * (1−μ))^(length Ms+1) by simp
  finally show ?thesis
    unfolding r-def by simp
qed

lemma upto-append:
  assumes i ≤ j j ≤ k
  shows  [i..<j]@[j..<k] = [i..<k]
  using assms by (metis less-eqE upt-add-eq-append)

definition bool-list-split :: bool list ⇒ (nat list × nat)
  where bool-list-split xs = foldl (λ(ys,z) x. (if x then (ys@[z],0) else (ys,z+1))) ([],0) xs

lemma bool-list-split:
  assumes bool-list-split xs = (ys,z)
  shows xs = concat (map (λk. replicate k False@[True]) ys)@replicate z False
  using assms
proof (induction xs arbitrary: ys z rule:rev-induct)
  case Nil
  then show ?case unfolding bool-list-split-def by simp
next
  case (snoc x xs)
  obtain u v where uv-def: bool-list-split xs = (u,v)
    by (metis surj-pair)

  show ?case
  proof (cases x)
    case True
    have a:ys = u@[v] z = 0
      using snoc(2) True uv-def unfolding bool-list-split-def by auto
    have xs@[x] = concat (map (λk. replicate k False@[True]) u)@replicate v False@[True]
      using snoc(1)[OF uv-def] True by simp
    also have ... = concat (map (λk. replicate k False@[True]) (u@[v]))@replicate 0 False
      by simp
    also have ... = concat (map (λk. replicate k False@[True]) (ys))@replicate z False
      using a by simp
    finally show ?thesis by simp
  next
    case False
    have a:ys = u z = v+1
      using snoc(2) False uv-def unfolding bool-list-split-def by auto
    have xs@[x] = concat (map (λk. replicate k False@[True]) u)@replicate (v+1) False
```

```
      using snoc(1)[OF uv-def] False unfolding replicate-add by simp
    also have ... = concat (map (λk. replicate k False@[True]) (ys))@replicate z False
      using a by simp
    finally show ?thesis by simp
  qed
qed

lemma bool-list-split-count:
  assumes bool-list-split xs = (ys,z)
  shows length (filter id xs) = length ys
  unfolding bool-list-split[OF assms(1)] by (simp add:filter-concat comp-def)

lemma foldl-concat:
  foldl f a (concat xss) = foldl (λy xs. foldl f y xs) a xss
  by (induction xss rule:rev-induct, auto)

lemma hitting-property-alg-2:
  fixes S :: ('n :: finite) set and l :: nat
  fixes M :: real^'n^'n
  assumes α-range: α ∈ {0..1}
  assumes I ⊆ {..<l}
  defines P i ≡ (if i ∈ I then diag (ind-vec S) else mat 1)
  defines μ ≡ real (card S) / real (CARD('n))
  assumes spec-bound M α markov M
  shows
    foldl (λx M. M *v x) stat (intersperse M (map P [0..<l])) · 1 ≤ (μ+α*(1−μ))^card I
    (is ?L ≤ ?R)
proof (cases I ≠ {})
  case True
  define xs where xs = map (λi. i ∈ I) [0..<l]
  define Q where Q = diag (ind-vec S)
  define P' where P' = (λx. if x then Q else mat 1)

  let ?rep = (λx. replicate x (mat 1))

  have P-eq: P i = P' (i ∈ I) for i
    unfolding P-def P'-def Q-def by simp

  have l > 0
    using True assms(2) by auto
  hence xs-ne: xs ≠ []
    unfolding xs-def by simp

  obtain ys z where ys-z: bool-list-split xs = (ys,z)
    by (metis surj-pair)


  have length ys = length (filter id xs)
    using bool-list-split-count[OF ys-z] by simp
  also have ... = card (I ∩ {0..<l})
    unfolding xs-def filter-map by (simp add:comp-def distinct-length-filter)
  also have ... = card I
    using Int-absorb2[OF assms(2)] unfolding atLeast0LessThan by simp
  finally have  len-ys: length ys = card I by simp

  hence length ys > 0
    using True assms(2) by (metis card-gt-0-iff finite-nat-iff-bounded)
  then obtain yh yt where ys-split: ys = yh#yt
```

**by** (metis length-greater-0-conv neq-Nil-conv)

**have** a:foldl (λx N. M *v (N *v x)) x (?rep z) · 1 = x · 1 **for** x
**proof** (induction z)
**case** 0
**then show** ?case **by** simp
**next**
**case** (Suc z)
**have** foldl (λx N. M *v (N *v x)) x (?rep (z+1)) · 1 = x · 1
**unfolding** replicate-add **using** Suc
**by** (simp add:markov-orth-inv[OF assms(6)])
**then show** ?case **by** simp
**qed**

**have** M *v stat = stat
**using** assms(6) **unfolding** stat-def matrix-vector-mult-scaleR markov-def **by** simp
**hence** b: foldl (λx N. M *v (N *v x)) stat (?rep yh) = stat
**by** (induction yh, auto)

**have** foldl (λx N. N *v (M *v x)) a (?rep x) = matrix-pow M x *v a **for** x a
**proof** (induction x)
**case** 0
**then show** ?case **by** simp
**next**
**case** (Suc x)
**have** foldl (λx N. N *v (M *v x)) a (?rep (x+1)) = matrix-pow M (x+1) *v a
**unfolding** replicate-add **using** Suc **by** (simp add: matrix-vector-mul-assoc)
**then show** ?case **by** simp
**qed**
**hence** c: foldl (λx N. N *v (M *v x)) a (?rep x @ [Q]) = Q *v (matrix-pow M (x+1) *v a)
**for** x a
**by** (simp add:matrix-vector-mul-assoc matrix-mul-assoc)

**have** d: spec-bound N α ∧ markov N **if** t1: N ∈ set (map (λx. matrix-pow M (x + 1)) yt) **for**
N
**proof** −
**obtain** y **where** N-def: N = matrix-pow M (y+1)
**using** t1 **by** auto
**hence** d1: spec-bound N (α^(y+1))
**unfolding** N-def **using** spec-bound-pow assms(5,6) **by** blast
**have** spec-bound N (α^1)
**using** α-range **by** (intro spec-bound-mono[OF d1] power-decreasing) auto
**moreover have** markov N
**unfolding** N-def **by** (intro markov-matrix-pow assms(6))
**ultimately show** ?thesis **by** simp
**qed**

**have** ?L = foldl (λx M. M *v x) stat (intersperse M (map P′ xs)) · 1
**unfolding** P-eq xs-def map-map **by** (simp add:comp-def)
**also have** ... = foldl (λx M. M *v x) stat (intersperse M (map P′ xs)@[M]) · 1
**by** (simp add:markov-orth-inv[OF assms(6)])
**also have** ... = foldl (λx N. M *v (N *v x)) stat (map P′ xs) · 1
**using** xs-ne **by** (subst foldl-intersperse) auto
**also have** ... = foldl (λx N. M *v (N *v x)) stat ((ys ≫= (λx. ?rep x @ [Q])) @ ?rep z) · 1
**unfolding** bool-list-split[OF ys-z] P′-def List.bind-def **by** (simp add: comp-def map-concat)
**also have** ... = foldl (λx N. M *v (N *v x)) stat (ys ≫= (λx. ?rep x @ [Q])) · 1
**by** (simp add: a)
**also have** ... = foldl (λx N. M *v (N *v x)) stat (?rep yh @[Q]@(yt ≫=(λx. ?rep x @ [Q]))) · 1

```
    unfolding ys-split by simp
  also have ... = foldl (λx N. M *v (N *v x)) stat ([Q]@(yt ≫=(λx. ?rep x @ [Q]))) · 1
    by (simp add:b)
  also have ... = foldl (λx N. N *v x) stat (intersperse M (Q#(yt ≫=(λx.?rep x@[Q])))@[M])·1
    by (subst foldl-intersperse, auto)
  also have ... = foldl (λx N. N *v x) stat (intersperse M (Q#(yt ≫=(λx.?rep x@[Q])))) · 1
    by (simp add:markov-orth-inv[OF assms(6)])
  also have ... = foldl (λx N. N *v (M *v x)) (Q *v stat) (yt ≫=(λx.?rep x@[Q])) · 1
    by (subst foldl-intersperse-2, simp)
  also have ... = foldl (λa x. foldl (λx N. N *v (M *v x)) a (?rep x @ [Q])) (Q *v stat) yt · 1
    unfolding List.bind-def foldl-concat foldl-map by simp
  also have ... = foldl (λa x. Q *v (matrix-pow M (x+1) *v a)) (Q *v stat) yt · 1
    unfolding c by simp
  also have ... = foldl (λa N. Q *v (N *v a)) (Q *v stat) (map (λx. matrix-pow M (x+1)) yt) ·
1
    by (simp add:foldl-map)
  also have ... ≤ (μ + α*(1−μ))^(length (map (λx. matrix-pow M (x+1)) yt)+1)
    unfolding μ-def Q-def by (intro hitting-property-alg α-range d) simp
  also have ... = (μ + α*(1−μ))^(length ys)
    unfolding ys-split by simp
  also have ... = ?R unfolding len-ys by simp
  finally show ?thesis by simp
next
  case False
  hence I-empty: I = {} by simp

  have ?L = stat · (1 :: real^'n)
  proof (cases l > 0)
    case True
    have ?L = foldl (λx M. M *v x) stat ((intersperse M (map P [0..<l]))@[M]) · 1
      by (simp add:markov-orth-inv[OF assms(6)])
    also have ... = foldl (λx N. M *v (N *v x)) stat (map P [0..<l])  · 1
      using True by (subst foldl-intersperse, auto)
    also have ... = foldl (λx N. M *v (N *v x)) stat (map (λ-. mat 1) [0..<l])  · 1
      unfolding  P-def using I-empty by simp
    also have ... = foldl (λx -. M *v x) stat [0..<l] · 1
      unfolding foldl-map by simp
    also have ... = stat · (1 :: real^'n)
      by (induction l, auto simp add:markov-orth-inv[OF assms(6)])
    finally show ?thesis by simp
  next
    case False
    then show ?thesis by simp
  qed
  also have ... = 1
    unfolding stat-def by (simp add:inner-vec-def)
  also have ... ≤ ?R unfolding I-empty by simp
  finally show ?thesis by simp
qed

lemma uniform-property-alg:
  fixes x :: ('n :: finite) and l :: nat
  assumes i < l
  defines P j ≡ (if j = i then diag (ind-vec {x}) else mat 1)
  assumes markov M
  shows foldl (λx M. M *v x) stat (intersperse M (map P [0..<l])) · 1 = 1 / CARD('n)
    (is ?L = ?R)
proof −
```

**have** *a*: *l* > *0* **using** *assms*(*1*) **by** *simp*

**have** *0*: *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *y* (*xs*) · *1* = *y* · *1* **if** *set xs* ⊆ {*mat 1*} **for** *xs y*
**using** *that*
**proof** (*induction xs rule*:*rev-induct*)
**case** *Nil*
**then show** *?case* **by** *simp*
**next**
**case** (*snoc x xs*)
**have** *x* = *mat 1*
**using** *snoc*(*2*) **by** *simp*
**hence** *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *y* (*xs* @ [*x*]) · *1* = *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *y xs* · *1*
**by** (*simp add*:*markov-orth-inv*[*OF assms*(*3*)])
**also have** ... = *y* · *1*
**using** *snoc*(*2*) **by** (*intro snoc*(*1*)) *auto*
**finally show** *?case* **by** *simp*
**qed**

**have** *M-stat*: *M* ∗*v stat* = *stat*
**using** *assms*(*3*) **unfolding** *stat-def matrix-vector-mult-scaleR markov-def* **by** *simp*

**hence** *1*: (*foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *stat xs*) = *stat* **if** *set xs* ⊆ {*mat 1*} **for** *xs*
**using** *that* **by** (*induction xs*, *auto*)

**have** *?L* = *foldl* (λ*x M*. *M* ∗*v x*) *stat* ((*intersperse M* (*map P* [*0*..<*l*]))@[*M*]) · *1*
**by** (*simp add*:*markov-orth-inv*[*OF assms*(*3*)])
**also have** ... = *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *stat* (*map P* [*0*..<*l*]) · *1*
**using** *a* **by** (*subst foldl-intersperse*) *auto*
**also have** ... = *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *stat* (*map P* ([*0*..<*i*+*1*]@[*i*+*1*..<*l*])) · *1*
**using** *assms*(*1*) **by** (*subst upto-append*) *auto*
**also have** ... = *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *stat* (*map P* [*0*..<*i* + *1*]) · *1*
**unfolding** *map-append foldl-append P-def* **by** (*subst 0*) *auto*
**also have** ... = *foldl* (λ*x N*. *M* ∗*v* (*N* ∗*v x*)) *stat* (*map P* ([*0*..<*i*]@[*i*])) · *1*
**by** *simp*
**also have** ... = (*M* ∗*v* (*diag* (*ind-vec* {*x*}) ∗*v stat*)) · *1*
**unfolding** *map-append foldl-append P-def* **by** (*subst 1*) *auto*
**also have** ... = (*diag* (*ind-vec* {*x*}) ∗*v stat*) · *1*
**by** (*simp add*:*markov-orth-inv*[*OF assms*(*3*)])
**also have** ... = ((*1*/*CARD*(′*n*)) $*_R$ *ind-vec* {*x*}) · *1*
**unfolding** *diag-def ind-vec-def stat-def matrix-vector-mult-def*
**by** (*intro arg-cong2*[**where** *f*=(·)] *refl*)
(*vector of-bool-def sum.If-cases if-distrib if-distribR*)
**also have** ... = (*1*/*CARD*(′*n*)) ∗ (*ind-vec* {*x*} · *1*)
**by** *simp*
**also have** ... = (*1*/*CARD*(′*n*)) ∗ *1*
**unfolding** *inner-vec-def ind-vec-def of-bool-def*
**by** (*intro arg-cong2*[**where** *f*=(∗)] *refl*) (*simp*)
**finally show** *?thesis* **by** *simp*
**qed**

**end**

**lemma** *foldl-matrix-mult-expand*:
**fixes** *Ms* :: ((′*r*::{*semiring-1*,*comm-monoid-mult*})^′*a*^′*a*) *list*
**shows** (*foldl* (λ*x M*. *M* ∗*v x*) *a Ms*) $ *k* = (∑ *x* | *length x* = *length Ms*+*1* ∧ *x*! *length Ms* = *k*.
(∏ *i*< *length Ms*. (*Ms* ! *i*) $ (*x* ! (*i*+*1*)) $ (*x* ! *i*)) ∗ *a* $ (*x* ! *0*))
**proof** (*induction Ms arbitrary*: *k rule*:*rev-induct*)
**case** *Nil*

**have** *length x* = *Suc 0* ⟹ *x* = [*x*!*0*] **for** *x* :: *'a list*
**by** (*cases x*, *auto*)
**hence** {*x*. *length x* = *Suc 0* ∧ *x* ! *0* = *k*} = {[*k*]}
**by** *auto*
**thus** *?case* **by** *auto*
**next**
**case** (*snoc M Ms*)
**let** *?l* = *length Ms*

**have** *0*: *finite* {*w*. *length w* = *Suc* (*length Ms*) ∧ *w* ! *length Ms* = *i*} **for** *i* :: *'a*
**using** *finite-lists-length-eq*[**where** *A*=*UNIV*::*'a set* **and** *n*=*?l* +*1*] **by** *simp*

**have** *take* (*?l*+*1*) *x* @ [*x* ! (*?l*+*1*)] = *x* **if** *length x* = *?l*+*2* **for** *x* :: *'a list*
**proof** −
**have** *take* (*?l*+*1*) *x* @ [*x* ! (*?l*+*1*)] = *take* (*Suc* (*?l*+*1*)) *x*
**using** *that* **by** (*intro take-Suc-conv-app-nth*[*symmetric*], *simp*)
**also have** ... = *x*
**using** *that* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**
**hence** *1*: *bij-betw* (*take* (*?l*+*1*)) {*w*. *length w*=*?l*+*2* ∧ *w*!(*?l*+*1*) =*k*} {*w*. *length w* = *?l*+*1*}
**by** (*intro bij-betwI*[**where** *g*=*λx*. *x*@[*k*]]) (*auto simp add*:*nth-append*)

**have** *foldl* (*λx M*. *M* ∗*v x*) *a* (*Ms* @ [*M*]) $ *k* = (∑*j*∈*UNIV*. *M*$*k*$*j* ∗(*foldl* (*λx M*. *M* ∗*v x*) *a* *Ms* $ *j*))
**by** (*simp add*:*matrix-vector-mult-def*)
**also have** ... =
(∑*j*∈*UNIV*. *M*$*k*$*j* ∗ (∑*w*|*length w*=*?l*+*1*∧*w*!*?l*=*j*. (∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*) ∗ *a* $ *w*!*0*))
**unfolding** *snoc* **by** *simp*
**also have** ... =
(∑*j*∈*UNIV*. (∑*w*|*length w*=*?l*+*1*∧*w*!*?l*=*j*. *M*$*k*$*w*!*?l* ∗ (∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*) ∗ *a* $ *w*!*0*))
**by** (*intro sum.cong refl*) (*simp add*: *sum-distrib-left algebra-simps*)
**also have** ... = (∑*w*∈ (⋃*j* ∈ *UNIV*. {*w*. *length w*=*?l*+*1* ∧ *w*!*?l* =*j*}).
*M*$*k*$*w*!*?l*∗(∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*) ∗ *a* $ *w*!*0*)
**using** *0* **by** (*subst sum.UNION-disjoint*, *simp*, *simp*) *auto*
**also have** ... = (∑*w* | *length w*=*?l*+*1*. *M*$*k*$(*w*!*?l*)∗(∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*) ∗ *a* $ *w*!*0*)
**by** (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *refl*) *auto*
**also have** ... = (∑*w* ∈ *take* (*?l*+*1*) ‘ {*w*. *length w*=*?l*+*2* ∧ *w*!(*?l*+*1*) =*k*}.
*M*$*k*$*w*!*?l*∗(∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*) ∗ *a* $ *w*!*0*)
**using** *1* **unfolding** *bij-betw-def* **by** (*intro sum.cong refl*, *auto*)
**also have** ... = (∑*w*|*length w*=*?l*+*2*∧*w*!(*?l*+*1*)=*k*. *M*$*k*$*w*!*?l*∗(∏*i*<*?l*. *Ms*!*i* $ *w*!(*i*+*1*) $ *w*!*i*)∗ *a*$*w*!*0*)
**using** *1* **unfolding** *bij-betw-def* **by** (*subst sum.reindex*, *auto*)
**also have** ... = (∑*w*|*length w*=*?l*+*2*∧*w*!(*?l*+*1*)=*k*.
(*Ms*@[*M*])!*?l*$*k*$*w*!*?l*∗(∏*i*<*?l*. (*Ms*@[*M*])!*i* $ *w*!(*i*+*1*) $ *w*!*i*)∗ *a*$*w*!*0*)
**by** (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *prod.cong refl*) (*auto simp add*:*nth-append*)
**also have** ... = (∑*w*|*length w*=*?l*+*2*∧*w*!(*?l*+*1*)=*k*. (∏*i*<(*?l*+*1*). (*Ms*@[*M*])!*i* $ *w*!(*i*+*1*) $ *w*!*i*)∗ *a*$*w*!*0*)
**by** (*intro sum.cong*, *auto simp add*:*algebra-simps*)
**finally have** *foldl* (*λx M*. *M* ∗*v x*) *a* (*Ms* @ [*M*]) $ *k* =
(∑ *w* | *length w* = *?l*+*2* ∧ *w* ! (*?l*+*1*) = *k*. (∏*i*<(*?l*+*1*). (*Ms*@[*M*])!*i* $ *w*!(*i*+*1*) $ *w*!*i*)∗ *a*$*w*!*0*)
**by** *simp*
**then show** *?case* **by** *simp*
**qed**

**lemma** *foldl-matrix-mult-expand-2*:
**fixes** *Ms* :: (*real*^'*a*^'*a*) *list*
**shows** (*foldl* ($\lambda$*x M*. *M* $*v$ *x*) *a Ms*) $\cdot$ *1* = ($\sum$ *x* | *length x* = *length Ms*+*1*.
($\prod$ *i*< *length Ms*. (*Ms* ! *i*) \$ (*x* ! (*i*+*1*)) \$ (*x* ! *i*)) $*$ *a* \$ (*x* ! *0*))
(**is** *?L* = *?R*)
**proof** −
**let** *?l* = *length Ms*
**have** *?L* = ($\sum$ *j* $\in$ *UNIV*. (*foldl* ($\lambda$*x M*. *M* $*v$ *x*) *a Ms*) \$ *j*)
**by** (*simp add:inner-vec-def*)
**also have** ... = ($\sum$ *j*$\in$*UNIV*. $\sum$ *x*|*length x*=*?l*+*1* $\wedge$ *x*!*?l*=*j*.($\prod$ *i*<*?l*. *Ms*!*i* \$ *x*!(*i*+*1*) \$ *x*!*i*) $*$ *a* \$ *x*!*0*)
**unfolding** *foldl-matrix-mult-expand* **by** *simp*
**also have** ... = ($\sum$ *x* $\in$ ($\bigcup$ *j*$\in$ *UNIV*.{*w*. *length w* = *length Ms*+*1* $\wedge$ *w* ! *length Ms* = *j*}).
($\prod$ *i*< *length Ms*. (*Ms* ! *i*) \$ (*x* ! (*i*+*1*)) \$ (*x* ! *i*)) $*$ *a* \$ (*x* ! *0*))
**using** *finite-lists-length-eq*[**where** *A*=*UNIV*::'*a set* **and** *n*=*?l* +*1*]
**by** (*intro sum.UNION-disjoint*[*symmetric*]) *auto*
**also have** ... = *?R*
**by** (*intro sum.cong*, *auto*)
**finally show** *?thesis* **by** *simp*
**qed**

**end**

# 6 Spectral Theory

This section establishes the correspondence of the variationally defined expansion paramters with the definitions using the spectrum of the stochastic matrix. Additionally stronger results for the expansion parameters are derived.

**theory** *Expander-Graphs-Eigenvalues*
**imports**
*Expander-Graphs-Algebra*
*Expander-Graphs-TTS*
*Perron-Frobenius.HMA-Connect*
*Commuting-Hermitian.Commuting-Hermitian*
**begin**

**unbundle** *intro-cong-syntax*

**hide-const** *Matrix-Legacy.transpose*
**hide-const** *Matrix-Legacy.row*
**hide-const** *Matrix-Legacy.mat*
**hide-const** *Matrix.mat*
**hide-const** *Matrix.row*
**hide-fact** *Matrix-Legacy.row-def*
**hide-fact** *Matrix-Legacy.mat-def*
**hide-fact** *Matrix.vec-eq-iff*
**hide-fact** *Matrix.mat-def*
**hide-fact** *Matrix.row-def*
**no-notation** *Matrix.scalar-prod* (**infix** ‹·› *70*)
**no-notation** *Ordered-Semiring.max* (‹*Max*ı›)

**lemma** *mult-right-mono'*: $y \geq (0{::}real) \Longrightarrow x \leq z \vee y = 0 \Longrightarrow x * y \leq z * y$
**by** (*metis mult-cancel-right mult-right-mono*)

**lemma** *poly-prod-zero*:
**fixes** *x* :: '*a* :: *idom*
**assumes** *poly* ($\prod$ *a*$\in$#*xs*. [:− *a*, *1*:]) *x* = *0*

```
  shows x ∈# xs
  using assms by (induction xs, auto)

lemma poly-prod-inj-aux-1:
  fixes xs ys :: ('a :: idom) multiset
  assumes x ∈# xs
  assumes (∏ a∈#xs. [:− a, 1:]) = (∏ a∈#ys. [:− a, 1:])
  shows x ∈# ys
proof −
  have poly (∏ a∈#ys. [:− a, 1:]) x = poly (∏ a∈#xs. [:− a, 1:]) x using assms(2) by simp
  also have ... = poly (∏ a∈#xs − {#x#} + {#x#}. [:− a, 1:]) x
    using assms(1) by simp
  also have ... = 0
    by simp
  finally have poly (∏ a∈#ys. [:− a, 1:]) x = 0 by simp
  thus x ∈# ys using poly-prod-zero by blast
qed

lemma poly-prod-inj-aux-2:
  fixes xs ys :: ('a :: idom) multiset
  assumes x ∈# xs ∪# ys
  assumes (∏ a∈#xs. [:− a, 1:]) = (∏ a∈#ys. [:− a, 1:])
  shows x ∈# xs ∩# ys
proof (cases x ∈# xs)
  case True
  then show ?thesis using poly-prod-inj-aux-1[OF True assms(2)] by simp
next
  case False
  hence a:x ∈# ys
    using assms(1) by simp
  then show ?thesis
    using poly-prod-inj-aux-1[OF a assms(2)[symmetric]] by simp
qed

lemma poly-prod-inj:
  fixes xs ys :: ('a :: idom) multiset
  assumes (∏ a∈#xs. [:− a, 1:]) = (∏ a∈#ys. [:− a, 1:])
  shows xs = ys
  using assms
proof (induction size xs + size ys arbitrary: xs ys rule:nat-less-induct)
  case 1
  show ?case
  proof (cases xs ∪# ys = {#})
    case True
    then show ?thesis by simp
  next
    case False
    then obtain x where x ∈# xs ∪# ys by auto
    hence a:x ∈# xs ∩# ys
      by (intro poly-prod-inj-aux-2[OF - 1(2)])
    have b: [:− x, 1:] ≠ 0
      by simp
    have c: size (xs−{#x#}) + size (ys−{#x#}) < size xs + size ys
      using a by (simp add: add-less-le-mono size-Diff1-le size-Diff1-less)

    have [:− x, 1:] ∗ (∏ a∈#xs − {#x#}. [:− a, 1:]) = (∏ a∈#xs. [:− a, 1:])
      using a by (subst prod-mset.insert[symmetric]) simp
    also have ... = (∏ a∈#ys. [:− a, 1:]) using 1 by simp
```

**also have** ... = [:− *x*, *1*:] ∗ (∏ *a*∈#*ys* − {#*x*#}. [:− *a*, *1*:])
**using** *a* **by** (*subst prod-mset.insert*[*symmetric*]) *simp*
**finally have** [:− *x*, *1*:]∗(∏ *a*∈#*xs*−{#*x*#}. [:− *a*, *1*:])=[:−*x*, *1*:]∗(∏ *a*∈#*ys*−{#*x*#}. [:− *a*, *1*:])
**by** *simp*
**hence** (∏ *a*∈#*xs*−{#*x*#}. [:− *a*, *1*:]) = (∏ *a*∈#*ys*−{#*x*#}. [:− *a*, *1*:])
**using** *mult-left-cancel*[*OF b*] **by** *simp*
**hence** *d*:*xs* − {#*x*#} = *ys* − {#*x*#}
**using** *1 c* **by** *simp*
**have** *xs* = *xs* − {#*x*#} + {#*x*#}
**using** *a* **by** *simp*
**also have** ... = *ys* − {#*x*#} + {#*x*#}
**unfolding** *d* **by** *simp*
**also have** ... = *ys*
**using** *a* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**
**qed**

**definition** *eigenvalues* :: (′*a*::*comm-ring-1*)^′*n*^′*n* ⇒ ′*a multiset*
**where**
*eigenvalues A* = (*SOME as*. *charpoly A* = (∏ *a*∈#*as*. [:− *a*, *1*:]) ∧ *size as* = *CARD* (′*n*))

**lemma** *char-poly-factorized-hma*:
**fixes** *A* :: *complex*^′*n*^′*n*
**shows** ∃ *as*. *charpoly A* = (∏ *a*←*as*. [:− *a*, *1*:]) ∧ *length as* = *CARD* (′*n*)
**by** (*transfer-hma rule*:*char-poly-factorized*)

**lemma** *eigvals-poly-length*:
**fixes** *A* :: *complex*^′*n*^′*n*
**shows**
*charpoly A* = (∏ *a*∈#*eigenvalues A*. [:− *a*, *1*:]) (**is** *?A*)
*size* (*eigenvalues A*) = *CARD* (′*n*) (**is** *?B*)
**proof** −
**define** *f* **where** *f as* = (*charpoly A* = (∏ *a*∈#*as*. [:− *a*, *1*:]) ∧ *size as* = *CARD*(′*n*)) **for** *as*
**obtain** *as* **where** *as-def*: *charpoly A* = (∏ *a*←*as*. [:− *a*, *1*:]) *length as* = *CARD*(′*n*)
**using** *char-poly-factorized-hma* **by** *auto*

**have** *charpoly A* = (∏ *a*←*as*. [:− *a*, *1*:])
**unfolding** *as-def* **by** *simp*
**also have** ... = (∏ *a*∈#*mset as*. [:− *a*, *1*:])
**unfolding** *prod-mset-prod-list*[*symmetric*] *mset-map* **by** *simp*
**finally have** *charpoly A* = (∏ *a*∈#*mset as*. [:− *a*, *1*:]) **by** *simp*
**moreover have** *size* (*mset as*) = *CARD*(′*n*)
**using** *as-def* **by** *simp*
**ultimately have** *f* (*mset as*)
**unfolding** *f-def* **by** *auto*
**hence** *f* (*eigenvalues A*)
**unfolding** *eigenvalues-def f-def*[*symmetric*] **using** *someI*[**where** *x* = *mset as* **and** *P*=*f*] **by** *auto*
**thus** *?A ?B*
**unfolding** *f-def* **by** *auto*
**qed**

**lemma** *similar-matrix-eigvals*:
**fixes** *A B* :: *complex*^′*n*^′*n*
**assumes** *similar-matrix A B*
**shows** *eigenvalues A* = *eigenvalues B*

**proof** −
  **have** ($\prod a{\in}\#eigenvalues\ A.\ [:-\ a,\ 1:]) = (\prod a{\in}\#eigenvalues\ B.\ [:-\ a,\ 1:])$
    **using** *similar-matrix-charpoly*[*OF assms*] **unfolding** *eigvals-poly-length*(*1*) **by** *simp*
  **thus** *?thesis*
    **by** (*intro poly-prod-inj*) *simp*
**qed**

**definition** *upper-triangular-hma* :: $'a::zero^{\frown}'n^{\frown}'n \Rightarrow bool$
  **where** *upper-triangular-hma A* ≡
    $\forall i.\ \forall\ j.\ (to\text{-}nat\ j < Bij\text{-}Nat.to\text{-}nat\ i \longrightarrow A\ \$h\ i\ \$h\ j = 0)$

**lemma** *for-all-reindex2*:
  **assumes** *range f = A*
  **shows** $(\forall x \in A.\ \forall y \in A.\ P\ x\ y) \longleftrightarrow (\forall x\ y.\ P\ (f\ x)\ (f\ y))$
  **using** *assms* **by** *auto*

**lemma** *upper-triangular-hma*:
  **fixes** $A :: ('a::zero)^{\frown}'n^{\frown}'n$
  **shows** *upper-triangular* (*from-hma$_m$ A*) = *upper-triangular-hma A* (**is** *?L = ?R*)
**proof** −
   **have** $?L \longleftrightarrow (\forall i{\in}\{0..{<}CARD('n)\}.\ \forall j{\in}\{0..{<}CARD('n)\}.\ j < i \longrightarrow A\ \$h\ from\text{-}nat\ i\ \$h\ from\text{-}nat\ j = 0)$
    **unfolding** *upper-triangular-def from-hma$_m$-def* **by** *auto*
  **also have** ... $\longleftrightarrow (\forall (i::'n)\ (j::'n).\ to\text{-}nat\ j < to\text{-}nat\ i \longrightarrow A\ \$h\ from\text{-}nat\ (to\text{-}nat\ i)\ \$h\ from\text{-}nat\ (to\text{-}nat\ j) = 0)$
    **by** (*intro for-all-reindex2 range-to-nat*[**where** *'a='n*])
  **also have** ... ⟷ *?R*
    **unfolding** *upper-triangular-hma-def* **by** *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *from-hma-carrier*:
  **fixes** $A :: 'a^{\frown}('n::finite)^{\frown}('m::finite)$
  **shows** *from-hma$_m$ A* ∈ *carrier-mat* (*CARD* (*'m*)) (*CARD* (*'n*))
  **unfolding** *from-hma$_m$-def* **by** *simp*

**definition** *diag-mat-hma* :: $'a^{\frown}'n^{\frown}'n \Rightarrow 'a\ multiset$
  **where** *diag-mat-hma A* = *image-mset* ($\lambda i.\ A\ \$h\ i\ \$h\ i$) (*mset-set UNIV*)

**lemma** *diag-mat-hma*:
  **fixes** $A :: 'a^{\frown}'n^{\frown}'n$
  **shows**  *mset* (*diag-mat* (*from-hma$_m$ A*)) = *diag-mat-hma A* (**is** *?L = ?R*)
**proof** −
  **have** $?L = \{\#from\text{-}hma_m\ A\ \$\$\ (i,\ i).\ i \in\#\ mset\ [0..{<}CARD('n)]\#\}$
    **using** *from-hma-carrier*[**where** *A=A*] **unfolding** *diag-mat-def mset-map* **by** *simp*
  **also have** ... = $\{\#from\text{-}hma_m\ A\ \$\$\ (i,\ i).\ i \in\#\ image\text{-}mset\ to\text{-}nat\ (mset\text{-}set\ (UNIV :: 'n\ set))\#\}$
    **using** *range-to-nat*[**where** *'a='n*]
   **by** (*intro arg-cong2*[**where** *f=image-mset*] *refl*) (*simp add:image-mset-mset-set*[*OF inj-to-nat*])
  **also have** ... = $\{\#from\text{-}hma_m\ A\ \$\$\ (to\text{-}nat\ i,\ to\text{-}nat\ i).\ i \in\#\ (mset\text{-}set\ (UNIV :: 'n\ set))\#\}$
    **by** (*simp add:image-mset.compositionality comp-def*)
  **also have** ... = *?R*
    **unfolding** *diag-mat-hma-def from-hma$_m$-def* **using** *to-nat-less-card*[**where** *'a='n*]
    **by** (*intro image-mset-cong*) *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**definition** *adjoint-hma* :: $complex^{\frown}'m^{\frown}'n \Rightarrow complex^{\frown}'n^{\frown}'m$ **where**
  *adjoint-hma A* = *map-matrix cnj* (*transpose A*)

**lemma** *adjoint-hma-eq*: *adjoint-hma A* $h *i* $h *j* = *cnj* (*A* $h *j* $h *i*)
  **unfolding** *adjoint-hma-def map-matrix-def map-vector-def transpose-def* **by** *auto*

**lemma** *adjoint-hma*:
  **fixes** *A* :: *complex*^('*n*::*finite*)^('*m*::*finite*)
  **shows** *mat-adjoint* (*from-hma*$_m$ *A*) = *from-hma*$_m$ (*adjoint-hma A*)
**proof** −
  **have** *mat-adjoint* (*from-hma*$_m$ *A*) $$ (*i*,*j*) = *from-hma*$_m$ (*adjoint-hma A*) $$ (*i*,*j*)
    **if** *i* < *CARD*('*n*) *j* < *CARD*('*m*) **for** *i j*
    **using** *from-hma-carrier that* **unfolding** *mat-adjoint-def from-hma*$_m$*-def adjoint-hma-def*
      *Matrix.mat-of-rows-def map-matrix-def map-vector-def transpose-def* **by** *auto*
  **thus** *?thesis*
    **using** *from-hma-carrier*
    **by** (*intro eq-matI*) *auto*
**qed**

**definition** *cinner* **where** *cinner v w* = *scalar-product v* (*map-vector cnj w*)

**context**
  **includes** *lifting-syntax*
**begin**

**lemma** *cinner-hma*:
  **fixes** *x y* :: *complex*^'*n*
  **shows** *cinner x y* = (*from-hma*$_v$ *x*) ·*c* (*from-hma*$_v$ *y*) (**is** *?L* = *?R*)
**proof** −
  **have** *?L* = ($\sum$ *i*∈*UNIV*. *x* $h *i* ∗ *cnj* (*y* $h *i*))
    **unfolding** *cinner-def map-vector-def scalar-product-def* **by** *simp*
  **also have** ... = ($\sum$ *i* = *0*..<*CARD*('*n*). *x* $h *from-nat i* ∗ *cnj* (*y* $h *from-nat i*))
    **using** *to-nat-less-card to-nat-from-nat-id*
    **by** (*intro sum.reindex-bij-betw*[*symmetric*] *bij-betwI*[**where** *g*=*to-nat*]) *auto*
  **also have** ... = *?R*
    **unfolding** *Matrix.scalar-prod-def from-hma*$_v$*-def*
    **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *cinner-hma-transfer*[*transfer-rule*]:
  (*HMA-V* ===> *HMA-V* ===> (=)) (·*c*) *cinner*
  **unfolding** *HMA-V-def cinner-hma*
  **by** (*auto simp*:*rel-fun-def*)

**lemma** *adjoint-hma-transfer*[*transfer-rule*]:
  (*HMA-M* ===> *HMA-M*) (*mat-adjoint*) *adjoint-hma*
  **unfolding** *HMA-M-def rel-fun-def* **by** (*auto simp add*:*adjoint-hma*)

**end**

**lemma** *adjoint-adjoint-id*[*simp*]: *adjoint-hma* (*adjoint-hma A* ) = *A*
  **by** (*transfer*) (*simp add*:*adjoint-adjoint*)

**lemma** *adjoint-def-alter-hma*:
  *cinner* (*A* ∗*v v*) *w* = *cinner v* (*adjoint-hma A* ∗*v w*)
  **by** (*transfer-hma rule*:*adjoint-def-alter*)

**lemma** *cinner-0*: *cinner 0 0* = *0*
  **by** (*transfer-hma*)

**lemma** *cinner-scale-left*: *cinner* $(a *s\ v)\ w = a * cinner\ v\ w$
**by** *transfer-hma*

**lemma** *cinner-scale-right*: *cinner* $v\ (a *s\ w) = cnj\ a * cinner\ v\ w$
**by** *transfer* (*simp add*: *inner-prod-smult-right*)

**lemma** *norm-of-real*:
**shows** *norm* (*map-vector complex-of-real v*) = *norm v*
**unfolding** *norm-vec-def map-vector-def*
**by** (*intro L2-set-cong*) *auto*

**definition** *unitary-hma* :: *complex*$\widehat{\ }$*'n*$\widehat{\ }$*'n* $\Rightarrow$ *bool*
**where** *unitary-hma A* $\longleftrightarrow$ *A* $**$ *adjoint-hma A* = *Finite-Cartesian-Product.mat 1*

**definition** *unitarily-equiv-hma* **where**
*unitarily-equiv-hma A B U* $\equiv$ (*unitary-hma U* $\wedge$ *similar-matrix-wit A B U* (*adjoint-hma U*))

**definition** *diagonal-mat* :: (*'a::zero*)$\widehat{\ }$(*'n::finite*)$\widehat{\ }$*'n* $\Rightarrow$ *bool* **where**
*diagonal-mat A* $\equiv$ ($\forall$ *i*. $\forall$ *j*. $i \neq j \longrightarrow A\ \$h\ i\ \$h\ j = 0$)

**lemma** *diagonal-mat-ex*:
**assumes** *diagonal-mat A*
**shows** *A* = *diag* ($\chi$ *i*. $A\ \$h\ i\ \$h\ i$)
**using** *assms* **unfolding** *diagonal-mat-def diag-def*
**by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) *auto*

**lemma** *diag-diagonal-mat*[*simp*]: *diagonal-mat* (*diag x*)
**unfolding** *diag-def diagonal-mat-def* **by** *auto*

**lemma** *diag-imp-upper-tri*: *diagonal-mat A* $\Longrightarrow$ *upper-triangular-hma A*
**unfolding** *diagonal-mat-def upper-triangular-hma-def*
**by** (*metis nat-neq-iff*)

**definition** *unitary-diag* **where**
*unitary-diag A b U* $\equiv$ *unitarily-equiv-hma A* (*diag b*) *U*

**definition** *real-diag-decomp-hma* **where**
*real-diag-decomp-hma A d U* $\equiv$ *unitary-diag A d U* $\wedge$
($\forall$ *i*. $d\ \$h\ i \in$ *Reals*)

**definition** *hermitian-hma* :: *complex*$\widehat{\ }$*'n*$\widehat{\ }$*'n* $\Rightarrow$ *bool* **where**
*hermitian-hma A* = (*adjoint-hma A* = *A*)

**lemma** *from-hma-one*:
*from-hma*$_m$ (*mat 1* :: ((*'a*::{*one*,*zero*})$\widehat{\ }$*'n*$\widehat{\ }$*'n*)) = $1_m$ *CARD*(*'n*)
**unfolding** *Finite-Cartesian-Product.mat-def from-hma*$_m$*-def* **using** *from-nat-inj*
**by** (*intro eq-matI*) *auto*

**lemma** *from-hma-mult*:
**fixes** *A* :: (*'a* :: *semiring-1*)$\widehat{\ }$*'m*$\widehat{\ }$*'n*
**fixes** *B* :: *'a*$\widehat{\ }$*'k*$\widehat{\ }$*'m::finite*
**shows** *from-hma*$_m$ *A* $*$ *from-hma*$_m$ *B* = *from-hma*$_m$ (*A* $**$ *B*)
**using** *HMA-M-mult* **unfolding** *rel-fun-def HMA-M-def* **by** *auto*

**lemma** *hermitian-hma*:
*hermitian-hma A* = *hermitian* (*from-hma*$_m$ *A*)
**unfolding** *hermitian-def adjoint-hma hermitian-hma-def* **by** *auto*

**lemma** *unitary-hma*:
  **fixes** $A$ :: *complex*^'n^'n
  **shows** *unitary-hma A* = *unitary* (*from-hma*$_m$ *A*) (**is** *?L* = *?R*)
**proof** −
  **have** *?R* ⟷ *from-hma*$_m$ *A* ∗ *mat-adjoint* (*from-hma*$_m$ *A*) = $1_m$ (*CARD*('n))
    **using** *from-hma-carrier*
    **unfolding** *unitary-def inverts-mat-def* **by** *simp*
  **also have** ... ⟷ *from-hma*$_m$ (*A* ∗∗ *adjoint-hma A*) = *from-hma*$_m$ (*mat 1*::*complex*^'n^'n)
    **unfolding** *adjoint-hma from-hma-mult from-hma-one* **by** *simp*
  **also have** ... ⟷ *A* ∗∗ *adjoint-hma A* = *Finite-Cartesian-Product.mat 1*
    **unfolding** *from-hma*$_m$*-inj* **by** *simp*
  **also have** ... ⟷ *?L* **unfolding** *unitary-hma-def* **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *unitary-hmaD*:
  **fixes** $A$ :: *complex*^'n^'n
  **assumes** *unitary-hma A*
  **shows** *adjoint-hma A* ∗∗ *A* = *mat 1* (**is** *?A*) *A* ∗∗ *adjoint-hma A* = *mat 1* (**is** *?B*)
**proof** −
  **have** *mat-adjoint* (*from-hma*$_m$ *A*) ∗ *from-hma*$_m$ *A* = $1_m$ *CARD*('n)
    **using** *assms unitary-hma* **by** (*intro unitary-simps from-hma-carrier* ) *auto*
  **thus** *?A*
    **unfolding** *adjoint-hma from-hma-mult from-hma-one*[*symmetric*] *from-hma*$_m$*-inj*
    **by** *simp*
  **show** *?B*
    **using** *assms* **unfolding** *unitary-hma-def* **by** *simp*
**qed**

**lemma** *unitary-hma-adjoint*:
  **assumes** *unitary-hma A*
  **shows** *unitary-hma* (*adjoint-hma A*)
  **unfolding** *unitary-hma-def adjoint-adjoint-id unitary-hmaD*[*OF assms*] **by** *simp*

**lemma** *unitarily-equiv-hma*:
  **fixes** $A$ :: *complex*^'n^'n
  **shows** *unitarily-equiv-hma A B U* =
    *unitarily-equiv* (*from-hma*$_m$ *A*) (*from-hma*$_m$ *B*) (*from-hma*$_m$ *U*)
    (**is** *?L* = *?R*)
**proof** −
  **have** *?R* ⟷ (*unitary-hma U* ∧ *similar-mat-wit* (*from-hma*$_m$ *A*) (*from-hma*$_m$ *B*) (*from-hma*$_m$
*U*) (*from-hma*$_m$ (*adjoint-hma U*)))
   **unfolding** *Spectral-Theory-Complements.unitarily-equiv-def unitary-hma*[*symmetric*] *adjoint-hma*
    **by** *simp*
  **also have** ... ⟷ *unitary-hma U* ∧ *similar-matrix-wit A B U* (*adjoint-hma U*)
    **using** *HMA-similar-mat-wit* **unfolding** *rel-fun-def HMA-M-def*
    **by** (*intro arg-cong2*[**where** *f*=(∧)] *refl*) *force*
  **also have** ... ⟷ *?L*
    **unfolding** *unitarily-equiv-hma-def* **by** *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *Matrix-diagonal-matD*:
  **assumes** *Matrix.diagonal-mat A*
  **assumes** *i*<*dim-row A j*<*dim-col A*
  **assumes** *i* ≠ *j*
  **shows** *A* $$ (*i*,*j*) = *0*

  **using** *assms* **unfolding** *Matrix.diagonal-mat-def* **by** *auto*

**lemma** *diagonal-mat-hma*:
  **fixes** *A* :: (*'a* :: *zero*)^(*'n* :: *finite*)^*'n*
  **shows** *diagonal-mat A* = *Matrix.diagonal-mat* (*from-hma*$_m$ *A*) (**is** *?L* = *?R*)
**proof**
  **show** *?L* ⟹ *?R*
    **unfolding** *diagonal-mat-def Matrix.diagonal-mat-def from-hma*$_m$*-def*
    **using** *from-nat-inj* **by** *auto*
**next**
  **assume** *a*:*?R*

  **have** *A* \$*h i* \$*h j* = *0* **if** *i* ≠ *j* **for** *i j*
  **proof** −
    **have** *A* \$*h i* \$*h j* = (*from-hma*$_m$ *A*) \$\$ (*to-nat i*,*to-nat j*)
      **unfolding** *from-hma*$_m$*-def* **using** *to-nat-less-card*[**where** *'a*=*'n*] **by** *simp*
    **also have** ... = *0*
      **using** *to-nat-less-card*[**where** *'a*=*'n*] *to-nat-inj that*
      **by** (*intro Matrix-diagonal-matD*[*OF a*]) *auto*
    **finally show** *?thesis* **by** *simp*
  **qed**
  **thus** *?L*
    **unfolding** *diagonal-mat-def* **by** *auto*
**qed**

**lemma** *unitary-diag-hma*:
  **fixes** *A* :: *complex*^*'n*^*'n*
  **shows** *unitary-diag A d U* =
    *Spectral-Theory-Complements.unitary-diag* (*from-hma*$_m$ *A*) (*from-hma*$_m$ (*diag d*)) (*from-hma*$_m$
*U*)
**proof** −
  **have** *Matrix.diagonal-mat* (*from-hma*$_m$ (*diag d*))
    **unfolding** *diagonal-mat-hma*[*symmetric*] **by** *simp*
  **thus** *?thesis*
    **unfolding** *unitary-diag-def Spectral-Theory-Complements.unitary-diag-def unitarily-equiv-hma*
    **by** *auto*
**qed**

**lemma** *real-diag-decomp-hma*:
  **fixes** *A* :: *complex*^*'n*^*'n*
  **shows** *real-diag-decomp-hma A d U* =
    *real-diag-decomp* (*from-hma*$_m$ *A*) (*from-hma*$_m$ (*diag d*)) (*from-hma*$_m$ *U*)
**proof** −
  **have** *0*:(∀ *i*. *d* \$*h i* ∈ ℝ) ⟷ (∀ *i* < *CARD*(*'n*). *from-hma*$_m$ (*diag d*) \$\$ (*i*,*i*) ∈ ℝ)
    **unfolding** *from-hma*$_m$*-def diag-def* **using** *to-nat-less-card* **by** *fastforce*
  **show** *?thesis*
    **unfolding** *real-diag-decomp-hma-def real-diag-decomp-def unitary-diag-hma 0*
    **by** *auto*
**qed**

**lemma** *diagonal-mat-diag-ex-hma*:
  **assumes** *Matrix.diagonal-mat A A* ∈ *carrier-mat CARD*(*'n*) *CARD* (*'n* :: *finite*)
  **shows** *from-hma*$_m$ (*diag* (χ (*i*::*'n*). *A* \$\$ (*to-nat i*,*to-nat i*))) = *A*
  **using** *assms from-nat-inj* **unfolding** *from-hma*$_m$*-def diag-def Matrix.diagonal-mat-def*
  **by** (*intro eq-matI*) (*auto simp add*:*to-nat-from-nat-id*)

**theorem** *commuting-hermitian-family-diag-hma*:
  **fixes** *Af* :: (*complex*^*'n*^*'n*) *set*

**assumes** *finite Af*
**and** $Af \neq \{\}$
**and** $\bigwedge A.\ A \in Af \Longrightarrow$ *hermitian-hma A*
**and** $\bigwedge A\ B.\ A \in Af \Longrightarrow B \in Af \Longrightarrow A ** B = B ** A$
**shows** $\exists\ U.\ \forall\ A \in Af.\ \exists B.$ *real-diag-decomp-hma A B U*
**proof** −
**have** *0*:*finite* (*from-hma*$_m$ ‘ *Af*)
**using** *assms*(*1*)**by** (*intro finite-imageI*)
**have** *1*: *from-hma*$_m$ ‘ $Af \neq \{\}$
**using** *assms*(*2*) **by** *simp*
**have** *2*: $A \in$ *carrier-mat* (*CARD* ('*n*)) (*CARD* ('*n*)) **if** $A \in$ *from-hma*$_m$ ‘ *Af* **for** *A*
**using** *that* **unfolding** *from-hma*$_m$*-def* **by** (*auto simp add*:*image-iff*)
**have** *3*: $0 <$ *CARD*('*n*)
**by** *simp*
**have** *4*: *hermitian A* **if** $A \in$ *from-hma*$_m$ ‘ *Af* **for** *A*
**using** *hermitian-hma assms*(*3*) *that* **by** *auto*
**have** *5*: $A * B = B * A$ **if** $A \in$ *from-hma*$_m$ ‘ *Af* $B \in$ *from-hma*$_m$ ‘ *Af* **for** *A B*
**using** *that assms*(*4*) **by** (*auto simp add*:*image-iff from-hma-mult*)
**have** $\exists U.\ \forall A \in$ *from-hma*$_m$ ‘ *Af*. $\exists B.$ *real-diag-decomp A B U*
**using** *commuting-hermitian-family-diag*[*OF 0 1 2 3 4 5*] **by** *auto*
**then obtain** *U Bmap* **where** *U-def*: $\bigwedge A.\ A \in$ *from-hma*$_m$ ‘ *Af* $\Longrightarrow$ *real-diag-decomp A* (*Bmap A*) *U*
**by** *metis*
**define** $U'$ :: *complex*^'*n*^'*n* **where** $U' =$ *to-hma*$_m$ *U*
**define** *Bmap*$'$ :: *complex*^'*n*^'*n* $\Rightarrow$ *complex*^'*n*
**where** *Bmap*$' = (\lambda M.\ (\chi\ i.$ (*Bmap* (*from-hma*$_m$ *M*)) \$\$ (*to-nat i*,*to-nat i*)))

**have** *real-diag-decomp-hma A* (*Bmap*$'$ *A*) $U'$ **if** $A \in Af$ **for** *A*
**proof** −
**have** *rdd*: *real-diag-decomp* (*from-hma*$_m$ *A*) (*Bmap* (*from-hma*$_m$ *A*)) *U*
**using** *U-def that* **by** *simp*

**have** $U \in$ *carrier-mat CARD*('*n*) *CARD*('*n*) *Bmap* (*from-hma*$_m$ *A*) $\in$ *carrier-mat CARD*('*n*) *CARD*('*n*)
*Matrix*.*diagonal-mat* (*Bmap* (*from-hma*$_m$ *A*))
**using** *rdd* **unfolding** *real-diag-decomp-def Spectral-Theory-Complements*.*unitary-diag-def*
*Spectral-Theory-Complements*.*unitarily-equiv-def similar-mat-wit-def*
**by** (*auto simp add*:*Let-def*)

**hence** (*from-hma*$_m$ (*diag* (*Bmap*$'$ *A*))) = *Bmap* (*from-hma*$_m$ *A*) (*from-hma*$_m$ $U'$) = *U*
**unfolding** *Bmap*$'$*-def* $U'$*-def* **by** (*auto simp add*:*diagonal-mat-diag-ex-hma*)
**hence** *real-diag-decomp* (*from-hma*$_m$ *A*) (*from-hma*$_m$ (*diag* (*Bmap*$'$ *A*))) (*from-hma*$_m$ $U'$)
**using** *rdd* **by** *auto*
**thus** *?thesis*
**unfolding** *real-diag-decomp-hma* **by** *simp*
**qed**
**thus** *?thesis*
**by** (*intro exI*[**where** *x*=*U*']) *auto*
**qed**

**lemma** *char-poly-upper-triangular*:
**fixes** *A* :: *complex*^'*n*^'*n*
**assumes** *upper-triangular-hma A*
**shows** *charpoly A* = ($\prod$ *a* $\in$# *diag-mat-hma A*. [:− *a*, *1*:])
**proof** −
**have** *charpoly A* = *char-poly* (*from-hma*$_m$ *A*)
**using** *HMA-char-poly* **unfolding** *rel-fun-def HMA-M-def*
**by** (*auto simp add*:*eq-commute*)

**also have** ... = ($\prod$ *a*←*diag-mat* (*from-hma*$_m$ *A*). [:− *a*, *1*:])
  **using** *assms* **unfolding** *upper-triangular-hma*[*symmetric*]
  **by** (*intro char-poly-upper-triangular*[**where** *n*=*CARD*(′*n*)] *from-hma-carrier*) *auto*
**also have** ... = ($\prod$ *a*∈# *mset* (*diag-mat* (*from-hma*$_m$ *A*)). [:− *a*, *1*:])
  **unfolding** *prod-mset-prod-list*[*symmetric*] *mset-map* **by** *simp*
**also have** ... = ($\prod$ *a*∈# *diag-mat-hma A*. [:− *a*, *1*:])
  **unfolding** *diag-mat-hma* **by** *simp*
**finally show** *charpoly A* = ($\prod$ *a*∈# *diag-mat-hma A*. [:− *a*, *1*:]) **by** *simp*
**qed**

**lemma** *upper-tri-eigvals*:
  **fixes** *A* :: *complex*^′*n*^′*n*
  **assumes** *upper-triangular-hma A*
  **shows** *eigenvalues A* = *diag-mat-hma A*
**proof** −
  **have** ($\prod$ *a*∈#*eigenvalues A*. [:− *a*, *1*:]) = *charpoly A*
    **unfolding** *eigvals-poly-length*[*symmetric*] **by** *simp*
  **also have** ... = ($\prod$ *a*∈#*diag-mat-hma A*. [:− *a*, *1*:])
    **by** (*intro char-poly-upper-triangular assms*)
  **finally have** ($\prod$ *a*∈#*eigenvalues A*. [:− *a*, *1*:]) = ($\prod$ *a*∈#*diag-mat-hma A*. [:− *a*, *1*:])
    **by** *simp*
  **thus** *?thesis*
    **by** (*intro poly-prod-inj*) *simp*
**qed**

**lemma** *cinner-self*:
  **fixes** *v* :: *complex*^′*n*
  **shows** *cinner v v* = *norm v*^*2*
**proof** −
  **have** *0*: *x* ∗ *cnj x* = *complex-of-real* (*x* · *x*) **for** *x* :: *complex*
    **unfolding** *inner-complex-def complex-mult-cnj* **by** (*simp add*:*power2-eq-square*)
  **thus** *?thesis*
    **unfolding** *cinner-def power2-norm-eq-inner scalar-product-def inner-vec-def*
      *map-vector-def* **by** *simp*
**qed**

**lemma** *unitary-iso*:
  **assumes** *unitary-hma U*
  **shows** *norm* (*U* ∗*v v*) = *norm v*
**proof** −
  **have** *norm* (*U* ∗*v v*)^*2* = *cinner* (*U* ∗*v v*) (*U* ∗*v v*)
    **unfolding** *cinner-self* **by** *simp*
  **also have** ... = *cinner v v*
    **unfolding** *adjoint-def-alter-hma matrix-vector-mul-assoc unitary-hmaD*[*OF assms*] **by** *simp*
  **also have** ... = *norm v*^*2*
    **unfolding** *cinner-self* **by** *simp*
  **finally have** *complex-of-real* (*norm* (*U* ∗*v v*)^*2*) = *norm v*^*2* **by** *simp*
  **thus** *?thesis*
    **by** (*meson norm-ge-zero of-real-hom.injectivity power2-eq-iff-nonneg*)
**qed**

**lemma** (**in** *semiring-hom*) *mult-mat-vec-hma*:
  *map-vector hom* (*A* ∗*v v*) = *map-matrix hom A* ∗*v map-vector hom v*
  **using** *mult-mat-vec-hom* **by** *transfer auto*

**lemma** (**in** *semiring-hom*) *mat-hom-mult-hma*:
  *map-matrix hom* (*A* ∗∗ *B*) = *map-matrix hom A* ∗∗ *map-matrix hom B*
  **using** *mat-hom-mult* **by** *transfer auto*

```
context regular-graph-tts
begin

lemma to-nat-less-n: to-nat (x::'n) < n
  using to-nat-less-card card-n by metis

lemma to-nat-from-nat: x < n ⟹ to-nat (from-nat x :: 'n) = x
  using to-nat-from-nat-id card-n by metis

lemma hermitian-A: hermitian-hma A
  using count-sym unfolding hermitian-hma-def adjoint-hma-def A-def map-matrix-def
    map-vector-def transpose-def by simp

lemma nonneg-A: nonneg-mat A
  unfolding nonneg-mat-def A-def by auto

lemma g-step-1:
  assumes v ∈ verts G
  shows g-step (λ-. 1) v = 1 (is ?L = ?R)
proof −
  have ?L = in-degree G v / d
    unfolding g-step-def in-degree-def by simp
  also have ... = 1
    unfolding reg(2)[OF assms] using d-gt-0 by simp
  finally show ?thesis by simp
qed

lemma markov: markov (A :: real^'n^'n)
proof −
  have A *v 1 = (1::real ^'n) (is ?L = ?R)
  proof −
    have A *v 1 = (χ i. g-step (λ-. 1) (enum-verts i))
      unfolding g-step-conv one-vec-def by simp
    also have ... = (χ i. 1)
      using bij-betw-apply[OF enum-verts] by (subst g-step-1) auto
    also have ... = 1 unfolding one-vec-def by simp
    finally show ?thesis by simp
  qed
  thus ?thesis
    by (intro markov-symI nonneg-A symmetric-A)
qed

lemma nonneg-J: nonneg-mat J
  unfolding nonneg-mat-def J-def by auto

lemma J-eigvals: eigenvalues J = {#1::complex#} + replicate-mset (n − 1) 0
proof −
  define α :: nat ⇒ real where α i = sqrt (i^2+i) for i :: nat

  define q :: nat ⇒ nat ⇒ real
    where q i j = (
        if i = 0 then (1/sqrt n) else (
        if j < i then ((−1) / α i) else (
        if j = i then (i / α i) else 0))) for i j

  define Q :: complex^'n^'n where Q = (χ i j. complex-of-real (q (to-nat i) (to-nat j)))
```

```
define D :: complex^'n^'n where
  D = (χ i j. if to-nat i = 0 ∧ to-nat j = 0 then 1 else 0)

have 2: [0..<n] = 0#[1..<n]
  using n-gt-0 upt-conv-Cons by auto

have aux0: (∑ k = 0..<n. q j k ∗ q i k) = of-bool (i = j) if 1:i ≤ j j < n  for i j
proof −
  consider (a) i = j ∧ j = 0 | (b) i = 0 ∧ i < j | (c)  0 < i ∧ i < j | (d) 0 < i ∧ i = j
    using 1 by linarith
  thus ?thesis
  proof (cases)
    case a
    then show ?thesis using n-gt-0 by (simp add:q-def)
  next
    case b
    have (∑ k = 0..<n. q j k∗q i k)=(∑ k∈insert j ({0..<j} ∪ {j+1..<n}). q j k∗q i k)
      using that(2) by (intro sum.cong) auto
    also have ...=q j j∗q i j+(∑ k=0..<j. q j k ∗ q i k)+(∑ k=j+1..<n. q j k ∗ q i k)
      by (subst sum.insert) (auto simp add: sum.union-disjoint)
    also have ... = 0 using b unfolding q-def by simp
    finally show ?thesis using b by simp
  next
    case c
    have (∑ k = 0..<n. q j k∗q i k)=(∑ k∈insert i ({0..<i} ∪ {i+1..<n}). q j k∗q i k)
      using that(2) c by (intro sum.cong) auto
    also have ...=q j i∗q i i+(∑ k=0..<i. q j k ∗ q i k)+(∑ k=i+1..<n. q j k ∗ q i k)
      by (subst sum.insert) (auto simp add: sum.union-disjoint)
    also have ... =(−1) / α j ∗ i / α i+ i ∗ ((−1) / α j ∗  (−1) / α i)
      using c unfolding q-def by simp
    also have ... = 0
      by (simp add:algebra-simps)
    finally show ?thesis using c by simp
  next
    case d
    have real i + real i^2 = real (i + i^2) by simp
    also have ... ≠ real 0
      unfolding of-nat-eq-iff using d by simp
    finally have d-1: real i  + real i^2 ≠ 0 by simp
    have (∑ k = 0..<n. q j k∗q i k)=(∑ k∈insert i ({0..<i} ∪ {i+1..<n}). q j k∗q i k)
      using that(2) d by (intro sum.cong) auto
    also have ...=q j i∗q i i+(∑ k=0..<i. q j k ∗ q i k)+(∑ k=i+1..<n. q j k ∗ q i k)
      by (subst sum.insert) (auto simp add: sum.union-disjoint)
    also have ... = i/ α i ∗ i / α i+ i ∗ ((−1) / α i ∗  (−1) / α i)
      using d that unfolding q-def by simp
    also have ... = (i^2 + i) / (α i)^2
      by (simp add: power2-eq-square divide-simps)
    also have ... = 1
      using d-1 unfolding α-def by (simp add:algebra-simps)
    finally show ?thesis using d by simp
  qed
qed

have 0:(∑ k = 0..<n. q j k ∗ q i k) = of-bool (i = j) (is ?L = ?R)  if i < n j < n  for i j
proof −
  have ?L = (∑ k = 0..<n. q (max i j) k ∗ q (min i j) k)
    by (cases i ≤ j) ( simp-all add:ac-simps cong:sum.cong)
  also have ... = of-bool (min i j = max i j)
```

using *that* **by** (*intro aux0*) *auto*
also have ... = *?R*
by (*cases* $i \le j$) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**have** ($\sum k \in UNIV.\ Q\ \$h\ j\ \$h\ k * cnj\ (Q\ \$h\ i\ \$h\ k)) = of\text{-}bool\ (i{=}j)$ (**is** *?L = ?R*) **for** *i j*
**proof** −
**have** $?L = complex\text{-}of\text{-}real\ (\sum k \in (UNIV::'n\ set).\ q\ (to\text{-}nat\ j)\ (to\text{-}nat\ k) * q\ (to\text{-}nat\ i)\ (to\text{-}nat\ k))$
**unfolding** *Q-def*
**by** (*simp add*:*case-prod-beta scalar-prod-def map-vector-def inner-vec-def row-def inner-complex-def*)
**also have** ... = $complex\text{-}of\text{-}real\ (\sum k{=}0..{<}n.\ q\ (to\text{-}nat\ j)\ k * q\ (to\text{-}nat\ i)\ k)$
**using** *to-nat-less-n to-nat-from-nat*
**by** (*intro arg-cong*[**where** *f=of-real*] *sum.reindex-bij-betw bij-betwI*[**where** *g=from-nat*]) (*auto*)
**also have** ... = *complex-of-real* (*of-bool*(*to-nat i = to-nat j*))
**using** *to-nat-less-n* **by** (*intro arg-cong*[**where** *f=of-real*] *0*) *auto*
**also have** ... = *?R*
**using** *to-nat-inj* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**
**hence** $Q ** adjoint\text{-}hma\ Q = mat\ 1$
**by** (*intro iffD2*[*OF vec-eq-iff*]) (*auto simp add*:*matrix-matrix-mult-def mat-def adjoint-hma-eq*)
**hence** *unit-Q*: *unitary-hma Q*
**unfolding** *unitary-hma-def* **by** *simp*

**have** $card\ \{(k::'n).\ to\text{-}nat\ k = 0\} = card\ \{from\text{-}nat\ 0 :: 'n\}$
**using** *to-nat-from-nat*[**where** *x=0*] *n-gt-0*
**by** (*intro arg-cong*[**where** *f=card*] *iffD2*[*OF set-eq-iff*]) *auto*
**hence** $5{:}card\ \{(k::'n).\ to\text{-}nat\ k = 0\} = 1$ **by** *simp*
**hence** $1{:}adjoint\text{-}hma\ Q ** D = (\chi\ i\ j.\ (if\ to\text{-}nat\ j = 0\ then\ complex\text{-}of\text{-}real\ (1/sqrt\ n)\ else\ 0))$
**unfolding** *Q-def D-def* **by** (*intro iffD2*[*OF vec-eq-iff*] *allI*)
(*auto simp add*:*adjoint-hma-eq matrix-matrix-mult-def q-def if-distrib if-distribR sum.If-cases*)

**have** $(adjoint\text{-}hma\ Q ** D ** Q)\ \$h\ i\ \$h\ j = J\ \$h\ i\ \$h\ j$ (**is** *?L1 = ?R1*) **for** *i j*
**proof** −
**have** $?L1 = 1/((sqrt\ (real\ n)) * complex\text{-}of\text{-}real\ (sqrt\ (real\ n)))$
**unfolding** *1* **unfolding** *Q-def* **using** *n-gt-0 5*
**by** (*auto simp add*:*matrix-matrix-mult-def q-def if-distrib if-distribR sum.If-cases*)
**also have** ... = $1/sqrt\ (real\ n)\hat{\ }2$
**unfolding** *of-real-divide of-real-mult power2-eq-square*
**by** *simp*
**also have** ... = $J\ \$h\ i\ \$h\ j$
**unfolding** *J-def card-n* **using** *n-gt-0* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**hence** $adjoint\text{-}hma\ Q ** D ** Q = J$
**by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) *auto*

**hence** *similar-matrix-wit J D* (*adjoint-hma Q*) *Q*
**unfolding** *similar-matrix-wit-def unitary-hmaD*[*OF unit-Q*] **by** *auto*
**hence** *similar-matrix J D*
**unfolding** *similar-matrix-def* **by** *auto*
**hence** *eigenvalues J = eigenvalues D*
**by** (*intro similar-matrix-eigvals*)
**also have** ... = *diag-mat-hma D*

```
    by (intro upper-tri-eigvals diag-imp-upper-tri) (simp add:D-def diagonal-mat-def)
  also have ... = {# of-bool (to-nat i = 0). i ∈# mset-set (UNIV :: 'n set)#}
    unfolding diag-mat-hma-def D-def of-bool-def by simp
  also have ... = {# of-bool (i = 0). i ∈# mset-set (to-nat ` (UNIV :: 'n set))#}
    unfolding image-mset-mset-set[OF inj-to-nat, symmetric]
    by (simp add:image-mset.compositionality comp-def)
  also have ... = mset (map (λi. of-bool(i=0)) [0..<n])
    unfolding range-to-nat card-n mset-map by simp
  also have ... = mset (1 # map (λi. 0) [1..<n])
    unfolding 2 by (intro arg-cong[where f=mset]) simp
  also have ... = {#1#} + replicate-mset (n−1) 0
    using n-gt-0 by (simp add:map-replicate-const mset-repl)
  finally show ?thesis by simp
qed

lemma J-markov: markov J
proof −
  have nonneg-mat J
    unfolding J-def nonneg-mat-def by auto
  moreover have transpose J = J
    unfolding J-def transpose-def by auto
  moreover have J *v 1 = (1 :: real^'n)
    unfolding J-def by (simp add:matrix-vector-mult-def one-vec-def)
  ultimately show ?thesis
    by (intro markov-symI) auto
qed

lemma markov-complex-apply:
  assumes markov M
  shows (map-matrix complex-of-real M) *v (1 :: complex^'n) = 1 (is ?L = ?R)
proof −
  have ?L = (map-matrix complex-of-real M) *v (map-vector complex-of-real 1)
    by (intro arg-cong2[where f=(*v)] refl) (simp add: map-vector-def one-vec-def)
  also have ... = map-vector (complex-of-real) 1
    unfolding of-real-hom.mult-mat-vec-hma[symmetric] markov-apply[OF assms] by simp
  also have ... = ?R
    by (simp add: map-vector-def one-vec-def)
  finally show ?thesis by simp
qed

lemma J-A-comm-real: J ** A = A ** (J :: real^'n^'n)
proof −
  have 0: (∑k∈UNIV. A $h k $h i / real CARD('n)) = 1 / real CARD('n) (is ?L = ?R) for i
  proof −
    have ?L = (1 v* A) $h i / real CARD('n)
      unfolding vector-matrix-mult-def by (simp add:sum-divide-distrib)
    also have ... = ?R
      unfolding markov-apply[OF markov] by simp
    finally show ?thesis by simp
  qed
  have 1: (∑k∈UNIV. A $h i $h k / real CARD('n)) = 1 / real CARD('n) (is ?L = ?R) for i
  proof −
    have ?L = (A *v 1) $h i / real CARD('n)
      unfolding matrix-vector-mult-def by (simp add:sum-divide-distrib)
    also have ... = ?R
      unfolding markov-apply[OF markov] by simp
    finally show ?thesis by simp
  qed
```

**show** *?thesis*
  **unfolding** *J-def* **using** *0 1*
  **by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) (*simp add*:*matrix-matrix-mult-def*)
**qed**

**lemma** *J-A-comm*: $J ** A = A ** (J :: complex^{\frown\prime}n^{\frown\prime}n)$ (**is** *?L = ?R*)
**proof** −
  **have** $J ** A$ = *map-matrix complex-of-real* $(J ** A)$
    **unfolding** *of-real-hom.mat-hom-mult-hma J-def A-def*
    **by** (*auto simp add*:*map-matrix-def map-vector-def*)
  **also have** ... = *map-matrix complex-of-real* $(A ** J)$
    **unfolding** *J-A-comm-real* **by** *simp*
  **also have** ... = *map-matrix complex-of-real A* $**$ *map-matrix complex-of-real J*
    **unfolding** *of-real-hom.mat-hom-mult-hma* **by** *simp*
  **also have** ... = *?R*
    **unfolding** *A-def J-def*
    **by** (*auto simp add*:*map-matrix-def map-vector-def*)
  **finally show** *?thesis* **by** *simp*
**qed**

**definition** $\gamma_a$ :: *'n itself* ⇒ *real* **where**
  $\gamma_a$ - = (*if* $n > 1$ *then Max-mset* (*image-mset cmod* (*eigenvalues A* − {#*1*#})) *else 0*)

**definition** $\gamma_2$ :: *'n itself* ⇒ *real* **where**
  $\gamma_2$ - = (*if* $n > 1$ *then Max-mset* {# *Re x*. *x* ∈# (*eigenvalues A* − {#*1*#})#} *else 0*)

**lemma** *J-sym*: *hermitian-hma J*
  **unfolding** *J-def hermitian-hma-def*
  **by** (*intro iffD2*[*OF vec-eq-iff*] *allI*) (*simp add*: *adjoint-hma-eq*)

**lemma**
  **shows** *evs-real*: *set-mset* (*eigenvalues A*::*complex multiset*) ⊆ ℝ (**is** *?R1*)
    **and** *ev-1*: (*1*::*complex*) ∈# *eigenvalues A*
    **and** $\gamma_a$*-ge-0*: $\gamma_a$ *TYPE* (*'n*) ≥ *0*
    **and** *find-any-ev*:
      ∀ α ∈# *eigenvalues A* − {#*1*#}. ∃ *v*. *cinner v 1* = *0* ∧ *v* ≠ *0* ∧ *A* ∗*v v* = α ∗*s v*
    **and** $\gamma_a$*-bound*: ∀ *v*. *cinner v 1* = *0* ⟶ *norm* (*A* ∗*v v*) ≤ $\gamma_a$ *TYPE*(*'n*) ∗ *norm v*
    **and** $\gamma_2$*-bound*: ∀ (*v*::*real*$^{\frown\prime}$*n*). *v* · *1* = *0* ⟶ *v* · (*A* ∗*v v*) ≤ $\gamma_2$ *TYPE* (*'n*) ∗ *norm v*^*2*
**proof** −
  **have** ∃ *U*. ∀ *A*∈ {*J*,*A*}. ∃ *B*. *real-diag-decomp-hma A B U*
    **using** *J-sym hermitian-A J-A-comm*
    **by** (*intro commuting-hermitian-family-diag-hma*) *auto*
  **then obtain** *U Ad Jd*
    **where** *A-decomp*: *real-diag-decomp-hma A Ad U* **and** *K-decomp*: *real-diag-decomp-hma J Jd U*
    **by** *auto*
  **have** *J-sim*: *similar-matrix-wit J* (*diag Jd*) *U* (*adjoint-hma U*) **and**
    *unit-U*: *unitary-hma U*
    **using** *K-decomp* **unfolding** *real-diag-decomp-hma-def unitary-diag-def unitarily-equiv-hma-def*
    **by** *auto*

  **have** *diag-mat-hma* (*diag Jd*) = *eigenvalues* (*diag Jd*)
    **by** (*intro upper-tri-eigvals*[*symmetric*] *diag-imp-upper-tri J-sim*) *auto*
  **also have** ... = *eigenvalues J*
    **using** *J-sim* **by** (*intro similar-matrix-eigvals*[*symmetric*]) (*auto simp add*:*similar-matrix-def*)
  **also have** ... ={#*1*::*complex*#} + *replicate-mset* (*n* − *1*) *0*
    **unfolding** *J-eigvals* **by** *simp*

**finally have** *0*:*diag-mat-hma* (*diag Jd*) = {#*1*::*complex*#} + *replicate-mset* (*n* − *1*) *0* **by** *simp*
**hence** *1* ∈# *diag-mat-hma* (*diag Jd*) **by** *simp*
**then obtain** *i* **where** *i-def*:*Jd* $*h i* = *1*
**unfolding** *diag-mat-hma-def diag-def* **by** *auto*
**have** {# *Jd* $*h j*. *j* ∈# *mset-set* (*UNIV* − {*i*}) #} = {#*Jd* $*h j*. *j* ∈# *mset-set UNIV* − *mset-set* {*i*}#}
**unfolding** *diag-mat-hma-def* **by** (*intro arg-cong2*[**where** *f*=*image-mset*] *mset-set-Diff*) *auto*
**also have** ... = *diag-mat-hma* (*diag Jd*) − {#*1*#}
**unfolding** *diag-mat-hma-def diag-def* **by** (*subst image-mset-Diff*) (*auto simp add*:*i-def*)
**also have** ... = *replicate-mset* (*n* − *1*) *0*
**unfolding** *0* **by** *simp*
**finally have** {# *Jd* $*h j*. *j* ∈# *mset-set* (*UNIV* − {*i*}) #} = *replicate-mset* (*n* − *1*) *0*
**by** *simp*
**hence** *set-mset* {# *Jd* $*h j*. *j* ∈# *mset-set* (*UNIV* − {*i*}) #} ⊆ {*0*}
**by** *simp*
**hence** *1*:*Jd* $*h j* = *0* **if** *j* ≠ *i* **for** *j*
**using** *that* **by** *auto*

**define** *u* **where** *u* = *adjoint-hma U* ∗*v 1*
**define** *α* **where** *α* = *u* $*h i*

**have** *U* ∗*v u* = (*U* ∗∗ *adjoint-hma U*) ∗*v 1*
**unfolding** *u-def* **by** (*simp add*:*matrix-vector-mul-assoc*)
**also have** ... = *1*
**unfolding** *unitary-hmaD*[*OF unit-U*] **by** *simp*
**also have** ... ≠ *0*
**by** *simp*
**finally have** *U* ∗*v u* ≠ *0* **by** *simp*
**hence** *u-nz*: *u* ≠ *0*
**by** (*cases u* = *0*) *auto*

**have** *diag Jd* ∗*v u* = *adjoint-hma U* ∗∗ *U* ∗∗ *diag Jd* ∗∗ *adjoint-hma U* ∗*v 1*
**unfolding** *unitary-hmaD*[*OF unit-U*] *u-def* **by** (*auto simp add*:*matrix-vector-mul-assoc*)
**also have** ... = *adjoint-hma U* ∗∗ (*U* ∗∗ *diag Jd* ∗∗ *adjoint-hma U*) ∗*v 1*
**by** (*simp add*:*matrix-mul-assoc*)
**also have** ... = *adjoint-hma U* ∗∗ *J* ∗*v 1*
**using** *J-sim* **unfolding** *similar-matrix-wit-def* **by** *simp*
**also have** ... = *adjoint-hma U* ∗*v* (*map-matrix complex-of-real J* ∗*v 1*)
**by** (*simp add*:*map-matrix-def map-vector-def J-def matrix-vector-mul-assoc*)
**also have** ... = *u*
**unfolding** *u-def markov-complex-apply*[*OF J-markov*] **by** *simp*
**finally have** *u-ev*: *diag Jd* ∗*v u* = *u* **by** *simp*
**hence** *Jd* ∗ *u* = *u*
**unfolding** *diag-vec-mult-eq* **by** *simp*
**hence** *u* $*h j* = *0* **if** *j* ≠ *i* **for** *j*
**using** *1 that* **unfolding** *times-vec-def vec-eq-iff* **by** *auto*
**hence** *u-alt*: *u* = *axis i α*
**unfolding** *α-def axis-def vec-eq-iff* **by** *auto*
**hence** *α-nz*: *α* ≠ *0*
**using** *u-nz* **by** (*cases α*=*0*) *auto*

**have** *A-sim*: *similar-matrix-wit A* (*diag Ad*) *U* (*adjoint-hma U*) **and** *Ad-real*: ∀ *i*. *Ad* $*h i* ∈ ℝ
**using** *A-decomp* **unfolding** *real-diag-decomp-hma-def unitary-diag-def unitarily-equiv-hma-def*
**by** *auto*

**have** *diag-mat-hma* (*diag Ad*) = *eigenvalues* (*diag Ad*)
**by** (*intro upper-tri-eigvals*[*symmetric*] *diag-imp-upper-tri A-sim*) *auto*
**also have** ... = *eigenvalues A*

**using** *A-sim* **by** (*intro similar-matrix-eigvals*[*symmetric*]) (*auto simp add*:*similar-matrix-def*)
**finally have** *3*:*diag-mat-hma* (*diag Ad*) = *eigenvalues A*
**by** *simp*

**show** *?R1*
**unfolding** *3*[*symmetric*] *diag-mat-hma-def diag-def* **using** *Ad-real* **by** *auto*

**have** *diag Ad* ∗*v u* = *adjoint-hma U* ∗∗ *U* ∗∗ *diag Ad* ∗∗ *adjoint-hma U* ∗*v 1*
**unfolding** *unitary-hmaD*[*OF unit-U*] *u-def* **by** (*auto simp add*:*matrix-vector-mul-assoc*)
**also have** ... = *adjoint-hma U* ∗∗ (*U* ∗∗ *diag Ad* ∗∗ *adjoint-hma U*) ∗*v 1*
**by** (*simp add*:*matrix-mul-assoc*)
**also have** ... = *adjoint-hma U* ∗∗ *A* ∗*v 1*
**using** *A-sim* **unfolding** *similar-matrix-wit-def* **by** *simp*
**also have** ... = *adjoint-hma U* ∗*v* (*map-matrix complex-of-real A* ∗*v 1*)
**by** (*simp add*:*map-matrix-def map-vector-def A-def matrix-vector-mul-assoc*)
**also have** ... = *u*
**unfolding** *u-def markov-complex-apply*[*OF markov*] **by** *simp*
**finally have** *u-ev-A*: *diag Ad* ∗*v u* = *u* **by** *simp*
**hence** *Ad* ∗ *u* = *u*
**unfolding** *diag-vec-mult-eq* **by** *simp*
**hence** *5*:*Ad* \$*h i* = *1*
**using** $\alpha$*-nz* **unfolding** *u-alt times-vec-def vec-eq-iff axis-def* **by** *force*

**thus** *ev-1*: (*1*::*complex*) ∈# *eigenvalues A*
**unfolding** *3*[*symmetric*] *diag-mat-hma-def diag-def* **by** *auto*

**have** *eigenvalues A* − {#*1*#} = *diag-mat-hma* (*diag Ad*) − {#*1*#}
**unfolding** *3* **by** *simp*
**also have** ... = {#*Ad* \$*h j*. *j* ∈# *mset-set UNIV*#} − {# *Ad* \$*h i* #}
**unfolding** *5 diag-mat-hma-def diag-def* **by** *simp*
**also have** ... = {#*Ad* \$*h j*. *j* ∈# *mset-set UNIV* − *mset-set* {*i*}#}
**by** (*subst image-mset-Diff*) *auto*
**also have** ... = {#*Ad* \$*h j*. *j* ∈# *mset-set* (*UNIV* − {*i*})#}
**by** (*intro arg-cong2*[**where** *f*=*image-mset*] *mset-set-Diff*[*symmetric*]) *auto*
**finally have** *4*:*eigenvalues A* − {#*1*#} = {#*Ad* \$*h j*. *j* ∈# *mset-set* (*UNIV* − {*i*})#} **by** *simp*

**have** *cmod* (*Ad* \$*h k*) ≤ $\gamma_a$ *TYPE* (′*n*) **if** *n* > *1 k* ≠ *i* **for** *k*
**unfolding** $\gamma_a$*-def 4* **using** *that Max-ge* **by** *auto*
**moreover have** *k* = *i* **if** *n* = *1* **for** *k*
**using** *that to-nat-less-n* **by** *simp*
**ultimately have** *norm-Ad*: *norm* (*Ad* \$*h k*) ≤ $\gamma_a$ *TYPE* (′*n*) ∨ *k* = *i* **for** *k*
**using** *n-gt-0* **by** (*cases n* = *1*, *auto*)

**have** *Re* (*Ad* \$*h k*) ≤ $\gamma_2$ *TYPE* (′*n*) **if** *n* > *1 k* ≠ *i* **for** *k*
**unfolding** $\gamma_2$*-def 4* **using** *that Max-ge* **by** *auto*
**moreover have** *k* = *i* **if** *n* = *1* **for** *k*
**using** *that to-nat-less-n* **by** *simp*
**ultimately have** *Re-Ad*: *Re* (*Ad* \$*h k*) ≤ $\gamma_2$ *TYPE* (′*n*) ∨ *k* = *i* **for** *k*
**using** *n-gt-0* **by** (*cases n* = *1*, *auto*)

**show** $\Lambda_e$*-ge-0*: $\gamma_a$ *TYPE* (′*n*) ≥ *0*
**proof** (*cases n* > *1*)
**case** *True*
**then obtain** *k* **where** *k-def*: *k* ≠ *i*
**by** (*metis* (*full-types*) *card-n from-nat-inj n-gt-0 one-neq-zero*)
**have** *0* ≤ *cmod* (*Ad* \$*h k*)
**by** *simp*
**also have** ... ≤ $\gamma_a$ *TYPE* (′*n*)

**using** *norm-Ad k-def* **by** *auto*
**finally show** *?thesis* **by** *auto*
**next**
**case** *False*
**thus** *?thesis* **unfolding** $\gamma_a$*-def* **by** *simp*
**qed**

**have** $\exists v.$ *cinner v 1* = *0* $\wedge$ *v* $\neq$ *0* $\wedge$ *A* $*v$ *v* = $\beta$ $*s$ *v* **if** $\beta$*-ran*: $\beta$ $\in\#$ *eigenvalues A* $-$ {#*1*#}
**for** $\beta$
**proof** $-$
**obtain** *j* **where** *j-def*: $\beta$ = *Ad* $h *j j* $\neq$ *i*
**using** $\beta$*-ran* **unfolding** *4* **by** *auto*
**define** *v* **where** *v* = *U* $*v$ *axis j 1*

**have** *A* $*v$ *v* = *A* $**$ *U* $*v$ *axis j 1*
**unfolding** *v-def* **by** (*simp add*:*matrix-vector-mul-assoc*)
**also have** ... = ((*U* $**$ *diag Ad* $**$ *adjoint-hma U*) $**$ *U*) $*v$ *axis j 1*
**using** *A-sim* **unfolding** *similar-matrix-wit-def* **by** *simp*
**also have** ... = *U* $**$ *diag Ad* $**$ (*adjoint-hma U* $**$ *U*) $*v$ *axis j 1*
**by** (*simp add*:*matrix-mul-assoc*)
**also have** ... = *U* $**$ *diag Ad* $*v$ *axis j 1*
**using** *unitary-hmaD*[*OF unit-U*] **by** *simp*
**also have** ... = *U* $*v$ (*Ad* $*$ *axis j 1*)
**by** (*simp add*:*matrix-vector-mul-assoc*[*symmetric*] *diag-vec-mult-eq*)
**also have** ... = *U* $*v$ ($\beta$ $*s$ *axis j 1*)
**by** (*intro arg-cong2*[**where** *f*=($*v$)] *iffD2*[*OF vec-eq-iff*]) (*auto simp*:*j-def axis-def*)
**also have** ... = $\beta$ $*s$ *v*
**unfolding** *v-def* **by** (*simp add*:*vector-scalar-commute*)
**finally have** *5*:*A* $*v$ *v* = $\beta$ $*s$ *v* **by** *simp*

**have** *cinner v 1* = *cinner* (*axis j 1*) (*adjoint-hma U* $*v$ *1*)
**unfolding** *v-def adjoint-def-alter-hma* **by** *simp*
**also have** ... = *cinner* (*axis j 1*) (*axis i* $\alpha$)
**unfolding** *u-def*[*symmetric*] *u-alt* **by** *simp*
**also have** ... = *0*
**using** *j-def*(*2*) **unfolding** *cinner-def axis-def scalar-product-def map-vector-def*
**by** (*auto simp*:*if-distrib if-distribR sum.If-cases*)
**finally have** *6*:*cinner v 1* = *0*
**by** *simp*

**have** *cinner v v* = *cinner* (*axis j 1*) (*adjoint-hma U* $*v$ (*U* $*v$ (*axis j 1*)))
**unfolding** *v-def adjoint-def-alter-hma* **by** *simp*
**also have** ... = *cinner* (*axis j 1*) (*axis j 1*)
**unfolding** *matrix-vector-mul-assoc unitary-hmaD*[*OF unit-U*] **by** *simp*
**also have** ... = *1*
**unfolding** *cinner-def axis-def scalar-product-def map-vector-def*
**by** (*auto simp*:*if-distrib if-distribR sum.If-cases*)
**finally have** *cinner v v* = *1*
**by** *simp*
**hence** *7*:*v* $\neq$ *0*
**by** (*cases v*=*0*) (*auto simp add*:*cinner-0*)

**show** *?thesis*
**by** (*intro exI*[**where** *x*=*v*] *conjI 6 7 5*)
**qed**

**thus** $\forall\alpha$ $\in\#$ *eigenvalues A* $-$ {#*1*#}. $\exists v.$ *cinner v 1* = *0* $\wedge$ *v* $\neq$ *0* $\wedge$ *A* $*v$ *v* = $\alpha$ $*s$ *v*
**by** *simp*

**have** *norm* $(A *v\ v) \le \gamma_a$ *TYPE*$('n) *$ *norm v* **if** *cinner v 1* $= 0$ **for** *v*
**proof** –
**define** *w* **where** *w*= *adjoint-hma U* $*v$ *v*

**have** *w* $h *i* = *cinner w* (*axis i 1*)
**unfolding** *cinner-def axis-def scalar-product-def map-vector-def*
**by** (*auto simp*:*if-distrib if-distribR sum.If-cases*)
**also have** ... = *cinner v* (*U* $*v$ *axis i 1*)
**unfolding** *w-def adjoint-def-alter-hma* **by** *simp*
**also have** ... = *cinner v* $((1\ /\ \alpha) *s\ (U *v\ u))$
**unfolding** *vector-scalar-commute*[*symmetric*] *u-alt* **using** $\alpha$-*nz*
**by** (*intro-cong* [$\sigma_2$ *cinner*, $\sigma_2$ $(*v)$]) (*auto simp add*:*axis-def vec-eq-iff*)
**also have** ... = *cinner v* $((1\ /\ \alpha) *s\ 1)$
**unfolding** *u-def matrix-vector-mul-assoc unitary-hmaD*[*OF unit-U*] **by** *simp*
**also have** ... = *0*
**unfolding** *cinner-scale-right that* **by** *simp*
**finally have** *w-orth*: *w* $h *i* = *0* **by** *simp*

**have** *norm* $(A *v\ v) =$ *norm* $(U *v$ (*diag Ad* $*v\ w))$
**using** *A-sim* **unfolding** *matrix-vector-mul-assoc similar-matrix-wit-def w-def*
**by** (*simp add*:*matrix-mul-assoc*)
**also have** ... = *norm* (*diag Ad* $*v\ w)$
**unfolding** *unitary-iso*[*OF unit-U*] **by** *simp*
**also have** ... = *norm* $(Ad * w)$
**unfolding** *diag-vec-mult-eq* **by** *simp*
**also have** ... = *sqrt* $(\sum i\in UNIV.$ (*cmod* (*Ad* $h *i*) $*$ *cmod* (*w* $h *i*))$^2$)
**unfolding** *norm-vec-def L2-set-def times-vec-def* **by** (*simp add*:*norm-mult*)
**also have** ... $\le$ *sqrt* $(\sum i\in UNIV.$ $((\gamma_a$ *TYPE*$('n)) *$ *cmod* (*w* $h *i*))^2)
**using** *w-orth norm-Ad*
**by** (*intro iffD2*[*OF real-sqrt-le-iff*] *sum-mono power-mono mult-right-mono*$'$) *auto*
**also have** ... = $|\gamma_a$ *TYPE*$('n)| *$ *sqrt* $(\sum i\in UNIV.$ (*cmod* (*w* $h *i*))$^2$)
**by** (*simp add*:*power-mult-distrib sum-distrib-left*[*symmetric*] *real-sqrt-mult*)
**also have** ... = $|\gamma_a$ *TYPE*$('n)| *$ *norm w*
**unfolding** *norm-vec-def L2-set-def* **by** *simp*
**also have** ... = $\gamma_a$ *TYPE*$('n) *$ *norm w*
**using** $\Lambda_e$-*ge-0* **by** *simp*
**also have** ... = $\gamma_a$ *TYPE*$('n) *$ *norm v*
**unfolding** *w-def unitary-iso*[*OF unitary-hma-adjoint*[*OF unit-U*]] **by** *simp*
**finally show** *norm* $(A *v\ v) \le \gamma_a$ *TYPE*$('n) *$ *norm v*
**by** *simp*
**qed**

**thus** $\forall v.$ *cinner v 1* $= 0 \longrightarrow$ *norm* $(A *v\ v) \le \gamma_a$ *TYPE*$('n) *$ *norm v* **by** *auto*

**have** $v \cdot (A *v\ v) \le \gamma_2$ *TYPE* $('n) *$ *norm v*^*2* **if** $v \cdot 1 = 0$ **for** *v* :: *real*^$'n$
**proof** –
**define** $v'$ **where** $v'$ = *map-vector complex-of-real v*
**define** *w* **where** *w*= *adjoint-hma U* $*v\ v'$

**have** *w* $h *i* = *cinner w* (*axis i 1*)
**unfolding** *cinner-def axis-def scalar-product-def map-vector-def*
**by** (*auto simp*:*if-distrib if-distribR sum.If-cases*)
**also have** ... = *cinner* $v'$ (*U* $*v$ *axis i 1*)
**unfolding** *w-def adjoint-def-alter-hma* **by** *simp*
**also have** ... = *cinner* $v'$ $((1\ /\ \alpha) *s\ (U *v\ u))$
**unfolding** *vector-scalar-commute*[*symmetric*] *u-alt* **using** $\alpha$-*nz*
**by** (*intro-cong* [$\sigma_2$ *cinner*, $\sigma_2$ $(*v)$]) (*auto simp add*:*axis-def vec-eq-iff*)

**also have** ... = *cinner* $v'$ ((*1* / $\alpha$) *$*s$ *1*)
**unfolding** *u-def matrix-vector-mul-assoc unitary-hmaD*[*OF unit-U*] **by** *simp*
**also have** ... = *cnj* (*1* / $\alpha$) $*$ *cinner* $v'$ *1*
**unfolding** *cinner-scale-right* **by** *simp*
**also have** ... = *cnj* (*1* / $\alpha$) $*$ *complex-of-real* ($v \cdot$ *1*)
**unfolding** *cinner-def scalar-product-def map-vector-def inner-vec-def* $v'$*-def*
**by** (*intro arg-cong2*[**where** *f*=($*$)] *refl*) (*simp*)
**also have** ... = *0*
**unfolding** *that* **by** *simp*
**finally have** *w-orth*: *w* \$*h* *i* = *0* **by** *simp*

**have** *complex-of-real* (*norm* $v$^*2*) = *complex-of-real* ($v \cdot v$)
**by** (*simp add*: *power2-norm-eq-inner*)
**also have** ... = *cinner* $v'$ $v'$
**unfolding** $v'$*-def cinner-def scalar-product-def inner-vec-def map-vector-def* **by** *simp*
**also have** ... = *norm* $v'$^*2*
**unfolding** *cinner-self* **by** *simp*
**also have** ... = *norm* *w*^*2*
**unfolding** *w-def unitary-iso*[*OF unitary-hma-adjoint*[*OF unit-U*]] **by** *simp*
**also have** ... = *cinner w w*
**unfolding** *cinner-self* **by** *simp*
**also have** ... = ($\sum$*j*$\in$*UNIV*. *complex-of-real* (*cmod* (*w* \$*h* *j*)^*2*))
**unfolding** *cinner-def scalar-product-def map-vector-def*
*cmod-power2 complex-mult-cnj*[*symmetric*] **by** *simp*
**also have** ... = *complex-of-real* ($\sum$*j*$\in$*UNIV*. (*cmod* (*w* \$*h* *j*)^*2*))
**by** *simp*
**finally have** *complex-of-real* (*norm* $v$^*2*) = *complex-of-real* ($\sum$*j*$\in$*UNIV*. (*cmod* (*w* \$*h* *j*)^*2*))
**by** *simp*
**hence** *norm-v*: *norm* $v$^*2* = ($\sum$*j*$\in$*UNIV*. (*cmod* (*w* \$*h* *j*)^*2*))
**using** *of-real-hom.injectivity* **by** *blast*

**have** *complex-of-real* ($v \cdot$ (*A* $*v$ *v*)) = *cinner* $v'$ (*map-vector of-real* (*A* $*v$ *v*))
**unfolding** $v'$*-def cinner-def scalar-product-def inner-vec-def map-vector-def*
**by** *simp*
**also have** ... = *cinner* $v'$ (*map-matrix of-real A* $*v$ $v'$)
**unfolding** $v'$*-def of-real-hom.mult-mat-vec-hma* **by** *simp*
**also have** ... = *cinner* $v'$ (*A* $*v$ $v'$)
**unfolding** *map-matrix-def map-vector-def A-def* **by** *auto*
**also have** ... = *cinner* $v'$ (*U* $**$ *diag Ad* $**$ *adjoint-hma U* $*v$ $v'$)
**using** *A-sim* **unfolding** *similar-matrix-wit-def* **by** *simp*
**also have** ... = *cinner* (*adjoint-hma U* $*v$ $v'$) (*diag Ad* $**$ *adjoint-hma U* $*v$ $v'$)
**unfolding** *adjoint-def-alter-hma adjoint-adjoint adjoint-adjoint-id*
**by** (*simp add*:*matrix-vector-mul-assoc matrix-mul-assoc*)
**also have** ... = *cinner w* (*diag Ad* $*v$ *w*)
**unfolding** *w-def* **by** (*simp add*:*matrix-vector-mul-assoc*)
**also have** ... = *cinner w* (*Ad* $*$ *w*)
**unfolding** *diag-vec-mult-eq* **by** *simp*
**also have** ... = ($\sum$*j*$\in$*UNIV*. *cnj* (*Ad* \$*h* *j*) $*$ *cmod* (*w* \$*h* *j*)^*2*)
**unfolding** *cinner-def map-vector-def scalar-product-def cmod-power2 complex-mult-cnj*[*symmetric*]
**by** (*simp add*:*algebra-simps*)
**also have** ... = ($\sum$*j*$\in$*UNIV*. *Ad* \$*h* *j* $*$ *cmod* (*w* \$*h* *j*)^*2*)
**using** *Ad-real* **by** (*intro sum.cong refl arg-cong2*[**where** *f*=($*$)] *iffD1*[*OF Reals-cnj-iff*]) *auto*
**also have** ... = ($\sum$*j*$\in$*UNIV*. *complex-of-real* (*Re* (*Ad* \$*h* *j*) $*$ *cmod* (*w* \$*h* *j*)^*2*))
**using** *Ad-real* **by** (*intro sum.cong refl*) *simp*
**also have** ... = *complex-of-real* ($\sum$*j*$\in$ *UNIV*. *Re* (*Ad* \$*h* *j*) $*$ *cmod* (*w* \$*h* *j*)^*2*)
**by** *simp*
**finally have** *complex-of-real* ($v\cdot$(*A* $*v$ *v*)) = *of-real*($\sum$*j*$\in$*UNIV*. *Re* (*Ad* \$*h* *j*) $*$ *cmod* (*w* \$*h*
*j*)^*2*)

**by** *simp*
**hence** $v \cdot (A *v\ v) = (\sum j \in UNIV.\ Re\ (Ad\ \$h\ j) * cmod\ (w\ \$h\ j)\hat{}2)$
**using** *of-real-hom.injectivity* **by** *blast*
**also have** ... $\leq (\sum j \in UNIV.\ \gamma_2\ TYPE\ ('n) * cmod\ (w\ \$h\ j)\hat{}2)$
**using** *w-orth Re-Ad* **by** (*intro sum-mono mult-right-mono'*) *auto*
**also have** ... $= \gamma_2\ TYPE\ ('n) * (\sum j \in UNIV.\ cmod\ (w\ \$h\ j)\hat{}2)$
**by** (*simp add:sum-distrib-left*)
**also have** ... $= \gamma_2\ TYPE\ ('n) * norm\ v\hat{}2$
**unfolding** *norm-v* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**thus** $\forall (v::real\hat{}'n).\ v \cdot 1 = 0 \longrightarrow v \cdot (A *v\ v) \leq \gamma_2\ TYPE\ ('n) * norm\ v\hat{}2$
**by** *auto*
**qed**

**lemma** *find-any-real-ev*:
**assumes** *complex-of-real* $\alpha \in\#$ *eigenvalues* $A - \{\#1\#\}$
**shows** $\exists v.\ v \cdot 1 = 0 \wedge v \neq 0 \wedge A *v\ v = \alpha *s\ v$
**proof** −
**obtain** *w* **where** *w-def*: *cinner w 1* $= 0$ $w \neq 0$ $A *v\ w = \alpha *s\ w$
**using** *find-any-ev assms* **by** *auto*

**have** $w = 0$ **if** *map-vector Re* $(1 *s\ w) = 0$ *map-vector Re* $(\mathrm{i} *s\ w) = 0$
**using** *that* **by** (*simp add:vec-eq-iff map-vector-def complex-eq-iff*)
**then obtain** *c* **where** *c-def*: *map-vector Re* $(c *s\ w) \neq 0$
**using** *w-def(2)* **by** *blast*

**define** *u* **where** $u = c *s\ w$

**define** *v* **where** $v =$ *map-vector Re u*

**hence** $v \cdot 1 = Re\ (cinner\ u\ 1)$
**unfolding** *cinner-def inner-vec-def scalar-product-def map-vector-def* **by** *simp*
**also have** ... $= 0$
**unfolding** *u-def cinner-scale-left w-def(1)* **by** *simp*
**finally have** $1{:}v \cdot 1 = 0$ **by** *simp*

**have** $A *v\ v = (\chi\ i.\ \sum j \in UNIV.\ A\ \$h\ i\ \$h\ j * Re\ (u\ \$h\ j))$
**unfolding** *matrix-vector-mult-def v-def map-vector-def* **by** *simp*
**also have** ... $= (\chi\ i.\ \sum j \in UNIV.\ Re\ (\ of\text{-}real\ (A\ \$h\ i\ \$h\ j) * u\ \$h\ j))$
**by** *simp*
**also have** ... $= (\chi\ i.\ Re\ (\sum j \in UNIV.\ A\ \$h\ i\ \$h\ j * u\ \$h\ j))$
**unfolding** *A-def* **by** *simp*
**also have** ... $=$ *map-vector Re* $(A *v\ u)$
**unfolding** *map-vector-def matrix-vector-mult-def* **by** *simp*
**also have** ... $=$ *map-vector Re* $(of\text{-}real\ \alpha *s\ u)$
**unfolding** *u-def vector-scalar-commute w-def(3)*
**by** (*simp add:ac-simps*)
**also have** ... $= \alpha *s\ v$
**unfolding** *v-def* **by** (*simp add:vec-eq-iff map-vector-def*)
**finally have** *2*: $A *v\ v = \alpha *s\ v$ **by** *simp*

**have** $3{:}v \neq 0$
**unfolding** *v-def u-def* **using** *c-def* **by** *simp*

**show** *?thesis*
**by** (*intro exI*[**where** $x{=}v$] *conjI 1 2 3*)

**qed**

**lemma** *size-evs*:
*size* (*eigenvalues A* $-$ {#*1*::*complex*#}) = *n*$-$*1*
**proof** $-$
**have** *size* (*eigenvalues A* :: *complex multiset*) = *n*
**using** *eigvals-poly-length card-n*[*symmetric*] **by** *auto*
**thus** *size* (*eigenvalues A* $-$ {#(*1*::*complex*)#}) = *n* $-$*1*
**using** *ev-1* **by** (*simp add*: *size-Diff-singleton*)
**qed**

**lemma** *find-*$\gamma_2$:
**assumes** *n* > *1*
**shows** $\gamma_a$ *TYPE*(′*n*) ∈# *image-mset cmod* (*eigenvalues A* $-$ {#*1*::*complex*#})
**proof** $-$
**have** *set-mset* (*eigenvalues A* $-$ {#(*1*::*complex*)#}) ≠ {}
**using** *assms size-evs* **by** *auto*
**hence** *2*: *cmod* ‘ *set-mset* (*eigenvalues A* $-$ {#*1*#}) ≠ {}
**by** *simp*
**have** $\gamma_a$ *TYPE*(′*n*) ∈ *set-mset* (*image-mset cmod* (*eigenvalues A* $-$ {#*1*#}))
**unfolding** $\gamma_a$*-def* **using** *assms 2 Max-in* **by** *auto*
**thus** $\gamma_a$ *TYPE*(′*n*) ∈# *image-mset cmod* (*eigenvalues A* $-$ {#*1*#})
**by** *simp*
**qed**

**lemma** $\gamma_2$*-real-ev*:
**assumes** *n* > *1*
**shows** ∃ *v*. (∃ α. *abs* α=$\gamma_a$ *TYPE*(′*n*) ∧ *v* · *1*=*0* ∧ *v* ≠ *0* ∧ *A* ∗*v v* = α ∗*s v*)
**proof** $-$
**obtain** α **where** α*-def*: *cmod* α = $\gamma_a$ *TYPE*(′*n*) α ∈# *eigenvalues A* $-$ {#*1*#}
**using** *find-*$\gamma_2$[*OF assms*] **by** *auto*
**have** α ∈ ℝ
**using** *in-diffD*[*OF* α*-def*(*2*)] *evs-real* **by** *auto*
**then obtain** β **where** β*-def*: α = *of-real* β
**using** *Reals-cases* **by** *auto*

**have** *0*:*complex-of-real* β ∈# *eigenvalues A*$-${#*1*#}
**using** α*-def* **unfolding** β*-def* **by** *auto*

**have** *1*: |β| = $\gamma_a$ *TYPE*(′*n*)
**using** α*-def* **unfolding** β*-def* **by** *simp*
**show** *?thesis*
**using** *find-any-real-ev*[*OF 0*] *1* **by** *auto*
**qed**

**lemma** $\gamma_a$*-real-bound*:
**fixes** *v* :: *real*^′*n*
**assumes** *v* · *1* = *0*
**shows** *norm* (*A* ∗*v v*) ≤ $\gamma_a$ *TYPE*(′*n*) ∗ *norm v*
**proof** $-$
**define** *w* **where** *w* = *map-vector complex-of-real v*

**have** *cinner w 1* = *v* · *1*
**unfolding** *w-def cinner-def map-vector-def scalar-product-def inner-vec-def*
**by** *simp*
**also have** ... = *0* **using** *assms* **by** *simp*
**finally have** *0*: *cinner w 1* = *0* **by** *simp*
**have** *norm* (*A* ∗*v v*) = *norm* (*map-matrix complex-of-real A* ∗*v* (*map-vector complex-of-real v*))

**unfolding** *norm-of-real of-real-hom.mult-mat-vec-hma*[*symmetric*] **by** *simp*
**also have** ... = *norm* ($A$ *∗v* $w$)
**unfolding** *w-def A-def map-matrix-def map-vector-def* **by** *simp*
**also have** ... $\le \gamma_a$ *TYPE*('*n*) ∗ *norm w*
**using** $\gamma_a$*-bound 0* **by** *auto*
**also have** ... = $\gamma_a$ *TYPE*('*n*) ∗ *norm v*
**unfolding** *w-def norm-of-real* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** $\Lambda_e$*-eq-*$\Lambda$: $\Lambda_a = \gamma_a$ *TYPE*('*n*)
**proof** −
**have** |*g-inner f* (*g-step f*)| $\le \gamma_a$ *TYPE*('*n*) ∗ (*g-norm f*)$^2$
(**is** *?L* ≤ *?R*) **if** *g-inner f* (λ-. *1*) = *0* **for** *f*
**proof** −
**define** *v* **where** *v* = (χ *i*. *f* (*enum-verts i*))
**have** *0*: *v* · *1* = *0*
**using** *that* **unfolding** *g-inner-conv one-vec-def v-def* **by** *auto*
**have** *?L* = |*v* · (*A* ∗*v v*)|
**unfolding** *g-inner-conv g-step-conv v-def* **by** *simp*
**also have** ... ≤ (*norm v* ∗ *norm* (*A* ∗*v v*))
**by** (*intro Cauchy-Schwarz-ineq2*)
**also have** ... ≤ (*norm v* ∗ ($\gamma_a$ *TYPE*('*n*) ∗ *norm v*))
**by** (*intro mult-left-mono* $\gamma_a$*-real-bound 0*) *auto*
**also have** ... = *?R*
**unfolding** *g-norm-conv v-def* **by** (*simp add*:*algebra-simps power2-eq-square*)
**finally show** *?thesis* **by** *simp*
**qed**
**hence** $\Lambda_a \le \gamma_a$ *TYPE*('*n*)
**using** $\gamma_a$*-ge-0* **by** (*intro expander-intro-1*) *auto*

**moreover have** $\Lambda_a \ge \gamma_a$ *TYPE*('*n*)
**proof** (*cases n* > *1*)
**case** *True*
**then obtain** *v* α **where** *v-def*: *abs* α = $\gamma_a$ *TYPE*('*n*) *A* ∗*v v* =α ∗*s v v* ≠ *0 v* · *1* = *0*
**using** $\gamma_2$*-real-ev* **by** *auto*
**define** *f* **where** *f x* = *v* \$*h enum-verts-inv x* **for** *x*
**have** *v-alt*: *v* = (χ *i*. *f* (*enum-verts i*))
**unfolding** *f-def Rep-inverse* **by** *simp*

**have** *g-inner f* (λ-. *1*) = *v* · *1*
**unfolding** *g-inner-conv v-alt one-vec-def* **by** *simp*
**also have** ... = *0* **using** *v-def* **by** *simp*
**finally have** *2*:*g-inner f* (λ-. *1*) = *0* **by** *simp*

**have** $\gamma_a$ *TYPE*('*n*) ∗ *g-norm f*^*2* = $\gamma_a$ *TYPE*('*n*) ∗ *norm v*^*2*
**unfolding** *g-norm-conv v-alt* **by** *simp*
**also have** ... = $\gamma_a$ *TYPE*('*n*) ∗ |*v* · *v*|
**by** (*simp add*: *power2-norm-eq-inner*)
**also have** ... = |*v* · (α ∗*s v*)|
**unfolding** *v-def*(*1*)[*symmetric*] *scalar-mult-eq-scaleR*
**by** (*simp add*:*abs-mult*)
**also have** ... = |*v* · (*A* ∗*v v*)|
**unfolding** *v-def* **by** *simp*
**also have** ... = |*g-inner f* (*g-step f*)|
**unfolding** *g-inner-conv g-step-conv v-alt* **by** *simp*
**also have** ... $\le \Lambda_a$ ∗ *g-norm f*^*2*
**by** (*intro expansionD1 2*)

**finally have** $\gamma_a$ *TYPE('n)* $*$ *g-norm f^2* $\leq$ $\Lambda_a$ $*$ *g-norm f^2* **by** *simp*
**moreover have** *norm v^2* $> 0$
**using** *v-def(3)* **by** *simp*
**hence** *g-norm f^2* $> 0$
**unfolding** *g-norm-conv v-alt* **by** *simp*
**ultimately show** *?thesis* **by** *simp*
**next**
**case** *False*
**hence** $n = 1$ **using** *n-gt-0* **by** *simp*
**hence** $\gamma_a$ *TYPE('n)* $= 0$
**unfolding** $\gamma_a$*-def* **by** *simp*

**then show** *?thesis* **using** $\Lambda$*-ge-0* **by** *simp*
**qed**
**ultimately show** *?thesis* **by** *simp*
**qed**

**lemma** $\gamma_2$*-ev*:
**assumes** $n > 1$
**shows** $\exists v.\ v \cdot 1 = 0 \wedge v \neq 0 \wedge A *v\ v = \gamma_2$ *TYPE('n)* $*s\ v$
**proof** –
**have** *set-mset (eigenvalues A* $-$ *{#1::complex#})* $\neq$ *{}*
**using** *size-evs assms* **by** *auto*
**hence** *Max (Re ‘ set-mset (eigenvalues A* $-$ *{#1#}))* $\in$ *Re ‘ set-mset (eigenvalues A* $-$ *{#1#})*
**by** *(intro Max-in) auto*
**hence** $\gamma_2$ *TYPE ('n)* $\in$ *Re ‘ set-mset (eigenvalues A* $-$ *{#1#})*
**unfolding** $\gamma_2$*-def* **using** *assms* **by** *simp*
**then obtain** $\alpha$ **where** $\alpha$*-def*: $\alpha \in$ *set-mset (eigenvalues A* $-$ *{#1#})* $\gamma_2$ *TYPE ('n)* $=$ *Re* $\alpha$
**by** *auto*
**have** $\alpha$*-real*: $\alpha \in \mathbb{R}$
**using** *evs-real in-diffD[OF* $\alpha$*-def(1)]* **by** *auto*
**have** *complex-of-real (*$\gamma_2$ *TYPE ('n))* $=$ *of-real (Re* $\alpha$*)*
**unfolding** $\alpha$*-def* **by** *simp*
**also have** ... $= \alpha$
**using** $\alpha$*-real* **by** *simp*
**also have** ... $\in$# *eigenvalues A* $-$ *{#1#}*
**using** $\alpha$*-def(1)* **by** *simp*
**finally have** *0:complex-of-real (*$\gamma_2$ *TYPE ('n))* $\in$# *eigenvalues A* $-$ *{#1#}* **by** *simp*
**thus** *?thesis*
**using** *find-any-real-ev[OF 0]* **by** *auto*
**qed**

**lemma** $\Lambda_2$*-eq-*$\gamma_2$: $\Lambda_2 = \gamma_2$ *TYPE ('n)*
**proof** *(cases n > 1)*
**case** *True*

**obtain** *v* **where** *v-def*: $v \cdot 1 = 0\ v \neq 0\ A *v\ v = \gamma_2$ *TYPE('n)* $*s\ v$
**using** $\gamma_2$*-ev[OF True]* **by** *auto*

**define** *f* **where** *f x = v $h enum-verts-inv x* **for** *x*
**have** *v-alt*: $v = (\chi\ i.\ f$ *(enum-verts i))*
**unfolding** *f-def Rep-inverse* **by** *simp*

**have** *g-inner f (*$\lambda$*-. 1)* $= v \cdot 1$
**unfolding** *g-inner-conv v-alt one-vec-def* **by** *simp*
**also have** ... $= 0$ **unfolding** *v-def(1)* **by** *simp*
**finally have** *f-orth*: *g-inner f (*$\lambda$*-. 1)* $= 0$ **by** *simp*

**have** $\gamma_2$ *TYPE('n) ∗ norm v^2= v* · ($\gamma_2$ *TYPE('n) ∗s v*)
  **unfolding** *power2-norm-eq-inner* **by** (*simp add*:*algebra-simps scalar-mult-eq-scaleR*)
**also have** ... = *v* · (*A ∗v v*)
  **unfolding** *v-def* **by** *simp*
**also have** ... = *g-inner f* (*g-step f*)
  **unfolding** *v-alt g-inner-conv g-step-conv* **by** *simp*
**also have** ... ≤ $\Lambda_2$ ∗ *g-norm f^2*
  **by** (*intro os-expanderD f-orth*)
**also have** ... = $\Lambda_2$ ∗ *norm v^2*
  **unfolding** *v-alt g-norm-conv* **by** *simp*
**finally have** $\gamma_2$ *TYPE('n) ∗ norm v^2* ≤ $\Lambda_2$ ∗ *norm v^2* **by** *simp*
**hence** $\gamma_2$ *TYPE('n)* ≤ $\Lambda_2$
  **using** *v-def*(*2*) **by** *simp*
**moreover have** $\Lambda_2$ ≤ $\gamma_2$ *TYPE ('n)*
  **using** $\gamma_2$*-bound*
  **by** (*intro os-expanderI*[*OF True*])
    (*simp add*: *g-inner-conv g-step-conv g-norm-conv one-vec-def*)
**ultimately show** *?thesis* **by** *simp*

**next**
  **case** *False*
  **then show** *?thesis*
    **unfolding** $\Lambda_2$*-def* $\gamma_2$*-def* **by** *simp*

**qed**

**lemma** *expansionD2*:
  **assumes** *g-inner f* (λ-. *1*) = *0*
  **shows** *g-norm* (*g-step f*) ≤ $\Lambda_a$ ∗ *g-norm f* (**is** *?L* ≤ *?R*)

**proof** −
  **define** *v* **where** *v* = (χ *i. f* (*enum-verts i*))
  **have** *v* · *1* = *g-inner f* (λ-. *1*)
    **unfolding** *g-inner-conv v-def one-vec-def* **by** *simp*
  **also have** ... = *0* **using** *assms* **by** *simp*
  **finally have** *0*:*v* · *1* = *0* **by** *simp*
  **have** *g-norm* (*g-step f*) = *norm* (*A ∗v v*)
    **unfolding** *g-norm-conv g-step-conv v-def* **by** *auto*
  **also have** ... ≤ $\Lambda_a$ ∗ *norm v*
    **unfolding** $\Lambda_e$*-eq-*Λ **by** (*intro* $\gamma_a$*-real-bound 0*)
  **also have** ... = $\Lambda_a$ ∗ *g-norm f*
    **unfolding** *g-norm-conv v-def* **by** *simp*
  **finally show** *?thesis* **by** *simp*

**qed**

**lemma** *rayleigh-bound*:
  **fixes** *v* :: *real^'n*
  **shows** |*v* · (*A ∗v v*)| ≤ *norm v^2*

**proof** −
  **define** *f* **where** *f x* = *v* \$*h enum-verts-inv x* **for** *x*
  **have** *v-alt*: *v* = (χ *i. f* (*enum-verts i*))
    **unfolding** *f-def Rep-inverse* **by** *simp*

  **have** |*v* · (*A ∗v v*)| = |*g-inner f* (*g-step f*)|
    **unfolding** *v-alt g-inner-conv g-step-conv* **by** *simp*
  **also have** ... = |($\sum$ *a*∈*arcs G. f* (*head G a*) ∗ *f* (*tail G a*))|/*d*
    **unfolding** *g-inner-step-eq* **by** *simp*
  **also have** ... ≤ (*d* ∗ (*g-norm f*)$^2$) / *d*
    **by** (*intro divide-right-mono bdd-above-aux*) *auto*
  **also have** ... = *g-norm f^2*
    **using** *d-gt-0* **by** *simp*

**also have** ... = *norm v^2*
  **unfolding** *g-norm-conv v-alt* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

The following implies that two-sided expanders are also one-sided expanders.

**lemma** $\Lambda_2$*-range*: $|\Lambda_2| \leq \Lambda_a$
**proof** (*cases n > 1*)
  **case** *True*
  **hence** *0*:*set-mset* (*eigenvalues A* − {#*1*::*complex*#}) ≠ {}
    **using** *size-evs* **by** *auto*

  **have** $\gamma_2$ *TYPE* ('*n*) = *Max* (*Re* ' *set-mset* (*eigenvalues A* − {#*1*::*complex*#}))
    **unfolding** $\gamma_2$*-def* **using** *True* **by** *simp*
  **also have** ... ∈ *Re* ' *set-mset* (*eigenvalues A* − {#*1*::*complex*#})
    **using** *Max-in 0* **by** *simp*
  **finally have** $\gamma_2$ *TYPE* ('*n*) ∈ *Re* ' *set-mset* (*eigenvalues A* − {#*1*::*complex*#})
    **by** *simp*
  **then obtain** $\alpha$ **where** $\alpha$*-def*: $\alpha$ ∈ *set-mset* (*eigenvalues A* − {#*1*::*complex*#}) $\gamma_2$ *TYPE* ('*n*)
= *Re* $\alpha$
    **by** *auto*

  **have** $|\Lambda_2|$ = |$\gamma_2$ *TYPE* ('*n*) |
    **using** $\Lambda_2$*-eq-*$\gamma_2$ **by** *simp*
  **also have** ... = |*Re* $\alpha$|
    **using** $\alpha$*-def* **by** *simp*
  **also have** ... ≤ *cmod* $\alpha$
    **using** *abs-Re-le-cmod* **by** *simp*
  **also have** ... ≤ *Max* (*cmod* ' *set-mset* (*eigenvalues A* − {#*1*#}))
    **using** $\alpha$*-def*(*1*) **by** (*intro Max-ge*) *auto*
  **also have** ... ≤ $\gamma_a$ *TYPE*('*n*)
    **unfolding** $\gamma_a$*-def* **using** *True* **by** *simp*
  **also have** ... = $\Lambda_a$
    **using** $\Lambda_e$*-eq-*$\Lambda$ **by** *simp*
  **finally show** *?thesis* **by** *simp*
**next**
  **case** *False*
  **thus** *?thesis*
    **unfolding** $\Lambda_2$*-def* $\Lambda_a$*-def* **by** *simp*
**qed**

**end**

**lemmas** (**in** *regular-graph*) *expansionD2* =
  *regular-graph-tts.expansionD2*[*OF eg-tts-1*,
    *internalize-sort* '*n* :: *finite*, *OF* - *regular-graph-axioms*,
    *unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**lemmas** (**in** *regular-graph*) $\Lambda_2$*-range* =
  *regular-graph-tts.*$\Lambda_2$*-range*[*OF eg-tts-1*,
    *internalize-sort* '*n* :: *finite*, *OF* - *regular-graph-axioms*,
    *unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**unbundle** *no intro-cong-syntax*

**end**

# 7 Cheeger Inequality

The Cheeger inequality relates edge expansion (a combinatorial property) with the second largest eigenvalue.

**theory** *Expander-Graphs-Cheeger-Inequality*
  **imports** *Expander-Graphs-Eigenvalues*
**begin**

**unbundle** *intro-cong-syntax*
**hide-const** *Quantum.T*

**context** *regular-graph*
**begin**

**lemma** *edge-expansionD2*:
  **assumes** $m = card\ (S \cap verts\ G)\ 2{*}m \leq n$
  **shows** $\Lambda_e * m \leq real\ (card\ (edges\text{-}betw\ S\ (-S)))$
**proof** −
  **define** $S'$ **where** $S' = S \cap verts\ G$
  **have** $\Lambda_e * m = \Lambda_e * card\ S'$
    **using** *assms(1) S′-def* **by** *simp*
  **also have** ... $\leq real\ (card\ (edges\text{-}betw\ S'\ (-S')))$
    **using** *assms* **unfolding** *S′-def* **by** (*intro edge-expansionD*) *auto*
  **also have** ... $= real\ (card\ (edges\text{-}betw\ S\ (-S)))$
    **unfolding** *S′-def edges-betw-def*
    **by** (*intro arg-cong*[**where** *f=real*] *arg-cong*[**where** *f=card*]) *auto*
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *edges-betw-sym*:
  *card* (*edges-betw S T*) = *card* (*edges-betw T S*) (**is** *?L = ?R*)
**proof** −
  **have** $?L = (\sum a \in arcs\ G.\ of\text{-}bool\ (tail\ G\ a \in S \wedge head\ G\ a \in T))$
    **unfolding** *edges-betw-def of-bool-def* **by** (*simp add:sum.If-cases Int-def*)
  **also have** ... $= (\sum e \in\#\ edges\ G.\ of\text{-}bool\ (fst\ e \in S \wedge snd\ e \in T))$
    **unfolding** *sum-unfold-sum-mset edges-def arc-to-ends-def*
    **by** (*simp add:image-mset.compositionality comp-def*)
  **also have** ... $= (\sum e \in\#\ edges\ G.\ of\text{-}bool\ (snd\ e \in S \wedge fst\ e \in T))$
    **by** (*subst edges-sym*[*OF sym*, *symmetric*])
      (*simp add:image-mset.compositionality comp-def case-prod-beta*)
  **also have** ... $= (\sum a \in arcs\ G.\ of\text{-}bool\ (tail\ G\ a \in T \wedge head\ G\ a \in S))$
    **unfolding** *sum-unfold-sum-mset edges-def arc-to-ends-def*
    **by** (*simp add:image-mset.compositionality comp-def conj.commute*)
  **also have** ... = *?R*
    **unfolding** *edges-betw-def of-bool-def* **by** (*simp add:sum.If-cases Int-def*)
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *edges-betw-reg*:
  **assumes** $S \subseteq verts\ G$
  **shows** $card\ (edges\text{-}betw\ S\ UNIV) = card\ S * d$ (**is** *?L = ?R*)
**proof** −
  **have** $?L = card\ (\bigcup (out\text{-}arcs\ G\ `\ S))$
    **unfolding** *edges-betw-def out-arcs-def* **by** (*intro arg-cong*[**where** *f=card*]) *auto*
  **also have** ... $= (\sum i{\in}S.\ card\ (out\text{-}arcs\ G\ i))$
    **using** *finite-subset*[*OF assms*] **unfolding** *out-arcs-def*
    **by** (*intro card-UN-disjoint*) *auto*

```
  also have ... = (∑ i∈S. out-degree G i)
    unfolding out-degree-def by simp
  also have ... = (∑ i∈S. d)
    using assms by (intro sum.cong reg) auto
  also have ... = ?R
    by simp
  finally show ?thesis by simp
qed
```

The following proof follows Hoory et al. [4, §4.5.1].

```
lemma cheeger-aux-2:
  assumes n > 1
  shows Λ_e ≥ d∗(1−Λ_2)/2
proof −
  have real (card (edges-betw S (−S))) ≥ (d ∗ (1 − Λ_2) / 2) ∗ real (card S)
    if S ⊆ verts G 2 ∗ card S ≤ n for S
  proof −
    let ?ct = real (card (verts G − S))
    let ?cs = real (card S)

    have card (edges-betw S S)+card (edges-betw S (−S))=card(edges-betw S S∪edges-betw S (−S))
      unfolding edges-betw-def by (intro card-Un-disjoint[symmetric]) auto
    also have ... = card (edges-betw S UNIV)
      unfolding edges-betw-def by (intro arg-cong[where f=card]) auto
    also have ... = d ∗ ?cs
      using edges-betw-reg[OF that(1)] by simp
    finally have card (edges-betw S S) + card (edges-betw S (−S)) = d ∗ ?cs by simp
    hence 4: card (edges-betw S S) = d ∗ ?cs − card (edges-betw S (−S))
      by simp

    have card(edges-betw S(−S))+card(edges-betw(−S)(−S))=card(edges-betw S(−S)∪edges-betw(−S)(−S))
      unfolding edges-betw-def by (intro card-Un-disjoint[symmetric]) auto
    also have ... = card (edges-betw UNIV (verts G − S))
      unfolding edges-betw-def by (intro arg-cong[where f=card]) auto
    also have ... = card (edges-betw (verts G − S) UNIV)
      by (intro edges-betw-sym)
    also have ... = d ∗ ?ct
      using edges-betw-reg by auto
    finally have card (edges-betw S (−S)) + card (edges-betw (−S) (−S)) = d ∗ ?ct by simp
    hence 5: card (edges-betw (−S) (−S)) = d ∗ ?ct − card (edges-betw S (−S))
      by simp
    have 6: card (edges-betw (−S) S) = card (edges-betw S (−S))
      by (intro edges-betw-sym)

    have ?cs + ?ct =real (card (S ∪ (verts G− S)))
      unfolding of-nat-add[symmetric] using finite-subset[OF that(1)]
      by (intro-cong [σ_1 of-nat, σ_1 card] more:card-Un-disjoint[symmetric]) auto
    also have ... = real n
      unfolding n-def using that(1) by (intro-cong [σ_1 of-nat, σ_1 card]) auto
    finally have 7: ?cs + ?ct = n by simp

    define f where
      f x = real (card (verts G − S)) ∗ of-bool (x ∈ S) − card S ∗ of-bool (x ∉ S) for x

    have g-inner f (λ-. 1) = ?cs ∗ ?ct − real (card (verts G ∩ {x. x ∉ S})) ∗ ?cs
      unfolding g-inner-def f-def using Int-absorb1[OF that(1)] by (simp add:sum-subtractf)
    also have ... = ?cs ∗ ?ct − ?ct ∗ ?cs
      by (intro-cong [σ_2 (−), σ_2 (∗), σ_1 of-nat, σ_1 card]) auto
```

**also have** ... = *0* **by** *simp*
**finally have** *11*: *g-inner f* (λ-. *1*) = *0* **by** *simp*

**have** *g-norm f*^*2* = ($\sum$ *v*∈*verts G*. *f v*^*2*)
**unfolding** *g-norm-sq g-inner-def conjugate-real-def* **by** (*simp add*:*power2-eq-square*)
**also have** ...=($\sum$ *v*∈*verts G*. *?ct*^*2*∗(*of-bool* (*v* ∈ *S*))$^2$)+($\sum$ *v*∈*verts G*. *?cs*^*2*∗(*of-bool* (*v* ∉ *S*))$^2$)
**unfolding** *f-def power2-diff* **by** (*simp add*:*sum.distrib sum-subtractf power-mult-distrib*)
**also have** ... = *real* (*card* (*verts G* ∩ *S*))∗*?ct*^*2* + *real* (*card* (*verts G* ∩ {*v*. *v* ∉ *S*})) ∗ *?cs*^*2*
**unfolding** *of-bool-def* **by** (*simp add*:*if-distrib if-distribR sum.If-cases*)
**also have** ... = *real*(*card S*)∗(*real*(*card*(*verts G*−*S*)))$^2$ + *real*(*card*(*verts G*−*S*))∗(*real*(*card S*))$^2$
**using** *that*(*1*) **by** (*intro-cong* [$\sigma_2$(+), $\sigma_2$ (∗), $\sigma_2$ *power*, $\sigma_1$ *of-nat*, $\sigma_1$ *card*]) *auto*
**also have** ... = *real*(*card S*)∗*real* (*card* (*verts G* −*S*))∗(*?cs* + *?ct*)
**by** (*simp add*:*power2-eq-square algebra-simps*)
**also have** ... = *real*(*card S*)∗*real* (*card* (*verts G* −*S*))∗*n*
**unfolding** *7* **by** *simp*
**finally have** *9*: *g-norm f*^*2* = *real*(*card S*)∗*real* (*card* (*verts G* −*S*))∗*real n* **by** *simp*

**have** ($\sum$ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*)) =
(*card* (*edges-betw S S*) ∗ *?ct*∗*?ct*) + (*card* (*edges-betw* (−*S*) (−*S*)) ∗ *?cs*∗*?cs*) −
(*card* (*edges-betw S* (−*S*)) ∗ *?ct*∗*?cs*) − (*card* (*edges-betw* (−*S*) *S*) ∗ *?cs*∗*?ct*)
**unfolding** *f-def* **by** (*simp add*:*of-bool-def algebra-simps Int-def if-distrib if-distribR edges-betw-def sum.If-cases*)
**also have** ... = *d*∗*?cs*∗*?ct*∗(*?cs*+*?ct*) − *card* (*edges-betw S* (−*S*))∗(*?ct*∗*?ct*+*2*∗*?ct*∗*?cs*+*?cs*∗*?cs*)
**unfolding** *4 5 6* **by** (*simp add*:*algebra-simps*)
**also have** ... = *d*∗*?cs*∗*?ct*∗*n* − (*?ct*+*?cs*)^*2* ∗ *card* (*edges-betw S* (−*S*))
**unfolding** *power2-diff 7 power2-sum* **by** (*simp add*:*ac-simps power2-eq-square*)
**also have** ... = *d* ∗*?cs*∗*?ct*∗*n* − *n*^*2* ∗ *card* (*edges-betw S* (−*S*))
**using** *7* **by** (*simp add*:*algebra-simps*)
**finally have** *8*:($\sum$ *a* ∈ *arcs G*. *f*(*head G a*)∗*f*(*tail G a*))=*d*∗*?cs*∗*?ct*∗*n*−*n*^*2*∗*card*(*edges-betw S* (−*S*))
**by** *simp*

**have** *d*∗*?cs*∗*?ct*∗*n*−*n*^*2*∗*card*(*edges-betw S* (−*S*)) = ($\sum$ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*))
**unfolding** *8* **by** *simp*
**also have** ... ≤ *d* ∗ (*g-inner f* (*g-step f*))
**unfolding** *g-inner-step-eq* **using** *d-gt-0*
**by** *simp*
**also have** ... ≤ *d* ∗ ($\Lambda_2$ ∗ *g-norm f*^*2*)
**by** (*intro mult-left-mono os-expanderD 11*) *auto*
**also have** ... = *d* ∗ $\Lambda_2$ ∗ *?cs*∗*?ct*∗*n*
**unfolding** *9* **by** *simp*
**finally have** *d*∗*?cs*∗*?ct*∗*n*−*n*^*2*∗*card*(*edges-betw S* (−*S*)) ≤ *d* ∗ $\Lambda_2$ ∗ *?cs*∗*?ct*∗*n*
**by** *simp*
**hence** *n* ∗ *n* ∗ *card* (*edges-betw S* (−*S*)) ≥ *n* ∗ (*d* ∗ *?cs* ∗ *?ct* ∗ (*1*−$\Lambda_2$))
**by** (*simp add*:*power2-eq-square algebra-simps*)
**hence** *10*:*n* ∗ *card* (*edges-betw S* (−*S*)) ≥ *d* ∗ *?cs* ∗ *?ct* ∗ ( *1*−$\Lambda_2$)
**using** *n-gt-0* **by** *simp*

**have** (*d* ∗ (*1* − $\Lambda_2$) / *2*) ∗ *?cs* = (*d* ∗ (*1*−$\Lambda_2$) ∗ (*1* − *1* / *2*)) ∗ *?cs*
**by** *simp*
**also have** ... ≤ *d* ∗ (*1*−$\Lambda_2$) ∗ ((*n* − *?cs*) / *n*) ∗ *?cs*
**using** *that n-gt-0* $\Lambda_2$*-le-1*
**by** (*intro mult-left-mono mult-right-mono mult-nonneg-nonneg*) *auto*
**also have** ... = (*d* ∗ (*1*−$\Lambda_2$) ∗ *?ct* / *n*) ∗ *?cs*
**using** *7* **by** *simp*
**also have** ... = *d* ∗ *?cs* ∗ *?ct* ∗ (*1*−$\Lambda_2$) / *n*

```
      by simp
    also have ... ≤ n ∗ card (edges-betw S (−S)) / n
      by (intro divide-right-mono 10) auto
    also have ... = card (edges-betw S (−S))
      using n-gt-0 by simp
    finally show ?thesis by simp
  qed
  thus ?thesis
    by (intro edge-expansionI assms) auto
qed

end

lemma surj-onI:
  assumes ⋀x. x ∈ B ⟹ g x ∈ A ∧ f (g x) = x
  shows B ⊆ f ‘ A
  using assms by force

lemma find-sorted-bij-1:
  fixes g :: 'a ⇒ ('b :: linorder)
  assumes finite S
  shows ∃f. bij-betw f {..<card S} S ∧ mono-on {..<card S} (g∘ f)
proof −
  define h where h x = from-nat-into S x for x

  have h-bij:bij-betw h {..<card S} S
    unfolding h-def using bij-betw-from-nat-into-finite[OF assms] by simp

  define xs where xs = sort-key (g ∘ h) [0..<card S]
  define f where f i = h (xs ! i) for i

  have l-xs: length xs = card S
    unfolding xs-def by auto
  have set-xs: set xs = {..<card S}
    unfolding xs-def by auto
  have dist-xs: distinct xs
    using l-xs set-xs by (intro card-distinct) simp
  have sorted-xs: sorted (map (g ∘ h) xs)
    unfolding xs-def using sorted-sort-key by simp

  have (λi. xs ! i) ‘ {..<card S} = set xs
    using l-xs by (auto simp:in-set-conv-nth)
  also have ... = {..<card S}
    unfolding set-xs by simp
  finally have set-xs':
    (λi. xs ! i) ‘ {..<card S} = {..<card S} by simp

  have f ‘ {..<card S} = h ‘ ((λi. xs ! i) ‘ {..<card S})
    unfolding f-def image-image by simp
  also have ... = h ‘ {..<card S}
    unfolding set-xs' by simp
  also have ... = S
    using bij-betw-imp-surj-on[OF h-bij] by simp
  finally have 0: f ‘ {..<card S} = S by simp

  have inj-on ((!) xs) {..<card S}
    using dist-xs l-xs unfolding distinct-conv-nth
    by (intro inj-onI) auto
```

**hence** *inj-on* $(h \circ (\lambda i.\ xs\ !\ i))$ $\{..<card\ S\}$
  **using** *set-xs′ bij-betw-imp-inj-on*[*OF h-bij*]
  **by** (*intro comp-inj-on*) *auto*
**hence** *1*: *inj-on f* $\{..<card\ S\}$
  **unfolding** *f-def comp-def* **by** *simp*
**have** *2*: *mono-on* $\{..<card\ S\}$ $(g \circ f)$
  **using** *sorted-nth-mono*[*OF sorted-xs*] *l-xs* **unfolding** *f-def*
  **by** (*intro mono-onI*) *simp*
**thus** *?thesis*
  **using** *0 1 2* **unfolding** *bij-betw-def* **by** *auto*
**qed**

**lemma** *find-sorted-bij-2*:
**fixes** $g$ :: $'a \Rightarrow ('b$ :: *linorder*)
**assumes** *finite S*
**shows** $\exists f.$ *bij-betw f S* $\{..<card\ S\} \wedge (\forall x\ y.\ x \in S \wedge y \in S \wedge f\ x < f\ y \longrightarrow g\ x \leq g\ y)$
**proof** −
**obtain** $f$ **where** *f-def*: *bij-betw f* $\{..<card\ S\}$ $S$ *mono-on* $\{..<card\ S\}$ $(g \circ f)$
  **using** *find-sorted-bij-1* [*OF assms*] **by** *auto*

**define** $h$ **where** $h =$ *the-inv-into* $\{..<card\ S\}$ $f$
**have** *bij-h*: *bij-betw h S* $\{..<card\ S\}$
  **unfolding** *h-def* **by** (*intro bij-betw-the-inv-into f-def*)

**moreover have** $g\ x \leq g\ y$ **if** $h\ x < h\ y$ $x \in S$ $y \in S$ **for** $x\ y$
**proof** −
  **have** $h\ y <$ *card* $S$ $h\ x <$ *card* $S$ $h\ x \leq h\ y$
    **using** *bij-betw-apply*[*OF bij-h*] *that* **by** *auto*
  **hence** $g\ (f\ (h\ x)) \leq g\ (f\ (h\ y))$
    **using** *f-def*(*2*) **unfolding** *mono-on-def* **by** *simp*
  **moreover have** $f$ ‘ $\{..<card\ S\} = S$
    **using** *bij-betw-imp-surj-on*[*OF f-def*(*1*)] **by** *simp*
  **ultimately show** $g\ x \leq g\ y$
    **unfolding** *h-def* **using** *that f-the-inv-into-f*[*OF bij-betw-imp-inj-on*[*OF f-def*(*1*)]]
    **by** *auto*
**qed**
**ultimately show** *?thesis* **by** *auto*
**qed**

**context** *regular-graph-tts*
**begin**

Normalized Laplacian of the graph

**definition** $L$ **where** $L =$ *mat 1* $- A$

**lemma** *L-pos-semidefinite*:
**fixes** $v$ :: *real* ^′*n*
**shows** $v \cdot (L *v\ v) \geq 0$
**proof** −
**have** $0 = v \cdot v -$ *norm* $v$^*2* **unfolding** *power2-norm-eq-inner* **by** *simp*
**also have** ... $\leq v \cdot v -$ *abs* $(v \cdot (A *v\ v))$
  **by** (*intro diff-mono rayleigh-bound*) *auto*
**also have** ... $\leq v \cdot v - v \cdot (A *v\ v)$
  **by** (*intro diff-mono*) *auto*
**also have** ... $= v \cdot (L *v\ v)$
  **unfolding** *L-def* **by** (*simp add*:*algebra-simps*)
**finally show** *?thesis* **by** *simp*
**qed**

The following proof follows Hoory et al. [4, §4.5.2].

**lemma** *cheeger-aux-1*:
  **assumes** $n > 1$
  **shows** $\Lambda_e \le d * sqrt\ (2 * (1 - \Lambda_2))$
**proof** −
  **obtain** $v$ **where** *v-def*: $v \cdot 1 = 0$ $v \neq 0$ $A *v\ v = \Lambda_2 *s\ v$
    **using** $\Lambda_2$-*eq*-$\gamma_2$ $\gamma_2$-*ev*[*OF assms*] **by** *auto*

  **have** *False* **if** $2*card\ \{i.\ (1 *s\ v)\ \$h\ i > 0\} > n$ $2*card\ \{i.\ ((-1) *s\ v)\ \$h\ i > 0\} > n$
  **proof** −
    **have** $2 * n = n + n$ **by** *simp*
    **also have** ... $< 2 * card\ \{i.\ (1 *s\ v)\ \$h\ i > 0\} + 2 * card\ \{i.\ ((-1) *s\ v)\ \$h\ i > 0\}$
      **by** (*intro add-strict-mono that*)
    **also have** ... $= 2 * (card\ \{i.\ (1 *s\ v)\ \$h\ i > 0\} + card\ \{i.\ ((-1) *s\ v)\ \$h\ i > 0\})$
      **by** *simp*
    **also have** ... $= 2 * (card\ (\{i.\ (1 *s\ v)\ \$h\ i > 0\} \cup \{i.\ ((-1) *s\ v)\ \$h\ i > 0\}))$
      **by** (*intro arg-cong2*[**where** $f=(*)$] *card-Un-disjoint*[*symmetric*]) *auto*
    **also have** ... $\le 2 * (card\ (UNIV :: 'n\ set))$
      **by** (*intro mult-left-mono card-mono*) *auto*
    **finally have** $2 * n < 2 * n$
      **unfolding** *n-def card-n* **by** *auto*
    **thus** *?thesis* **by** *simp*
  **qed**
  **then obtain** $\beta$ :: *real* **where** $\beta$-*def*: $\beta = 1 \vee \beta = (-1)$ $2*\ card\ \{i.\ (\beta *s\ v)\ \$h\ i > 0\ \} \le n$
    **unfolding** *not-le*[*symmetric*] **by** *blast*

  **define** $g$ **where** $g = \beta *s\ v$

  **have** *g-orth*: $g \cdot 1 = 0$ **unfolding** *g-def* **using** *v-def*(*1*)
    **by** (*simp add*: *scalar-mult-eq-scaleR*)
  **have** *g-nz*: $g \neq 0$
    **unfolding** *g-def* **using** $\beta$-*def*(*1*) *v-def*(*2*) **by** *auto*
  **have** *g-ev*: $A *v\ g = \Lambda_2 *s\ g$
    **unfolding** *g-def scalar-mult-eq-scaleR matrix-vector-mult-scaleR v-def*(*3*) **by** *auto*
  **have** *g-supp*: $2 * card\ \{\ i.\ g\ \$h\ i > 0\ \} \le n$
    **unfolding** *g-def* **using** $\beta$-*def*(*2*) **by** *auto*

  **define** $f$ **where** $f = (\chi\ i.\ max\ (g\ \$h\ i)\ 0)$

  **have** $(L *v\ f)\ \$h\ i \le (1 - \Lambda_2) * g\ \$h\ i$ (**is** $?L \le ?R$) **if** $g\ \$h\ i > 0$ **for** $i$
  **proof** −
    **have** $?L = f\ \$h\ i - (A *v\ f)\ \$h\ i$
      **unfolding** *L-def* **by** (*simp add*:*algebra-simps*)
    **also have** ... $= g\ \$h\ i - (\sum j \in UNIV.\ A\ \$h\ i\ \$h\ j * f\ \$h\ j)$
      **unfolding** *matrix-vector-mult-def f-def* **using** *that* **by** *auto*
    **also have** ... $\le g\ \$h\ i - (\sum j \in UNIV.\ A\ \$h\ i\ \$h\ j * g\ \$h\ j)$
      **unfolding** *f-def A-def* **by** (*intro diff-mono sum-mono mult-left-mono*) *auto*
    **also have** ... $= g\ \$h\ i - (A *v\ g)\ \$h\ i$
      **unfolding** *matrix-vector-mult-def* **by** *simp*
    **also have** ... $= (1 - \Lambda_2) * g\ \$h\ i$
      **unfolding** *g-ev* **by** (*simp add*:*algebra-simps*)
    **finally show** *?thesis* **by** *simp*
  **qed**
  **moreover have** $f\ \$h\ i \neq 0 \Longrightarrow g\ \$h\ i > 0$ **for** $i$
    **unfolding** *f-def* **by** *simp*
  **ultimately have** $0$:$(L *v\ f)\ \$h\ i \le (1 - \Lambda_2) * g\ \$h\ i \vee f\ \$h\ i = 0$ **for** $i$
    **by** *auto*

Part (i) in Hoory et al. (§4.5.2) but the operator L here is normalized.

**have** $f \cdot (L *v\ f) = (\sum i{\in}UNIV.\ (L *v\ f)\ \$h\ i * f\ \$h\ i)$
  **unfolding** *inner-vec-def* **by** (*simp add*:*ac-simps*)
**also have** ... $\le (\sum i{\in}UNIV.\ ((1{-}\Lambda_2) * g\ \$h\ i) * f\ \$h\ i)$
  **by** (*intro sum-mono mult-right-mono′ 0*) (*simp add*:*f-def*)
**also have** ... $= (\sum i{\in}UNIV.\ (1{-}\Lambda_2) * f\ \$h\ i * f\ \$h\ i)$
  **unfolding** *f-def* **by** (*intro sum.cong refl*) *auto*
**also have** ... $= (1{-}\Lambda_2) * (f \cdot f)$
  **unfolding** *inner-vec-def* **by** (*simp add*:*sum-distrib-left ac-simps*)
**also have** ... $= (1 - \Lambda_2) * norm\ f\hat{\ }2$
  **by** (*simp add*: *power2-norm-eq-inner*)
**finally have** *h-part-i*: $f \cdot (L *v\ f) \le (1 - \Lambda_2) * norm\ f\hat{\ }2$ **by** *simp*

**define** $f'$ **where** $f'\ x = f\ \$h\ (enum\text{-}verts\text{-}inv\ x)$ **for** $x$
**have** $f'$-*alt*: $f = (\chi\ i.\ f'\ (enum\text{-}verts\ i))$
  **unfolding** $f'$-*def Rep-inverse* **by** *simp*

**define** $B_f$ **where** $B_f = (\sum a{\in}arcs\ G.\ |f'\ (tail\ G\ a)\hat{\ }2 - f'\ (head\ G\ a)\hat{\ }2|)$

**have** $(x + y)\hat{\ }2 \le 2 * (x\hat{\ }2 + y\hat{\ }2)$ **for** $x\ y :: real$
**proof** −
  **have** $(x + y)\hat{\ }2 = (x\hat{\ }2 + y\hat{\ }2) + 2 * x * y$
    **unfolding** *power2-sum* **by** *simp*
  **also have** ... $\le (x\hat{\ }2 + y\hat{\ }2) + (x\hat{\ }2 + y\hat{\ }2)$
    **by** (*intro add-mono sum-squares-bound*) *auto*
  **finally show** *?thesis* **by** *simp*
**qed**
**hence** $(\sum a{\in}arcs\ G.(f'(tail\ G\ a){+}\ f'(head\ G\ a))^2){\le}(\sum a{\in}arcs\ G.\ 2{*}(f'(tail\ G\ a)\hat{\ }2{+}f'(head\ G\ a)\hat{\ }2))$
  **by** (*intro sum-mono*) *auto*
**also have** ... $= 2{*}((\sum a{\in}arcs\ G.\ f'(tail\ G\ a)\hat{\ }2) + (\sum a{\in}arcs\ G.\ f'(head\ G\ a)\hat{\ }2))$
  **by** (*simp add*:*sum-distrib-left*)
**also have** ... $= 4 * d * g\text{-}norm\ f'\hat{\ }2$
  **unfolding** *sum-arcs-tail*[**where** $f{=}\lambda x.\ f'\ x\hat{\ }2$] *sum-arcs-head*[**where** $f{=}\lambda x.\ f'\ x\hat{\ }2$]
    *g-norm-sq g-inner-def* **by** (*simp add*:*power2-eq-square*)
**also have** ... $= 4 * d * norm\ f\hat{\ }2$
  **unfolding** *g-norm-conv* $f'$-*alt* **by** *simp*
**finally have** *1*: $(\sum i{\in}arcs\ G.\ (f'\ (tail\ G\ i) + f'\ (head\ G\ i))^2) \le 4{*}d{*}\ norm\ f\hat{\ }2$
  **by** *simp*

**have** $(\sum a{\in}arcs\ G.\ (f'\ (tail\ G\ a) - f'\ (head\ G\ a))^2) = (\sum a{\in}arcs\ G.\ (f'\ (tail\ G\ a))^2) +$
  $(\sum a{\in}arcs\ G.\ (f'\ (head\ G\ a))^2) - 2{*}\ (\sum a{\in}arcs\ G.\ f'\ (tail\ G\ a) * f'\ (head\ G\ a))$
  **unfolding** *power2-diff* **by** (*simp add*:*sum-subtractf sum-distrib-left ac-simps*)
**also have** ... $= 2 * (d * (\sum v{\in}verts\ G.\ (f'\ v)^2) - (\sum a{\in}arcs\ G.\ f'\ (tail\ G\ a) * f'\ (head\ G\ a)))$
  **unfolding** *sum-arcs-tail*[**where** $f{=}\lambda x.\ f'\ x\hat{\ }2$] *sum-arcs-head*[**where** $f{=}\lambda x.\ f'\ x\hat{\ }2$] **by** *simp*
**also have** ... $= 2 * (d * g\text{-}inner\ f'\ f' - d * g\text{-}inner\ f'\ (g\text{-}step\ f'))$
  **unfolding** *g-inner-step-eq* **using** *d-gt-0*
  **by** (*intro-cong* $[\sigma_2\ (*),\ \sigma_2\ (-)]$) (*auto simp*:*power2-eq-square g-inner-def ac-simps*)
**also have** ... $= 2 * d * (g\text{-}inner\ f'\ f' - g\text{-}inner\ f'\ (g\text{-}step\ f'))$
  **by** (*simp add*:*algebra-simps*)
**also have** ... $= 2 * d * (f \cdot f - f \cdot (A *v\ f))$
  **unfolding** *g-inner-conv g-step-conv* $f'$-*alt* **by** *simp*
**also have** ... $= 2 * d * (f \cdot (L *v\ f))$
  **unfolding** *L-def* **by** (*simp add*:*algebra-simps*)
**finally have** *2*:$(\sum a{\in}arcs\ G.\ (f'\ (tail\ G\ a) - f'\ (head\ G\ a))^2) = 2 * d * (f \cdot (L *v\ f))$ **by** *simp*

**have** $B_f = (\sum a{\in}arcs\ G.\ |f'\ (tail\ G\ a){+}f'\ (head\ G\ a)|{*}|f'\ (tail\ G\ a){-}f'\ (head\ G\ a)|)$
  **unfolding** $B_f$-*def abs-mult*[*symmetric*] **by** (*simp add*:*algebra-simps power2-eq-square*)

**also have** ...$\leq$ *L2-set* ($\lambda a.\ f'(tail\ G\ a) + f'(head\ G\ a)$) (*arcs G*) $*$
*L2-set* ($\lambda a.\ f'\ (tail\ G\ a) - f'(head\ G\ a)$) (*arcs G*)
**by** (*intro L2-set-mult-ineq*)
**also have** ... $\leq$ *sqrt* ($4{*}d{*}$ *norm* $f\hat{\ }2$) $*$ *sqrt* ($2 * d * (f \cdot (L *v\ f))$)
**unfolding** *L2-set-def 2*
**by** (*intro mult-right-mono iffD2*[*OF real-sqrt-le-iff*] *1 real-sqrt-ge-zero mult-nonneg-nonneg L-pos-semidefinite*) *auto*
**also have** ... $= 2 *$ *sqrt* $2 * d *$ *norm* $f *$ *sqrt* ($f \cdot (L *v\ f)$)
**by** (*simp add:real-sqrt-mult*)
**finally have** *hoory-4-12*: $B_f \leq 2 *$ *sqrt* $2 * d *$ *norm* $f *$ *sqrt* ($f \cdot (L *v\ f)$)
**by** *simp*

The last statement corresponds to Lemma 4.12 in Hoory et al.

**obtain** $\varrho$ :: $'a \Rightarrow nat$ **where** *$\varrho$-bij*: *bij-betw* $\varrho$ (*verts G*) $\{..<n\}$ **and**
*$\varrho$-dec*: $\bigwedge x\ y.\ x \in verts\ G \Longrightarrow y \in verts\ G \Longrightarrow \varrho\ x < \varrho\ y \Longrightarrow f'\ x \geq f'\ y$
**unfolding** *n-def*
**using** *find-sorted-bij-2*[**where** *S=verts G* **and** $g=(\lambda x.\ -\ f'\ x)$] **by** *auto*

**define** $\varphi$ **where** $\varphi =$ *the-inv-into* (*verts G*) $\varrho$
**have** *$\varphi$-bij*: *bij-betw* $\varphi$ $\{..<n\}$ (*verts G*)
**unfolding** *$\varphi$-def* **by** (*intro bij-betw-the-inv-into $\varrho$-bij*)

**have** *edges* $G = \{\#\ e \in\#\ edges\ G\ .\ \varrho(fst\ e) \neq \varrho(snd\ e) \vee \varrho(fst\ e) = \varrho(snd\ e)\ \#\}$
**by** *simp*
**also have** ... $= \{\#\ e \in\#\ edges\ G\ .\ \varrho(fst\ e) \neq \varrho(snd\ e)\ \#\} + \{\#e{\in}\#edges\ G.\ \varrho(fst\ e){=}\varrho(snd\ e)\#\}$
**by** (*simp add:filter-mset-ex-predicates*)
**also have** ...$=\{\#\ e{\in}\#edges\ G.\ \varrho(fst\ e){<}\varrho(snd\ e){\vee}\varrho(fst\ e){>}\varrho(snd\ e)\#\}{+}\{\#e{\in}\#edges\ G.\ fst\ e{=}snd\ e\#\}$
**using** *bij-betw-imp-inj-on*[*OF $\varrho$-bij*] *edge-set*
**by** (*intro arg-cong2*[**where** $f{=}(+)$] *filter-mset-cong refl inj-on-eq-iff*[**where** *A=verts G*])
*auto*
**also have** ... $= \{\#e \in\#\ edges\ G.\ \varrho(fst\ e) < \varrho\ (snd\ e)\ \#\} +$
$\{\#e \in\#\ edges\ G.\ \varrho(fst\ e) > \varrho\ (snd\ e)\ \#\} +$
$\{\#e \in\#\ edges\ G.\ fst\ e = snd\ e\ \#\}$
**by** (*intro arg-cong2*[**where** $f{=}(+)$] *filter-mset-ex-predicates*[*symmetric*]) *auto*
**finally have** *edges-split*: *edges* $G = \{\#e \in\#\ edges\ G.\ \varrho(fst\ e) < \varrho\ (snd\ e)\ \#\} +$
$\{\#e \in\#\ edges\ G.\ \varrho(fst\ e) > \varrho\ (snd\ e)\ \#\} + \{\#e \in\#\ edges\ G.\ fst\ e = snd\ e\ \#\}$
**by** *simp*

**have** *$\varrho$-lt-n*: $\varrho\ x < n$ **if** $x \in verts\ G$ **for** $x$
**using** *bij-betw-apply*[*OF $\varrho$-bij*] *that* **by** *auto*

**have** *$\varphi$-$\varrho$-inv*: $\varphi\ (\varrho\ x) = x$ **if** $x \in verts\ G$ **for** $x$
**unfolding** *$\varphi$-def* **using** *bij-betw-imp-inj-on*[*OF $\varrho$-bij*]
**by** (*intro the-inv-into-f-f that*) *auto*

**have** *$\varrho$-$\varphi$-inv*: $\varrho\ (\varphi\ x) = x$ **if** $x < n$ **for** $x$
**unfolding** *$\varphi$-def* **using** *bij-betw-imp-inj-on*[*OF $\varrho$-bij*] *bij-betw-imp-surj-on*[*OF $\varrho$-bij*] *that*
**by** (*intro f-the-inv-into-f*) *auto*

**define** $\tau$ **where** $\tau\ x = (if\ x < n\ then\ f'\ (\varphi\ x)\ else\ 0)$ **for** $x$

**have** *$\tau$-nonneg*: $\tau\ k \geq 0$ **for** $k$
**unfolding** *$\tau$-def $f'$-def f-def* **by** *auto*

**have** *$\tau$-antimono*: $\tau\ k \geq \tau\ l$ **if** $k < l$ **for** $k\ l$
**proof** (*cases* $l \geq n$)

  **case** *True*
  **hence** $\tau\ l = 0$ **unfolding** $\tau$*-def* **by** *simp*
  **then show** *?thesis* **using** $\tau$*-nonneg* **by** *simp*
**next**
  **case** *False*
  **hence** $\tau\ l = f'\ (\varphi\ l)$
    **unfolding** $\tau$*-def* **by** *simp*
  **also have** ... $\leq f'\ (\varphi\ k)$
    **using** $\varrho$-$\varphi$*-inv False that*
    **by** (*intro* $\varrho$*-dec bij-betw-apply*[*OF* $\varphi$*-bij*]) *auto*
  **also have** ... $= \tau\ k$
    **unfolding** $\tau$*-def* **using** *False that* **by** *simp*
  **finally show** *?thesis* **by** *simp*
**qed**

**define** $m$ :: *nat* **where** $m = Min\ \{i.\ \tau\ i = 0 \wedge i \leq n\}$

**have** $\tau\ n = 0$
  **unfolding** $\tau$*-def* **by** *simp*
**hence** $m \in \{i.\ \tau\ i = 0 \wedge i \leq n\}$
  **unfolding** *m-def* **by** (*intro Min-in*) *auto*

**hence** *m-rel-1*: $\tau\ m = 0$ **and** *m-le-n*: $m \leq n$ **by** *auto*

**have** $\tau\ k > 0$ **if** $k < m$ **for** $k$
**proof** (*rule ccontr*)
  **assume** $\neg(\tau\ k > 0)$
  **hence** $\tau\ k = 0$
    **by** (*intro order-antisym* $\tau$*-nonneg*) *simp*
  **hence** $k \in \{i.\ \tau\ i = 0 \wedge i \leq n\}$
    **using** *that m-le-n* **by** *simp*
  **hence** $m \leq k$
    **unfolding** *m-def* **by** (*intro Min-le*) *auto*
  **thus** *False* **using** *that* **by** *simp*
**qed**
**hence** *m-rel-2*: $f'\ x > 0$ **if** $x \in \varphi$ ‘ $\{..<m\}$ **for** $x$
  **unfolding** $\tau$*-def* **using** *m-le-n that* **by** *auto*

**have** $2 * m = 2 * card\ \{..<m\}$ **by** *simp*
**also have** ... $= 2 * card\ (\varphi$ ‘ $\{..<m\})$
  **using** *m-le-n inj-on-subset*[*OF bij-betw-imp-inj-on*[*OF* $\varphi$*-bij*]]
  **by** (*intro-cong* [$\sigma_2$ $(*)$] *more*:*card-image*[*symmetric*]) *auto*
**also have** ... $\leq 2 * card\ \{x \in verts\ G.\ f'\ x > 0\}$
  **using** *m-rel-2 bij-betw-apply*[*OF* $\varphi$*-bij*] *m-le-n*
  **by** (*intro mult-left-mono card-mono subsetI*) *auto*
**also have** ... $= 2 * card\ (enum\text{-}verts\text{-}inv$ ‘ $\{x \in verts\ G.\ f\ \$h\ (enum\text{-}verts\text{-}inv\ x) > 0\})$
  **unfolding** $f'$*-def* **using** *Abs-inject*
  **by** (*intro arg-cong2*[**where** $f{=}(*)$] *card-image*[*symmetric*] *inj-onI*) *auto*
**also have** ... $= 2 * card\ \{x.\ f\ \$h\ x > 0\}$
  **using** *Rep-inverse Rep-range* **unfolding** $f'$*-def* **by** (*intro-cong* [$\sigma_2$ $(*)$, $\sigma_1$ *card*]
      *more*:*subset-antisym image-subsetI surj-onI*[**where** *g*=*enum-verts*]) *auto*
**also have** ... $= 2 * card\ \{x.\ g\ \$h\ x > 0\}$
  **unfolding** *f-def* **by** (*intro-cong* [$\sigma_2$ $(*)$, $\sigma_1$ *card*]) *auto*
**also have** ... $\leq n$
  **by** (*intro g-supp*)
**finally have** *m2-le-n*: $2{*}m \leq n$ **by** *simp*

**have** $\tau\ k \leq 0$ **if** $k > m$ **for** $k$

**using** *m-rel-1 $\tau$-antimono that* **by** *metis*
**hence** $\tau\ k \leq 0$ **if** $k \geq m$ **for** $k$
**using** *m-rel-1 that* **by** (*cases* $k > m$) *auto*
**hence** *$\tau$-supp*: $\tau\ k = 0$ **if** $k \geq m$ **for** $k$
**using** *that* **by** (*intro order-antisym $\tau$-nonneg*) *auto*

**have** *4*: $\varrho\ v \leq x \longleftrightarrow v \in \varphi$ ‘ $\{..x\}$ **if** $v \in$ *verts* $G\ x < n$ **for** $v\ x$
**proof** −
**have** $\varrho\ v \leq x \longleftrightarrow \varrho\ v \in \{..x\}$
**by** *simp*
**also have** ... $\longleftrightarrow \varphi\ (\varrho\ v) \in \varphi$ ‘ $\{..x\}$
**using** *bij-betw-imp-inj-on*[*OF $\varphi$-bij*] *bij-betw-apply*[*OF $\varrho$-bij*] *that*
**by** (*intro inj-on-image-mem-iff*[**where** $B=\{..<n\}$, *symmetric*]) *auto*
**also have** ... $\longleftrightarrow v \in \varphi$ ‘ $\{..x\}$
**unfolding** *$\varphi$-$\varrho$-inv*[*OF that*(*1*)] **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**have** $B_f = (\sum a{\in}arcs\ G.\ |f'\ (tail\ G\ a)\hat{\ }2 - f'\ (head\ G\ a)\hat{\ }2|)$
**unfolding** *$B_f$-def* **by** *simp*
**also have** ... $= (\sum e \in\#\ edges\ G.\ |f'\ (fst\ e)\hat{\ }2 - f'\ (snd\ e)\hat{\ }2|)$
**unfolding** *edges-def arc-to-ends-def sum-unfold-sum-mset*
**by** (*simp add:image-mset.compositionality comp-def*)
**also have** ... =
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (fst\ e) < \varrho\ (snd\ e)\#\}.\ |(f'\ (fst\ e))^2 - (f'\ (snd\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (snd\ e) < \varrho\ (fst\ e)\#\}.\ |(f'\ (fst\ e))^2 - (f'\ (snd\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ fst\ e = snd\ e\#\}.\ |(f'\ (fst\ e))^2 - (f'\ (snd\ e))^2|)$
**by** (*subst edges-split*) *simp*
**also have** ... =
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (snd\ e) < \varrho\ (fst\ e)\#\}.\ |(f'\ (fst\ e))^2 - (f'\ (snd\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (snd\ e) < \varrho\ (fst\ e)\#\}.\ |(f'\ (snd\ e))^2 - (f'\ (fst\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ fst\ e = snd\ e\#\}.\ |(f'\ (fst\ e))^2 - (f'\ (snd\ e))^2|)$
**by** (*subst edges-sym*[*OF sym*, *symmetric*]) (*simp add*:*image-mset.compositionality comp-def image-mset-filter-mset-swap*[*symmetric*] *case-prod-beta*)
**also have** ... =
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (snd\ e) < \varrho\ (fst\ e)\#\}.\ |(f'\ (snd\ e))^2 - (f'\ (fst\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ \varrho\ (snd\ e) < \varrho\ (fst\ e)\#\}.\ |(f'\ (snd\ e))^2 - (f'\ (fst\ e))^2|) +$
$(\sum e{\in}\#\{\#e \in\#\ edges\ G.\ fst\ e = snd\ e\#\}.\ 0)$
**by** (*intro-cong* [$\sigma_2$ (+), $\sigma_1$ *sum-mset*] *more*:*image-mset-cong*) *auto*
**also have** ... $= 2 * (\sum e{\in}\#\{\#e{\in}\#edges\ G.\ \varrho(snd\ e){<}\varrho(fst\ e)\#\}.\ |(f'\ (snd\ e))^2 - (f'\ (fst\ e))^2|)$
**by** *simp*
**also have** ... $= 2 *(\sum a|a{\in}arcs\ G{\wedge}\varrho(tail\ G\ a){>}\varrho(head\ G\ a).\ |f'(head\ G\ a)\hat{\ }2 - f'(tail\ G\ a)\hat{\ }2|)$
**unfolding** *edges-def arc-to-ends-def sum-unfold-sum-mset*
**by** (*simp add*:*image-mset.compositionality comp-def image-mset-filter-mset-swap*[*symmetric*])
**also have** ... $= 2 *$
$(\sum a|a{\in}arcs\ G{\wedge}\varrho(tail\ G\ a){>}\varrho(head\ G\ a).\ |\tau(\varrho(head\ G\ a))\hat{\ }2 - \tau(\varrho(tail\ G\ a))\hat{\ }2|)$
**unfolding** *$\tau$-def* **using** *$\varphi$-$\varrho$-inv $\varrho$-lt-n*
**by** (*intro arg-cong2*[**where** $f=(*)$] *sum.cong refl*) *auto*
**also have** ... $= 2 * (\sum a|a{\in}arcs\ G{\wedge}\varrho(tail\ G\ a){>}\varrho(head\ G\ a).\ \tau(\varrho(head\ G\ a))\hat{\ }2 - \tau(\varrho(tail\ G\ a))\hat{\ }2)$
**using** *$\tau$-antimono power-mono $\tau$-nonneg*
**by** (*intro arg-cong2*[**where** $f=(*)$] *sum.cong refl abs-of-nonneg*)(*auto*)
**also have** ... $= 2 *$
$(\sum a|a{\in}arcs\ G{\wedge}\varrho(tail\ G\ a){>}\varrho(head\ G\ a).\ (-(\tau(\varrho(tail\ G\ a))\hat{\ }2)) - (-(\tau(\varrho(head\ G\ a))\hat{\ }2)))$
**by** (*simp add*:*algebra-simps*)
**also have** ... $= 2 *(\sum a|a{\in}arcs\ G{\wedge}\varrho(tail\ G\ a){>}\varrho(head\ G\ a).$
$(\sum i{=}\varrho(head\ G\ a)..{<}\varrho(tail\ G\ a).\ (-(\tau\ (Suc\ i)\hat{\ }2)) - (-(\tau\ i\hat{\ }2))))$
**by** (*intro arg-cong2*[**where** $f=(*)$] *sum.cong refl sum-Suc-diff'*[*symmetric*]) *auto*

**also have** ...$=2*(\sum (a,\ i)\in(SIGMA\ x{:}\{a \in arcs\ G.\ \varrho\ (head\ G\ a) < \varrho\ (tail\ G\ a)\}.$
$\{\varrho\ (head\ G\ x)..<\varrho\ (tail\ G\ x)\}).\ \ \tau\ i\hat{\ }2 - \tau\ (Suc\ i)\hat{\ }2)$
**by** (*subst sum.Sigma*) *auto*
**also have** ...$=2*(\sum p\in\{(a,i).a \in arcs\ G\wedge\varrho(head\ G\ a)\leq i\wedge i<\varrho(tail\ G\ a)\}.\ \tau(snd\ p)\hat{\ }2-\tau\ (snd\ p+1)\hat{\ }2)$
**by** (*intro arg-cong2*[**where** $f=(*)$] *sum.cong refl*) (*auto simp add:Sigma-def*)
**also have** ...$=2*(\sum p\in\{(i,a).a \in arcs\ G\wedge\varrho(head\ G\ a) \leq i\wedge i < \varrho(tail\ G\ a)\}.\ \tau(fst\ p)\hat{\ }2-\tau(fst\ p+1)\hat{\ }2)$
**by** (*intro sum.reindex-cong*[**where** *l=prod.swap*] *arg-cong2*[**where** $f=(*)$]) *auto*
**also have** ...$=2*$
$(\sum (i,\ a)\in(SIGMA\ x{:}\{..<n\}.\ \{a \in arcs\ G.\ \varrho\ (head\ G\ a)\leq x \wedge x<\varrho(tail\ G\ a)\}).\ \tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2)$
**using** *less-trans*[*OF* - $\varrho$*-lt-n*] **by** (*intro sum.cong arg-cong2*[**where** $f=(*)$]) *auto*
**also have** ...$=2*(\sum i<n.\ (\sum a|a\in arcs\ G \wedge\varrho(head\ G\ a) \leq i\wedge i < \varrho(tail\ G\ a).\ \tau\ i\hat{\ }2 - \tau\ (i+1)\hat{\ }2))$
**by** (*subst sum.Sigma*) *auto*
**also have** ...$=2*(\sum i<n.\ card\ \{a\in arcs\ G.\ \varrho(head\ G\ a)\leq i\wedge i<\varrho(tail\ G\ a)\} * (\tau\ i\hat{\ }2 - \tau\ (i+1)\hat{\ }2))$
**by** *simp*
**also have** ...$=2*(\sum i<n.\ card\ \{a\in arcs\ G.\ \varrho(head\ G\ a)\leq i\wedge\neg(\varrho(tail\ G\ a)\leq i)\} * (\tau\ i\hat{\ }2 - \tau\ (i+1)\hat{\ }2))$
**by** (*intro-cong* [$\sigma_2$ $(*)$, $\sigma_1$ *card*, $\sigma_1$ *of-nat*] *more:sum.cong Collect-cong*) *auto*
**also have** ...$=2*(\sum i<n.\ card\ \{a\in arcs\ G.\ head\ G\ a\in\varphi`\{..i\}\wedge tail\ G\ a\notin\varphi`\{..i\}\} * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**using** *4*
**by** (*intro-cong* [$\sigma_2$ $(*)$, $\sigma_1$ *card*, $\sigma_1$ *of-nat*, $\sigma_2$ $(\wedge)$] *more:sum.cong restr-Collect-cong*) *auto*
**also have** ... $= 2 * (\sum i<n.\ real\ (card\ (edges\text{-}betw\ (-\varphi`\{..i\})\ (\varphi`\{..i\}))) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**unfolding** *edges-betw-def* **by** (*auto simp:conj.commute*)
**also have** ... $= 2 * (\sum i<n.\ real\ (card\ (edges\text{-}betw\ (\varphi`\{..i\})\ (-\varphi`\{..i\}))) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**using** *edges-betw-sym* **by** *simp*
**also have** ... $= 2 * (\sum i<m.\ real\ (card\ (edges\text{-}betw\ (\varphi`\{..i\})\ (-\varphi`\{..i\}))) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**using** $\tau$*-supp m-le-n* **by** (*intro sum.mono-neutral-right arg-cong2*[**where** $f=(*)$]) *auto*
**finally have** *Bf-eq*:
$B_f = 2 * (\sum i<m.\ real\ (card\ (edges\text{-}betw\ (\varphi`\{..i\})\ (-\varphi`\{..i\}))) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**by** *simp*

**have** *3*:*card* $(\varphi\ `\ \{..i\} \cap verts\ G) = i + 1$ **if** $i < m$ **for** $i$
**proof** −
**have** *card* $(\varphi\ `\ \{..i\} \cap verts\ G) = card\ (\varphi\ `\ \{..i\})$
**using** *m-le-n that* **by** (*intro arg-cong*[**where** *f=card*] *Int-absorb2*
*image-subsetI bij-betw-apply*[*OF* $\varphi$*-bij*]) *auto*
**also have** ... $= card\ \{..i\}$
**using** *m-le-n that* **by** (*intro card-image*
*inj-on-subset*[*OF bij-betw-imp-inj-on*[*OF* $\varphi$*-bij*]]) *auto*
**also have** ... $= i+1$ **by** *simp*
**finally show** *?thesis*
**by** *simp*
**qed**

**have** $2 * \Lambda_e * norm\ f\hat{\ }2 = 2 * \Lambda_e * (g\text{-}norm\ f'\hat{\ }2)$
**unfolding** *g-norm-conv f′-alt* **by** *simp*
**also have** ... $\leq 2 * \Lambda_e * (\sum v\in verts\ G.\ f'\ v\hat{\ }2)$
**unfolding** *g-norm-sq g-inner-def* **by** (*simp add:power2-eq-square*)
**also have** ... $= 2 * \Lambda_e * (\sum i<n.\ f'\ (\varphi\ i)\hat{\ }2)$
**by** (*intro arg-cong2*[**where** $f=(*)$] *refl sum.reindex-bij-betw*[*symmetric*] $\varphi$*-bij*)
**also have** ... $= 2 * \Lambda_e * (\sum i<n.\ \tau\ i\hat{\ }2)$
**unfolding** $\tau$*-def* **by** (*intro arg-cong2*[**where** $f=(*)$] *refl sum.cong*) *auto*
**also have** ... $= 2 * \Lambda_e * (\sum i<m.\ \tau\ i\hat{\ }2)$
**using** $\tau$*-supp m-le-n* **by** (*intro sum.mono-neutral-cong-right arg-cong2*[**where** $f=(*)$] *refl*) *auto*
**also have** ... $\leq 2 * \Lambda_e * ((\sum i<m.\ \tau\ i\hat{\ }2) + (real\ 0 * \tau\ 0\hat{\ }2 - m * \tau\ m\hat{\ }2))$

**using** $\tau$*-supp*[*of m*] **by** *simp*
**also have** ... $\leq 2 * \Lambda_e * ((\sum i<m.\ \tau\ i\hat{\ }2) + (\sum i<m.\ i{*}\tau\ i\hat{\ }2-(Suc\ i){*}\tau\ (Suc\ i)\hat{\ }2))$
**by** (*subst sum-lessThan-telescope*′[*symmetric*]) *simp*
**also have** ... $\leq 2 * (\sum i<m.\ (\Lambda_e * (i+1)) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**by** (*simp add:sum-distrib-left algebra-simps sum.distrib*[*symmetric*])
**also have** ... $\leq 2 * (\sum i<m.\ real\ (card\ (edges\text{-}betw\ (\varphi`\{..i\})\ (-\varphi`\{..i\}))) * (\tau\ i\hat{\ }2-\tau\ (i+1)\hat{\ }2))$
**using** $\tau$*-nonneg* $\tau$*-antimono power-mono 3 m2-le-n*
**by** (*intro mult-left-mono sum-mono mult-right-mono edge-expansionD2*) *auto*
**also have** ... $= B_f$
**unfolding** *Bf-eq* **by** *simp*
**finally have** *hoory-4-13*: $2 * \Lambda_e * norm\ f\hat{\ }2 \leq B_f$
**by** *simp*

Corresponds to Lemma 4.13 in Hoory et al.

**have** *f-nz*: $f \neq 0$
**proof** (*rule ccontr*)
**assume** *f-nz-assms*: $\neg (f \neq 0)$
**have** $g\ \$h\ i \leq 0$ **for** *i*
**proof** −
**have** $g\ \$h\ i \leq max\ (g\ \$h\ i)\ 0$
**by** *simp*
**also have** ... $= 0$
**using** *f-nz-assms* **unfolding** *f-def vec-eq-iff* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**
**moreover have** $(\sum i \in UNIV.\ 0-g\ \$h\ i) = 0$
**using** *g-orth* **unfolding** *sum-subtractf inner-vec-def* **by** *auto*
**ultimately have** $\forall x{\in}UNIV.\ -(g\ \$h\ x) = 0$
**by** (*intro iffD1*[*OF sum-nonneg-eq-0-iff*]) *auto*
**thus** *False*
**using** *g-nz* **unfolding** *vec-eq-iff* **by** *simp*
**qed**
**hence** *norm-f-gt-0*: $norm\ f > 0$
**by** *simp*

**have** $\Lambda_e * norm\ f * norm\ f \leq sqrt\ 2 * real\ d * norm\ f * sqrt\ (f \cdot (L *v\ f))$
**using** *order-trans*[*OF hoory-4-13 hoory-4-12*] **by** (*simp add:power2-eq-square*)
**hence** $\Lambda_e \leq real\ d * sqrt\ 2 * sqrt\ (f \cdot (L *v\ f))\ /\ norm\ f$
**using** *norm-f-gt-0* **by** (*simp add:ac-simps divide-simps*)
**also have** ... $\leq real\ d * sqrt\ 2 * sqrt\ ((1 - \Lambda_2) * (norm\ f)^2)\ /\ norm\ f$
**by** (*intro mult-left-mono divide-right-mono real-sqrt-le-mono h-part-i*) *auto*
**also have** ... $= real\ d * sqrt\ 2 * sqrt\ (1- \Lambda_2)$
**using** *f-nz* **by** (*simp add:real-sqrt-mult*)
**also have** ... $= d * sqrt\ (2 * (1-\Lambda_2))$
**by** (*simp add:real-sqrt-mult*[*symmetric*])
**finally show** *?thesis*
**by** *simp*
**qed**

**end**

**context** *regular-graph*
**begin**

**lemmas** (**in** *regular-graph*) *cheeger-aux-1* =
*regular-graph-tts.cheeger-aux-1*[*OF eg-tts-1*,
*internalize-sort* ′*n* :: *finite*, *OF - regular-graph-axioms*,
*unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**theorem** *cheeger-inequality*:
  **assumes** $n > 1$
  **shows** $\Lambda_e \in \{d * (1 - \Lambda_2) / 2.. d * sqrt (2 * (1 - \Lambda_2))\}$
  **using** *cheeger-aux-1 cheeger-aux-2 assms* **by** *auto*

**unbundle** *no intro-cong-syntax*

**end**

**end**

# 8 Margulis Gabber Galil Construction

This section formalizes the Margulis-Gabber-Galil expander graph, which is defined on the product space $\mathbb{Z}_n \times \mathbb{Z}_n$. The construction is an adaptation of graph introduced by Margulis [8], for which he gave a non-constructive proof of its spectral gap. Later Gabber and Galil [3] adapted the graph and derived an explicit spectral gap, i.e., that the second largest eigenvalue is bounded by $\frac{5}{8}\sqrt{2}$. The proof was later improved by Jimbo and Marouka [6] using Fourier Analysis. Hoory et al. [4, §8] present a slight simplification of that proof (due to Boppala) which this formalization is based on.

**theory** *Expander-Graphs-MGG*
  **imports**
    *HOL−Analysis.Complex-Transcendental*
    *HOL−Decision-Procs.Approximation*
    *Expander-Graphs-Definition*
**begin**

**datatype** $('a, 'b)$ *arc* = *Arc* (*arc-tail*: $'a$)  (*arc-head*: $'a$) (*arc-label*: $'b$)

**fun** *mgg-graph-step* :: $nat \Rightarrow (int \times int) \Rightarrow (nat \times int) \Rightarrow (int \times int)$
  **where** *mgg-graph-step* $n$ $(i,j)$ $(l,\sigma)$ =
  $[\ ((i+\sigma*(2*j+0))\ mod\ int\ n,\ j),\ (i,\ (j+\sigma*(2*i+0))\ mod\ int\ n)$
  $,\ ((i+\sigma*(2*j+1))\ mod\ int\ n,\ j),\ (i,\ (j+\sigma*(2*i+1))\ mod\ int\ n)\ ]\ !\ l$

**definition** *mgg-graph* :: $nat \Rightarrow (int \times int, (int \times int, nat \times int)\ arc)$ *pre-digraph* **where**
  *mgg-graph* $n$ =
    $(\!|$ *verts* = $\{0..<n\} \times \{0..<n\}$,
    *arcs* = $(\lambda(t,l).\ (Arc\ t\ (mgg\text{-}graph\text{-}step\ n\ t\ l)\ l))$ ` $((\{0..<int\ n\}\times\{0..<int\ n\})\times(\{..<4\}\times\{-1,1\}))$,
      *tail* = *arc-tail*,
      *head* = *arc-head* $|\!)$

**locale** *margulis-gaber-galil* =
  **fixes** $m$ :: *nat*
  **assumes** *m-gt-0*: $m > 0$
**begin**

**abbreviation** $G$ **where** $G \equiv$ *mgg-graph* $m$

**lemma** *wf-digraph*: *wf-digraph* (*mgg-graph* $m$)
**proof** −
  **have**
    *tail* (*mgg-graph* $m$) $e \in$ *verts* (*mgg-graph* $m$) (**is** *?A*)
    *head* (*mgg-graph* $m$) $e \in$ *verts* (*mgg-graph* $m$) (**is** *?B*)
    **if** $a$:$e \in$ *arcs* (*mgg-graph* $m$) **for** $e$
  **proof** −

```
    obtain t l σ where tl-def:
      t ∈ {0..<int m} × {0..<int m} l ∈ {..<4} σ ∈ {−1,1}
      e = Arc t (mgg-graph-step m t (l,σ)) (l,σ)
      using a mgg-graph-def by auto
    thus ?A
      unfolding mgg-graph-def by auto
    have mgg-graph-step m (fst t, snd t) (l,σ) ∈ {0..<int m} × {0..<int m}
      unfolding mgg-graph-step.simps using tl-def(1,2) m-gt-0
      by (intro set-mp[OF - nth-mem]) auto
    hence arc-head e ∈ {0..<int m} × {0..<int m}
      unfolding tl-def(4) by simp
    thus ?B
      unfolding mgg-graph-def by simp
  qed
  thus ?thesis
    by unfold-locales auto
qed

lemma mgg-finite: fin-digraph (mgg-graph m)
proof −
  have finite (verts (mgg-graph m)) finite (arcs (mgg-graph m))
    unfolding mgg-graph-def by auto
  thus ?thesis
    using wf-digraph
    unfolding fin-digraph-def fin-digraph-axioms-def by auto
qed

interpretation fin-digraph mgg-graph m
  using mgg-finite by simp

definition arcs-pos :: (int × int, nat × int) arc set
    where arcs-pos = (λ(t,l). (Arc t (mgg-graph-step m t (l,1)) (l, 1)))`(verts G×{..<4})
definition arcs-neg :: (int × int, nat × int) arc set
    where arcs-neg = (λ(h,l). (Arc (mgg-graph-step m h (l,1)) h (l,−1)))`(verts G×{..<4})

lemma arcs-sym:
  arcs G = arcs-pos ∪ arcs-neg
proof −
  have 0: x ∈ arcs G if x ∈ arcs-pos for x
    using that unfolding arcs-pos-def mgg-graph-def by auto
  have 1: a ∈ arcs G if t:a ∈ arcs-neg for a
  proof −
    obtain h l where hl-def: h ∈ verts G l ∈ {..<4} a = Arc (mgg-graph-step m h (l,1)) h (l,−1)
      using t unfolding arcs-neg-def by auto

    define t where t = mgg-graph-step m h (l,1)

    have h-ran: h ∈ {0..<int m} × {0..<int m}
      using hl-def(1) unfolding mgg-graph-def by simp
    have l-ran: l ∈ set [0,1,2,3]
      using hl-def(2) by auto

    have t ∈ {0..<int m} × {0..<int m}
      using h-ran l-ran
      unfolding t-def by (cases h, auto simp add:mod-simps)
    hence t-ran: t ∈ verts G
      unfolding mgg-graph-def by simp
```

```
    have h = mgg-graph-step m t (l,−1)
      using h-ran l-ran unfolding t-def by (cases h, auto simp add:mod-simps)
    hence a = Arc t (mgg-graph-step m t (l,−1)) (l,−1)
      unfolding t-def hl-def(3) by simp
    thus ?thesis
      using t-ran hl-def(2) mgg-graph-def by (simp add:image-iff)
  qed

  have card (arcs-pos ∪ arcs-neg) = card arcs-pos + card arcs-neg
    unfolding arcs-pos-def arcs-neg-def by (intro card-Un-disjoint finite-imageI) auto
  also have ... = card (verts G×{..<4::nat}) + card (verts G×{..<4::nat})
    unfolding arcs-pos-def arcs-neg-def
    by (intro arg-cong2[where f=(+)] card-image inj-onI) auto
  also have ... = card (verts G×{..<4::nat}×{−1,1::int})
    by simp
  also have ... = card ((λ(t, l). Arc t (mgg-graph-step m t l) l) ‘ (verts G ×{..<4}×{−1,1}))
    by (intro card-image[symmetric] inj-onI) auto
  also have ... = card (arcs G)
    unfolding mgg-graph-def by simp
  finally have card (arcs-pos ∪ arcs-neg) = card (arcs G)
    by simp
  hence arcs-pos ∪ arcs-neg = arcs G
    using 0 1 by (intro card-subset-eq, auto)
  thus ?thesis by simp
qed

lemma sym: symmetric-multi-graph (mgg-graph m)
proof −
  define f :: (int × int, nat × int) arc ⇒ (int × int, nat × int) arc
    where f a = Arc (arc-head a) (arc-tail a) (apsnd (λx. (−1) ∗ x) (arc-label a)) for a

  have a: bij-betw f arcs-pos arcs-neg
    by (intro bij-betwI[where g=f])
     (auto simp add:f-def image-iff arcs-pos-def arcs-neg-def)

  have b: bij-betw f arcs-neg arcs-pos
    by (intro bij-betwI[where g=f])
     (auto simp add:f-def image-iff  arcs-pos-def arcs-neg-def)

  have c:bij-betw f (arcs-pos ∪ arcs-neg) (arcs-neg ∪ arcs-pos)
    by (intro bij-betw-combine[OF a b])  (auto simp add:arcs-pos-def arcs-neg-def)

  hence c:bij-betw f (arcs G) (arcs G)
    unfolding arcs-sym by (subst (2) sup-commute, simp)
  show ?thesis
    by (intro symmetric-multi-graphI[where f=f] fin-digraph-axioms c)
     (simp add:f-def mgg-graph-def)
qed

lemma out-deg:
  assumes v ∈ verts G
  shows out-degree G v = 8
proof −
  have out-degree (mgg-graph m) v = card (out-arcs (mgg-graph m) v)
    unfolding out-degree-def by simp
  also have ... = card {e. (∃ w ∈ verts (mgg-graph m). ∃ l ∈ {..<4} × {−1,1}.
    e = Arc w (mgg-graph-step m w l) l ∧ arc-tail e = v)}
    unfolding mgg-graph-def out-arcs-def by (simp add:image-iff)
```

**also have** ... = *card* {*e*. ($\exists$ *l* $\in$ {..<*4*} $\times$ {$-$*1*,*1*}. *e* = *Arc v* (*mgg-graph-step m v l*) *l*)}
**using** *assms* **by** (*intro arg-cong*[**where** *f*=*card*] *iffD2*[*OF set-eq-iff*] *allI*) *auto*
**also have** ... = *card* (($\lambda$*l*. *Arc v* (*mgg-graph-step m v l*) *l*) ' ({..<*4*} $\times$ {$-$*1*,*1*}))
**by** (*intro arg-cong*[**where** *f*=*card*]) (*auto simp add*:*image-iff*)
**also have** ... = *card* ({..<*4*::*nat*} $\times$ {$-$*1*,*1*::*int*})
**by** (*intro card-image inj-onI*) *simp*
**also have** ... = *8* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *verts-ne*:
*verts G* $\neq$ {}
**using** *m-gt-0* **unfolding** *mgg-graph-def* **by** *simp*

**sublocale** *regular-graph mgg-graph m*
**using** *out-deg verts-ne*
**by** (*intro regular-graphI*[**where** *d*=*8*] *sym*) *auto*

**lemma** *d-eq-8*: *d* = *8*
**proof** $-$
**obtain** *v* **where** *v-def*: *v* $\in$ *verts G*
**using** *verts-ne* **by** *auto*
**hence** *0*:(*SOME v*. *v* $\in$ *verts G*) $\in$ *verts G*
**by** (*rule someI*[**where** *x*=*v*])
**show** *?thesis*
**using** *out-deg*[*OF 0*]
**unfolding** *d-def* **by** *simp*
**qed**

We start by introducing Fourier Analysis on the torus $\mathbb{Z}_n \times \mathbb{Z}_n$. The following is too specialized for a general AFP entry.

**lemma** *g-inner-sum-left*:
**assumes** *finite I*
**shows** *g-inner* ($\lambda$*x*. ($\sum$ *i* $\in$ *I*. *f i x*)) *g* = ($\sum$ *i*$\in$ *I*. *g-inner* (*f i*) *g*)
**using** *assms* **by** (*induction I rule*:*finite-induct*) (*auto simp add*:*g-inner-simps*)

**lemma** *g-inner-sum-right*:
**assumes** *finite I*
**shows** *g-inner f* ($\lambda$*x*. ($\sum$ *i* $\in$ *I*. *g i x*)) = ($\sum$ *i*$\in$ *I*. *g-inner f* (*g i*))
**using** *assms* **by** (*induction I rule*:*finite-induct*) (*auto simp add*:*g-inner-simps*)

**lemma** *g-inner-reindex*:
**assumes** *bij-betw h* (*verts G*) (*verts G*)
**shows** *g-inner f g* = *g-inner* ($\lambda$*x*. (*f* (*h x*))) ($\lambda$*x*. (*g* (*h x*)))
**unfolding** *g-inner-def*
**by** (*subst sum.reindex-bij-betw*[*OF assms*,*symmetric*]) *simp*

**definition** $\omega_F$ :: *real* $\Rightarrow$ *complex* **where** $\omega_F$ *x* = *cis* (*2*$*$*pi*$*$*x*/*m*)

**lemma** $\omega_F$*-simps*:
$\omega_F$ (*x* + *y*) = $\omega_F$ *x* $*$ $\omega_F$ *y*
$\omega_F$ (*x* $-$ *y*) = $\omega_F$ *x* $*$ $\omega_F$ ($-$*y*)
*cnj* ($\omega_F$ *x*) = $\omega_F$ ($-$*x*)
**unfolding** $\omega_F$*-def* **by** (*auto simp add*:*algebra-simps diff-divide-distrib*
*add-divide-distrib cis-mult cis-divide cis-cnj*)

**lemma** $\omega_F$*-cong*:
**fixes** *x y* :: *int*

**assumes** $x\ mod\ m = y\ mod\ m$
**shows** $\omega_F$ (*of-int* $x$) $= \omega_F$ (*of-int* $y$)
**proof** −
**obtain** $z$ :: *int* **where** $y = x + m{*}z$ **using** *mod-eqE*[*OF assms*] **by** *auto*
**hence** $\omega_F$ (*of-int* $y$) $= \omega_F$ (*of-int* $x$ + *of-int* ($m{*}z$))
**by** *simp*
**also have** ... $= \omega_F$ (*of-int* $x$) $*$ $\omega_F$ (*of-int* ($m{*}z$))
**by** (*simp add*:$\omega_F$-*simps*)
**also have** ... $= \omega_F$ (*of-int* $x$) $*$ *cis* ($2 * pi *$ *of-int* ($z$))
**unfolding** $\omega_F$-*def* **using** *m-gt-0*
**by** (*intro arg-cong2*[**where** $f$=($*$)] *arg-cong*[**where** $f$=*cis*]) *auto*
**also have** ... $= \omega_F$ (*of-int* $x$) $*$ $1$
**by** (*intro arg-cong2*[**where** $f$=($*$)] *cis-multiple-2pi*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *cis-eq-1-imp*:
**assumes** *cis* ($2 * pi * x$) $= 1$
**shows** $x \in \mathbb{Z}$
**proof** −
**have** *cos* ($2 * pi * x$) $=$ *Re* (*cis* ($2{*}pi{*}x$))
**using** *cis.simps* **by** *simp*
**also have** ... $= 1$
**unfolding** *assms* **by** *simp*
**finally have** *cos* ($2 * pi * x$) $= 1$ **by** *simp*
**then obtain** $y$ **where** $2 * pi * x =$ *of-int* $y * 2 * pi$
**using** *cos-one-2pi-int* **by** *auto*
**hence** $y = x$ **by** *simp*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** $\omega_F$-*eq-1-iff*:
**fixes** $x$ :: *int*
**shows** $\omega_F\ x = 1 \longleftrightarrow x\ mod\ m = 0$
**proof**
**assume** $\omega_F$ (*real-of-int* $x$) $= 1$
**hence** *cis* ($2 * pi *$ *real-of-int* $x$ / *real* $m$) $= 1$
**unfolding** $\omega_F$-*def* **by** *simp*
**hence** *real-of-int* $x$ / *real* $m \in \mathbb{Z}$
**using** *cis-eq-1-imp* **by** *simp*
**then obtain** $z$ :: *int* **where** *of-int* $x$ / *real* $m = z$
**using** *Ints-cases* **by** *auto*
**hence** $x = z *$ *real* $m$
**using** *m-gt-0* **by** (*simp add*: *nonzero-divide-eq-eq*)
**hence** $x = z * m$ **using** *of-int-eq-iff* **by** *fastforce*
**thus** $x\ mod\ m = 0$ **by** *simp*
**next**
**assume** $x\ mod\ m = 0$
**hence** $\omega_F\ x = \omega_F$ (*of-int* $0$)
**by** (*intro* $\omega_F$-*cong*) *auto*
**also have** ... $= 1$ **unfolding** $\omega_F$-*def* **by** *simp*
**finally show** $\omega_F\ x{=}\ 1$ **by** *simp*
**qed**

**definition** *FT* :: (*int* $\times$ *int* $\Rightarrow$ *complex*) $\Rightarrow$ (*int* $\times$ *int* $\Rightarrow$ *complex*)
**where** *FT f v* $=$ *g-inner f* ($\lambda x.\ \omega_F$ (*fst* $x *$ *fst* $v$ + *snd* $x *$ *snd* $v$))

**lemma** *FT-altdef*: *FT f* ($u$,$v$) $=$ *g-inner f* ($\lambda x.\ \omega_F$ (*fst* $x * u$ + *snd* $x * v$))

**unfolding** *FT-def* **by** (*simp add:case-prod-beta*)

**lemma** *FT-add*: *FT* ($\lambda x.$ *f x* + *g x*) *v* = *FT f v* + *FT g v*
**unfolding** *FT-def* **by** (*simp add:g-inner-simps algebra-simps*)

**lemma** *FT-zero*: *FT* ($\lambda x.$ *0*) *v* = *0*
**unfolding** *FT-def g-inner-def* **by** *simp*

**lemma** *FT-sum*:
**assumes** *finite I*
**shows** *FT* ($\lambda x.$ ($\sum i \in I.$ *f i x*)) *v* = ($\sum i \in I.$ *FT* (*f i*) *v*)
**using** *assms* **by** (*induction rule*: *finite-induct*, *auto simp add:FT-zero FT-add*)

**lemma** *FT-scale*: *FT* ($\lambda x.$ *c* $*$ *f x*) *v* = *c* $*$ *FT f v*
**unfolding** *FT-def* **by** (*simp add*: *g-inner-simps*)

**lemma** *FT-cong*:
**assumes** $\bigwedge x.$ *x* $\in$ *verts G* $\Longrightarrow$ *f x* = *g x*
**shows** *FT f* = *FT g*
**unfolding** *FT-def* **by** (*intro ext g-inner-cong assms refl*)

**lemma** *parseval*:
*g-inner f g* = *g-inner* (*FT f*) (*FT g*)/*m*^*2* (**is** *?L* = *?R*)
**proof** −
**define** $\delta$ :: (*int* $\times$ *int*) $\Rightarrow$ (*int* $\times$ *int*) $\Rightarrow$ *complex* **where** $\delta$ *x y* = *of-bool* (*x* = *y*) **for** *x y*

**have** *FT-*$\delta$: *FT* ($\delta$ *v*) *x* = $\omega_F$ (−(*fst v* $*$*fst x* +*snd v* $*$ *snd x*)) **if** *v* $\in$ *verts G* **for** *v x*
**using** *that* **by** (*simp add:FT-def g-inner-def* $\delta$*-def* $\omega_F$*-simps*)

**have** *1*: ($\sum$*x=0..<int m.* $\omega_F$ (*z*$*$*x*)) = *m* $*$ *of-bool*(*z mod m* = *0*) (**is** *?L1* = *?R1*) **for** *z* :: *int*
**proof** (*cases z mod m* = *0*)
**case** *True*
**have** ($\sum$*x=0..<int m.* $\omega_F$ (*z*$*$*x*)) = ($\sum$*x=0..<int m.* $\omega_F$ (*of-int 0*))
**using** *True* **by** (*intro sum.cong* $\omega_F$*-cong refl*) *auto*
**also have** ... = *m* $*$ *of-bool*(*z mod m* = *0*)
**unfolding** $\omega_F$*-def True* **by** *simp*
**finally show** *?thesis* **by** *simp*
**next**
**case** *False*
**have** (*1*−$\omega_F$ *z*) $*$ *?L1* = (*1*−$\omega_F$ *z*) $*$ ($\sum$*x* $\in$ *int* ‘ {..<*m*}. $\omega_F$(*z*$*$*x*))
**by** (*intro arg-cong2*[**where** *f*=($*$)] *sum.cong refl*)
(*simp add*: *image-atLeastZeroLessThan-int*)
**also have** ... = ($\sum$*x*<*m.* $\omega_F$(*z*$*$*real x*) − $\omega_F$(*z*$*$(*real* (*Suc x*))))
**by** (*subst sum.reindex*, *auto simp add:algebra-simps sum-distrib-left* $\omega_F$*-simps*)
**also have** ... = $\omega_F$ (*z* $*$ *0*) − $\omega_F$ (*z* $*$ *m*)
**by** (*subst sum-lessThan-telescope'*) *simp*
**also have** ... = $\omega_F$ (*of-int 0*) − $\omega_F$ (*of-int 0*)
**by** (*intro arg-cong2*[**where** *f*=(−)] $\omega_F$*-cong*) *auto*
**also have** ... = *0*
**by** *simp*
**finally have** (*1*− $\omega_F$ *z*) $*$ *?L1* = *0* **by** *simp*
**moreover have** $\omega_F$ *z* $\neq$ *1* **using** $\omega_F$*-eq-1-iff False* **by** *simp*
**hence** (*1*− $\omega_F$ *z*) $\neq$ *0* **by** *simp*
**ultimately have** *?L1* = *0* **by** *simp*
**then show** *?thesis* **using** *False* **by** *simp*
**qed**

**have** *0*:*g-inner* ($\delta$ *v*) ($\delta$ *w*) = *g-inner* (*FT* ($\delta$ *v*)) (*FT* ($\delta$ *w*))/*m*^*2* (**is** *?L1* = *?R1*/-)

**if** $v \in$ *verts G* $w \in$ *verts G* **for** *v w*
**proof** −
**have** *?R1=g-inner(λx. $\omega_F$(−(fst v ∗fst x +snd v ∗ snd x)))(λx. $\omega_F$(−(fst w ∗fst x +snd w ∗ snd x)))*
**using** *that* **by** (*intro g-inner-cong, auto simp add:FT-δ*)
**also have** *...=(∑ (x,y)∈{0..<int m}×{0..<int m}. $\omega_F$((fst w−fst v)∗x)∗$\omega_F$((snd w − snd v)∗ y))*
**unfolding** *g-inner-def* **by** (*simp add:$\omega_F$-simps algebra-simps case-prod-beta mgg-graph-def*)
**also have** *...=(∑ x=0..<int m. ∑ y = 0..<int m. $\omega_F$((fst w − fst v)∗x)∗$\omega_F$((snd w − snd v) ∗ y))*
**by** (*subst sum.cartesian-product[symmetric]*) *simp*
**also have** *...=(∑ x=0..<int m. $\omega_F$((fst w − fst v)∗x))∗(∑ y = 0..<int m. $\omega_F$((snd w − snd v) ∗ y))*
**by** (*subst sum.swap*) (*simp add:sum-distrib-left sum-distrib-right*)
**also have** *... = of-nat (m ∗ of-bool(fst v mod m = fst w mod m)) ∗*
*of-nat (m ∗ of-bool(snd v mod m = snd w mod m))*
**using** *m-gt-0* **unfolding** *1*
**by** (*intro arg-cong2*[**where** *f=(∗)*] *arg-cong*[**where** *f=of-bool*]
*arg-cong*[**where** *f=of-nat*] *refl*) (*auto simp add:algebra-simps cong:mod-diff-cong*)
**also have** *... = m^2 ∗ of-bool(v = w)*
**using** *that* **by** (*auto simp add:prod-eq-iff mgg-graph-def power2-eq-square*)
**also have** *... = m^2 ∗ ?L1*
**using** *that* **unfolding** *g-inner-def δ-def* **by** *simp*
**finally have** *?R1 = m^2 ∗ ?L1* **by** *simp*
**thus** *?thesis* **using** *m-gt-0* **by** *simp*
**qed**

**have** *?L = g-inner (λx. (∑ v ∈ verts G. (f v) ∗ δ v x)) (λx. (∑ v ∈ verts G. (g v) ∗ δ v x))*
**unfolding** *δ-def* **by** (*intro g-inner-cong*) *auto*
**also have** *... = (∑ v∈verts G. (f v) ∗ (∑ w∈verts G. cnj (g w) ∗ g-inner (δ v) (δ w)))*
**by** (*simp add:g-inner-simps g-inner-sum-left g-inner-sum-right*)
**also have** *... = (∑ v∈verts G. (f v) ∗ (∑ w∈verts G. cnj (g w) ∗ g-inner(FT (δ v)) (FT (δ w))))/m^2*
**by** (*simp add:0 sum-divide-distrib sum-distrib-left algebra-simps*)
**also have** *...=g-inner(λx.(∑ v∈verts G. (f v)∗FT (δ v) x))(λx.(∑ v∈verts G. (g v)∗FT (δ v) x))/$m^2$*
**by** (*simp add:g-inner-simps g-inner-sum-left g-inner-sum-right*)
**also have** *...=g-inner(FT(λx.(∑ v∈verts G.(f v)∗δ v x)))(FT(λx.(∑ v∈verts G.(g v)∗δ v x)))/$m^2$*
**by** (*intro g-inner-cong arg-cong2*[**where** *f=(/)*]) (*simp-all add: FT-sum FT-scale*)
**also have** *... = g-inner (FT f) (FT g)/m^2*
**unfolding** *δ-def comp-def*
**by** (*intro g-inner-cong arg-cong2*[**where** *f=(/)*] *fun-cong[OF FT-cong]*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *plancharel*:
(*∑ v ∈ verts G. norm (f v)^2) = (∑ v ∈ verts G. norm (FT f v)^2)/m^2* (**is** *?L = ?R*)
**proof** −
**have** *complex-of-real ?L = g-inner f f*
**by** (*simp flip:of-real-power add:complex-norm-square g-inner-def algebra-simps*)
**also have** *... = g-inner (FT f) (FT f) / m^2*
**by** (*subst parseval*) *simp*
**also have** *... = complex-of-real ?R*
**by** (*simp flip:of-real-power add:complex-norm-square g-inner-def algebra-simps*) *simp*
**finally have** *complex-of-real ?L = complex-of-real ?R* **by** *simp*
**thus** *?thesis*
**using** *of-real-eq-iff* **by** *blast*
**qed**

```
lemma FT-swap:
  FT (λx. f (snd x, fst x)) (u,v) = FT f (v,u)
proof −
  have 0:bij-betw (λ(x::int × int). (snd x, fst x)) (verts G) (verts G)
    by (intro bij-betwI[where g=(λ(x::int × int). (snd x, fst x))])
     (auto simp add:mgg-graph-def)
  show ?thesis
    unfolding FT-def
    by (subst g-inner-reindex[OF 0]) (simp add:algebra-simps)
qed

lemma mod-add-mult-eq:
  fixes a x y :: int
  shows (a + x ∗ (y mod m)) mod m = (a+x∗y) mod m
  using mod-add-cong mod-mult-right-eq by blast

definition periodic where periodic f = (∀ x y. f (x,y) = f (x mod int m, y mod int m))

lemma periodicD:
  assumes periodic f
  shows f (x,y) = f (x mod m, y mod m)
  using assms unfolding periodic-def by simp

lemma periodic-comp:
  assumes periodic f
  shows periodic (λx. g (f x))
  using assms unfolding periodic-def by simp

lemma periodic-cong:
  fixes x y u v :: int
  assumes periodic f
  assumes x mod m = u mod m y mod m = v mod m
  shows f (x,y) = f (u, v)
  using periodicD[OF assms(1)] assms(2,3) by metis

lemma periodic-FT: periodic (FT f)
proof −
  have FT f (x,y) = FT f (x mod m,y mod m) for x y
    unfolding FT-altdef by (intro g-inner-cong ω_F-cong ext)
     (auto simp add:mod-simps cong:mod-add-cong)
  thus ?thesis
    unfolding periodic-def by simp
qed

lemma FT-sheer-aux:
  fixes u v c d :: int
  assumes periodic f
  shows  FT (λx. f (fst x,snd x+c∗fst x+d)) (u,v) = ω_F (d∗ v) ∗ FT f (u−c∗ v,v)
    (is ?L = ?R)
proof −
  define s where s = (λ(x,y). (x, (y − c ∗ x−d) mod m))
  define s0 where s0 = (λ(x,y). (x, (y−c∗x) mod m))
  define s1 where s1 = (λ(x::int,y). (x, (y−d) mod m))

  have 0:bij-betw s0 (verts G) (verts G)
    by (intro bij-betwI[where g=λ(x,y). (x,(y+c∗x) mod m)])
     (auto simp add:mgg-graph-def s0-def Pi-def mod-simps)
```

**have** *1*:*bij-betw s1* (*verts G*) (*verts G*)
  **by** (*intro bij-betwI*[**where** *g*=λ(*x*,*y*). (*x*,(*y*+*d*) *mod m*)])
   (*auto simp add*:*mgg-graph-def s1-def Pi-def mod-simps*)
**have** *2*: *s* = (*s1* ∘ *s0*)
  **by** (*simp add*:*s1-def s0-def s-def comp-def mod-simps case-prod-beta ext*)
**have** *3*:*bij-betw s* (*verts G*) (*verts G*)
  **unfolding** *2* **using** *bij-betw-trans*[*OF 0 1*] **by** *simp*

**have** *4*:(*snd* (*s x*) + *c* ∗ *fst x* + *d*) *mod int m* = *snd x mod m* **for** *x*
  **unfolding** *s-def* **by** (*simp add*:*case-prod-beta cong*:*mod-add-cong*) (*simp add*:*algebra-simps*)
**have** *5*: *fst* (*s x*) = *fst x* **for** *x*
  **unfolding** *s-def* **by** (*cases x*, *simp*)

**have** *?L* = *g-inner* (λ*x*. *f* (*fst x*, *snd x* + *c*∗*fst x*+*d*)) (λ*x*. $\omega_F$ (*fst x*∗*u* + *snd x*∗ *v*))
  **unfolding** *FT-altdef* **by** *simp*
**also have** ... = *g-inner* (λ*x*. *f* (*fst x*, (*snd x* + *c*∗*fst x*+*d*) *mod m*)) (λ*x*. $\omega_F$ (*fst x*∗*u* + *snd x*∗ *v*))
  **by** (*intro g-inner-cong periodic-cong*[*OF assms*]) (*auto simp add*:*algebra-simps*)
**also have** ... = *g-inner* (λ*x*. *f* (*fst x*, *snd x mod m*)) (λ*x*. $\omega_F$ (*fst x*∗*u*+ *snd* (*s x*)∗ *v*))
  **by** (*subst g-inner-reindex*[*OF 3*]) (*simp add*:*4 5*)
**also have** ... =
  *g-inner* (λ*x*. *f* (*fst x*, *snd x mod m*)) (λ*x*. $\omega_F$ (*fst x*∗*u*+ ((*snd x*−*c*∗*fst x*−*d*) *mod m*)∗ *v*))
  **by** (*simp add*:*s-def case-prod-beta*)
**also have** ... = *g-inner f* (λ*x*. $\omega_F$ (*fst x*∗ (*u*−*c* ∗ *v*) + *snd x* ∗ *v*−*d* ∗ *v*))
  **by** (*intro g-inner-cong* $\omega_F$*-cong*) (*auto simp add*:*mgg-graph-def algebra-simps mod-add-mult-eq*)
**also have** ... = *g-inner f* (λ*x*. $\omega_F$ (−*d*∗ *v*)∗$\omega_F$ (*fst x*∗(*u*−*c*∗ *v*) + *snd x* ∗ *v*))
  **by** (*simp add*: $\omega_F$*-simps algebra-simps*)
**also have** ... = $\omega_F$ (*d*∗ *v*)∗*g-inner f* (λ*x*. $\omega_F$ (*fst x*∗(*u*−*c*∗ *v*) + *snd x* ∗ *v*))
  **by** (*simp add*:*g-inner-simps* $\omega_F$*-simps*)
**also have** ... = *?R*
  **unfolding** *FT-altdef* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *FT-sheer*:
  **fixes** *u v c d* :: *int*
  **assumes** *periodic f*
  **shows**
    *FT* (λ*x*. *f* (*fst x*,*snd x*+*c*∗*fst x*+*d*)) (*u*,*v*) = $\omega_F$ (*d*∗ *v*) ∗ *FT f* (*u*−*c*∗ *v*,*v*) (**is** *?A*)
    *FT* (λ*x*. *f* (*fst x*,*snd x*+*c*∗*fst x*)) (*u*,*v*) = *FT f* (*u*−*c*∗ *v*,*v*) (**is** *?B*)
    *FT* (λ*x*. *f* (*fst x*+*c*∗ *snd x*+*d*,*snd x*)) (*u*,*v*) = $\omega_F$ (*d*∗ *u*) ∗ *FT f* (*u*,*v*−*c*∗*u*) (**is** *?C*)
    *FT* (λ*x*. *f* (*fst x*+*c*∗ *snd x*,*snd x*)) (*u*,*v*) = *FT f* (*u*,*v*−*c*∗*u*) (**is** *?D*)
**proof** −
  **have** *1*: *periodic* (λ*x*. *f* (*snd x*, *fst x*))
    **using** *assms* **unfolding** *periodic-def* **by** *simp*

  **have** *0*:$\omega_F$ *0* = *1*
    **unfolding** $\omega_F$*-def* **by** *simp*
  **show** *?A*
    **using** *FT-sheer-aux*[*OF assms*] **by** *simp*
  **show** *?B*
    **using** *0 FT-sheer-aux*[*OF assms*, **where** *d*=*0*] **by** *simp*
  **show** *?C*
    **using** *FT-sheer-aux*[*OF 1*] **by** (*subst* (*1 2*) *FT-swap*[*symmetric*], *simp*)
  **show** *?D*
    **using** *0 FT-sheer-aux*[*OF 1*, **where** *d*=*0*] **by** (*subst* (*1 2*) *FT-swap*[*symmetric*], *simp*)
**qed**

**definition** $T_1$ :: $int \times int \Rightarrow int \times int$ **where** $T_1\ x = ((fst\ x + 2 * snd\ x)\ mod\ m,\ snd\ x)$
**definition** $S_1$ :: $int \times int \Rightarrow int \times int$ **where** $S_1\ x = ((fst\ x - 2 * snd\ x)\ mod\ m,\ snd\ x)$
**definition** $T_2$ :: $int \times int \Rightarrow int \times int$ **where** $T_2\ x = (fst\ x,\ (snd\ x + 2 * fst\ x)\ mod\ m)$
**definition** $S_2$ :: $int \times int \Rightarrow int \times int$ **where** $S_2\ x = (fst\ x,\ (snd\ x - 2 * fst\ x)\ mod\ m)$

**definition** $\gamma$-*aux* :: $int \times int \Rightarrow real \times real$
  **where** $\gamma$-*aux* $x = (|fst\ x/m - 1/2|, |snd\ x/m - 1/2|)$

**definition** *compare* :: $real \times real \Rightarrow real \times real \Rightarrow bool$
 **where** *compare* $x\ y = (fst\ x \le fst\ y \wedge snd\ x \le snd\ y \wedge x \ne y)$

The value here is different from the value in the source material. This is because the proof in Hoory [4, §8] only establishes the bound $\frac{73}{80}$ while this formalization establishes the improved bound of $\frac{5}{8}\sqrt{2}$.

**definition** $\alpha$ :: *real* **where** $\alpha = sqrt\ 2$

**lemma** $\alpha$-*inv*: $1/\alpha = \alpha/2$
 **unfolding** $\alpha$-*def* **by** (*simp add*: *real-div-sqrt*)

**definition** $\gamma$ :: $int \times int \Rightarrow int \times int \Rightarrow real$
 **where** $\gamma\ x\ y$ = (*if compare* ($\gamma$-*aux* $x$) ($\gamma$-*aux* $y$) *then* $\alpha$ *else* (*if compare* ($\gamma$-*aux* $y$) ($\gamma$-*aux* $x$) *then* $(1\ /\ \alpha)$ *else* $1$))

**lemma** $\gamma$-*sym*: $\gamma\ x\ y * \gamma\ y\ x = 1$
 **unfolding** $\gamma$-*def* $\alpha$-*def* *compare-def* **by** (*auto simp add*:*prod-eq-iff*)

**lemma** $\gamma$-*nonneg*: $\gamma\ x\ y \ge 0$
 **unfolding** $\gamma$-*def* $\alpha$-*def* **by** *auto*

**definition** $\tau$ :: $int \Rightarrow real$ **where** $\tau\ x = |cos(pi*x/m)|$

**definition** $\gamma'$ :: $real \Rightarrow real \Rightarrow real$
 **where** $\gamma'\ x\ y$ = (*if* $abs\ (x - 1/2) < abs\ (y - 1/2)$ *then* $\alpha$ *else* (*if* $abs\ (x-1/2) > abs\ (y-1/2)$ *then* $(1\ /\ \alpha)$ *else* $1$))

**definition** $\varphi$ :: $real \Rightarrow real \Rightarrow real$
 **where** $\varphi\ x\ y = \gamma'\ y\ (frac(y-2*x)) + \gamma'\ y\ (frac\ (y+2*x))$

**lemma** $\gamma'$-*cases*:
 $abs\ (x-1/2) = abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = 1$
 $abs\ (x-1/2) > abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = 1/\alpha$
 $abs\ (x-1/2) < abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = \alpha$
 **unfolding** $\gamma'$-*def* **by** *auto*

**lemma** *if-cong-direct*:
 **assumes** $a = b$
 **assumes** $c = d'$
 **assumes** $e = f$
 **shows** (*if* $a$ *then* $c$ *else* $e$) = (*if* $b$ *then* $d'$ *else* $f$)
 **using** *assms* **by** (*intro if-cong*) *auto*

**lemma** $\gamma'$-*cong*:
 **assumes** $abs\ (x-1/2) = abs\ (u-1/2)$
 **assumes** $abs\ (y-1/2) = abs\ (v-1/2)$
 **shows** $\gamma'\ x\ y = \gamma'\ u\ v$
 **unfolding** $\gamma'$-*def*
 **using** *assms* **by** (*intro if-cong-direct refl*) *auto*

**lemma** *add-swap-cong*:
  **fixes** $x\ y\ u\ v :: 'a :: ab\text{-}semigroup\text{-}add$
  **assumes** $x = y\ u = v$
  **shows** $x + u = v + y$
  **using** *assms* **by** (*simp add*:*algebra-simps*)

**lemma** *frac-cong*:
  **fixes** $x\ y :: real$
  **assumes** $x - y \in \mathbb{Z}$
  **shows** *frac* $x =$ *frac* $y$
**proof** –
  **obtain** $k$ **where** *x-eq*: $x = y + of\text{-}int\ k$
    **using** *Ints-cases*[*OF assms*] **by** (*metis add-minus-cancel uminus-add-conv-diff*)
  **thus** *?thesis*
    **unfolding** *x-eq* **unfolding** *frac-def* **by** *simp*
**qed**

**lemma** *frac-expand*:
  **fixes** $x :: real$
  **shows** *frac* $x = (if\ x < (-1)\ then\ (x-\lfloor x\rfloor)\ else\ (if\ x < 0\ then\ (x+1)\ else\ (if\ x < 1\ then\ x\ else$ $(if\ x < 2\ then\ (x-1)\ else\ (x-\lfloor x\rfloor)))))$
**proof** –
  **have** $real\text{-}of\text{-}int\ y = -1 \longleftrightarrow y= -1$ **for** $y$
    **by** *auto*
  **thus** *?thesis*
    **unfolding** *frac-def* **by** (*auto simp add*:*not-less floor-eq-iff*)
**qed**

**lemma** *one-minus-frac*:
  **fixes** $x :: real$
  **shows** $1 - frac\ x = (if\ x \in \mathbb{Z}\ then\ 1\ else\ frac\ (-x))$
  **unfolding** *frac-neg* **by** *simp*

**lemma** *abs-rev-cong*:
  **fixes** $x\ y :: real$
  **assumes** $x = -\ y$
  **shows** $abs\ x = abs\ y$
  **using** *assms* **by** *simp*

**lemma** *cos-pi-ge-0*:
  **assumes** $x \in\{-1/2..\ 1/2\}$
  **shows** $cos\ (pi * x) \geq 0$
**proof** –
  **have** $pi * x \in ((*)\ pi\ `\ \{-1/2..1/2\})$
    **by** (*intro imageI assms*)
  **also have** ... $=\ \{-pi/2..pi/2\}$
    **by** (*subst image-mult-atLeastAtMost*[*OF pi-gt-zero*]) *simp*
  **finally have** $pi * x \in \{-pi/2..pi/2\}$ **by** *simp*
  **thus** *?thesis*
    **by** (*intro cos-ge-zero*) *auto*
**qed**

The following is the first step in establishing Eq. 15 in Hoory et al. [4, §8]. Afterwards using various symmetries (diagonal, x-axis, y-axis) the result will follow for the entire square $[0, 1] \times [0, 1]$.

**lemma** *fun-bound-real-3*:
  **assumes** $0 \leq x\ \ x \leq y\ y \leq 1/2\ (x,y) \neq (0,0)$
  **shows** $|cos(pi*x)|*\varphi\ x\ y + |cos(pi*y)|*\varphi\ y\ x \leq 2.5 * sqrt\ 2$ (**is** $?L \leq ?R$)

```
proof −
  have apx:4 ≤ 5 ∗ sqrt (2::real) 8 ∗ cos (pi / 4) ≤ 5 ∗ sqrt (2::real)
    by (approximation 5)+

  have cos (pi ∗ x) ≥ 0
    using assms(1,2,3)  by (intro cos-pi-ge-0) simp
  moreover have cos (pi ∗ y) ≥ 0
    using assms(1,2,3)  by (intro cos-pi-ge-0) simp
  ultimately have 0:?L = cos(pi∗x)∗φ x y + cos(pi∗y)∗φ y x  (is - = ?T)
    by simp

  consider (a) x+y < 1/2 | (b) y = 1/2− x | (c) x+y > 1/2 by argo
  hence ?T ≤ 2.5 ∗ sqrt 2 (is ?T ≤ ?R)
  proof (cases)
    case a
    consider
      (1) x < y x > 0 |
      (2) x=0 y < 1/2 |
      (3) y=x x > 0
      using assms(1,2,3,4) a by fastforce
    thus ?thesis
    proof (cases)
      case 1
      have φ x y = α + 1/α
        unfolding φ-def using 1 a
        by (intro arg-cong2[where f=(+)] γ′-cases) (auto simp add:frac-expand)
      moreover have φ y x = 1/α + 1/α
        unfolding φ-def using 1 a
        by (intro arg-cong2[where f=(+)] γ′-cases) (auto simp add:frac-expand)
      ultimately have ?T = cos (pi ∗ x) ∗ (α + 1/α) + cos (pi ∗ y) ∗ (1/α + 1/α)
        by simp
      also have ... ≤ 1 ∗ (α + 1/α) + 1 ∗ (1/α + 1/α)
        unfolding α-def by (intro add-mono mult-right-mono) auto
      also have ... = ?R
        unfolding α-def by (simp add:divide-simps)
      finally show ?thesis by simp
    next
      case 2
      have y-range: y ∈ {0<..<1/2}
        using assms 2 by simp
      have φ 0 y = 1 + 1
        unfolding φ-def using y-range
        by (intro arg-cong2[where f=(+)] γ′-cases) (auto simp add:frac-expand)
      moreover
      have |x| ∗ 2 < 1 ←→ x < 1/2 ∧ −x < 1/2 for x :: real by auto
      hence φ y 0 = 1 / α + 1/ α
        unfolding φ-def using y-range
        by (intro arg-cong2[where f=(+)] γ′-cases) (simp-all add:frac-expand)
      ultimately have ?T = 2 + cos (pi ∗ y) ∗ (2 / α)
        unfolding 2 by simp
      also have ... ≤ 2 + 1 ∗ (2 / α)
        unfolding α-def by (intro add-mono mult-right-mono) auto
      also have ... ≤ ?R
        unfolding α-def by (approximation 10)
      finally show ?thesis by simp
    next
      case 3
      have φ x y = 1 + 1/α
```

```
      unfolding φ-def using 3 a
      by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
    moreover have φ y x = 1 + 1/α
      unfolding φ-def using 3 a
      by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
    ultimately have ?T = cos (pi * x) * (2*(1+1/ α))
      unfolding 3 by simp
    also have ... ≤ 1 * (2*(1+1/ α))
      unfolding α-def by (intro mult-right-mono) auto
    also have ... ≤ ?R
      unfolding α-def by (approximation 10)
    finally show ?thesis by simp
  qed
next
  case b
  have x-range: x ∈ {0..1/4}
    using assms b by simp
  then consider (1) x = 0 | (2) x = 1/4 | (3) x ∈ {0<..<1/4} by fastforce
  thus ?thesis
  proof (cases)
    case 1
    hence y-eq: y = 1/2 using b by simp
    show ?thesis using apx unfolding 1 y-eq φ-def by (simp add:γ'-def α-def frac-def)
  next
    case 2
    hence y-eq: y = 1/4 using b by simp
    show ?thesis using apx unfolding y-eq 2 φ-def by (simp add:γ'-def frac-def)
  next
    case 3
    have φ x y = α + 1
      unfolding φ-def b using 3
      by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
    moreover have φ y x = 1/α + 1
      unfolding φ-def b using 3
      by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
    ultimately have ?T = cos (pi * x) * (α + 1) + cos (pi * (1 / 2 − x)) * (1/α + 1)
      unfolding b by simp
    also have ... ≤ ?R
      unfolding α-def using x-range
      by (approximation 10 splitting: x=10)
    finally show ?thesis by simp
  qed
next
  case c
  consider
    (1) x < y y < 1/2 |
    (2) y=1/2 x < 1/2 |
    (3) y=x x < 1/2 |
    (4) x=1/2 y =1/2
    using assms(2,3) c  by fastforce
  thus ?thesis
  proof (cases)
    case 1
    define ϑ :: real where ϑ = arcsin (6 / 10)
    have cos ϑ = sqrt (1−0.6^2)
      unfolding ϑ-def by (intro cos-arcsin) auto
    also have ... = sqrt ( 0.8^2)
      by (intro arg-cong[where f=sqrt]) (simp add:power2-eq-square)
```

```
  also have ... = 0.8 by simp
  finally have cos-ϑ: cos ϑ = 0.8 by simp
  have sin-ϑ: sin ϑ = 0.6
    unfolding ϑ-def by simp

  have φ x y = α + α
    unfolding φ-def using c 1
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  moreover have φ y x = 1/α + α
    unfolding φ-def using c 1
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  ultimately have ?T = cos (pi ∗ x) ∗ (2 ∗ α) + cos (pi ∗ y) ∗ (α + 1 / α)
    by simp
  also have ... ≤ cos (pi ∗ (1/2−y)) ∗ (2∗α) + cos (pi ∗ y) ∗ (α+1 / α)
    unfolding α-def using assms(1,2,3) c
    by (intro add-mono mult-right-mono order.refl iffD2[OF cos-mono-le-eq]) auto
  also have ... = (2.5∗α)∗ (sin (pi ∗ y) ∗ 0.8 + cos (pi ∗ y) ∗ 0.6)
    unfolding sin-cos-eq α-inv by (simp add:algebra-simps)
  also have ... = (2.5∗α)∗ sin(pi∗y + ϑ)
    unfolding sin-add cos-ϑ sin-ϑ
    by (intro arg-cong2[where f=(∗)] arg-cong2[where f=(+)] refl)
  also have ... ≤ (?R) ∗ 1
    unfolding α-def by (intro mult-left-mono) auto
  finally show ?thesis by simp
next
  case 2
  have x-range: x > 0 x < 1/2
    using c 2 by auto
  have φ x y = α + α
    unfolding φ-def 2 using x-range
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  moreover have φ y x = 1 + 1
    unfolding φ-def 2 using x-range
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  ultimately have ?T = cos (pi ∗ x) ∗ (2∗α)
    unfolding 2 by simp
  also have ... ≤ 1 ∗ (2∗ sqrt 2)
    unfolding α-def by (intro mult-right-mono) auto
  also have ... ≤ ?R
    by (approximation 5)
  finally show ?thesis by simp
next
  case 3
  have x-range: x ∈ {1/4..1/2} using 3 c by simp
  hence cos-bound: cos (pi ∗ x) ≤ 0.71
    by (approximation 10)
  have φ x y = 1+α
    unfolding φ-def 3 using 3 c
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  moreover have φ y x = 1+α
    unfolding φ-def 3 using 3 c
    by (intro arg-cong2[where f=(+)] γ'-cases) (auto simp add:frac-expand)
  ultimately have ?T = 2 ∗ cos (pi ∗ x) ∗ (1+α)
    unfolding 3 by simp
  also have ... ≤ 2 ∗ 0.71 ∗ (1+sqrt 2)
    unfolding α-def by (intro mult-right-mono mult-left-mono cos-bound) auto
  also have ... ≤ ?R
    by (approximation 6)
```

**finally show** *?thesis* **by** *simp*
**next**
**case** *4*
**show** *?thesis* **unfolding** *4* **by** *simp*
**qed**
**qed**
**thus** *?thesis* **using** *0* **by** *simp*
**qed**

Extend to square $[0, \frac{1}{2}] \times [0, \frac{1}{2}]$ using symmetry around x=y axis.

**lemma** *fun-bound-real-2*:
**assumes** $x \in \{0..1/2\}$ $y \in \{0..1/2\}$ $(x,y) \neq (0,0)$
**shows** $|cos(pi*x)|*\varphi\ x\ y + |cos(pi*y)|*\varphi\ y\ x \leq 2.5 * sqrt\ 2$ (**is** $?L \leq ?R$)
**proof** (*cases* $y < x$)
**case** *True*
**have** $?L = |cos(pi*y)|*\varphi\ y\ x + |cos(pi*x)|*\varphi\ x\ y$
**by** *simp*
**also have** ... $\leq ?R$
**using** *True assms*
**by** (*intro fun-bound-real-3*) *auto*
**finally show** *?thesis* **by** *simp*
**next**
**case** *False*
**then show** *?thesis* **using** *assms*
**by** (*intro fun-bound-real-3*) *auto*
**qed**

Extend to $x > \frac{1}{2}$ using symmetry around $x = \frac{1}{2}$ axis.

**lemma** *fun-bound-real-1*:
**assumes** $x \in \{0..<1\}$ $y \in \{0..1/2\}$ $(x,y) \neq (0,0)$
**shows** $|cos(pi*x)|*\varphi\ x\ y + |cos(pi*y)|*\varphi\ y\ x \leq 2.5 * sqrt\ 2$ (**is** $?L \leq ?R$)
**proof** (*cases* $x > 1/2$)
**case** *True*
**define** $x'$ **where** $x' = 1-x$

**have** $|frac\ (x - 2 * y) - 1\ /\ 2| = |frac\ (1 - x + 2 * y) - 1\ /\ 2|$
**proof** (*cases* $x - 2 * y \in \mathbb{Z}$)
**case** *True*
**then obtain** *k* **where** *x-eq*: $x = 2*y + of\text{-}int\ k$ **using** *Ints-cases*[*OF True*]
**by** (*metis add-minus-cancel uminus-add-conv-diff*)
**show** *?thesis* **unfolding** *x-eq frac-def* **by** *simp*
**next**
**case** *False*
**hence** $1 - x + 2 * y \notin \mathbb{Z}$
**using** *Ints-1 Ints-diff* **by** *fastforce*
**thus** *?thesis*
**by** (*intro abs-rev-cong*) (*auto intro*:*frac-cong simp*:*one-minus-frac*)
**qed**

**moreover have** $|frac\ (x + 2 * y) - 1\ /\ 2| = |frac\ (1 - x - 2 * y) - 1\ /\ 2|$
**proof** (*cases* $x + 2 * y \in \mathbb{Z}$)
**case** *True*
**then obtain** *k* **where** *x-eq*: $x = of\text{-}int\ k - 2*y$ **using** *Ints-cases*[*OF True*]
**by** (*metis add.right-neutral add-diff-eq cancel-comm-monoid-add-class.diff-cancel*)
**show** *?thesis* **unfolding** *x-eq frac-def* **by** *simp*
**next**
**case** *False*
**hence** $1 - x - 2 * y \notin \mathbb{Z}$

```
      using Ints-1 Ints-diff by fastforce
    thus ?thesis
      by (intro abs-rev-cong) (auto intro:frac-cong simp:one-minus-frac)
  qed
  ultimately have φ y x = φ y x′
    unfolding φ-def  x′-def by (intro γ′-cong add-swap-cong) simp-all

  moreover have φ x y = φ x′ y
    unfolding φ-def x′-def
    by (intro γ′-cong add-swap-cong refl arg-cong[where f=(λx. abs (x−1/2))] frac-cong)
     (simp-all add:algebra-simps)

  moreover have |cos(pi∗x)| = |cos(pi∗x′)|
    unfolding x′-def by (intro abs-rev-cong) (simp add:algebra-simps)

  ultimately have ?L = |cos(pi∗x′)|∗φ x′ y + |cos(pi∗y)|∗φ y x′
    by simp
  also have ... ≤ ?R
    using assms True by (intro fun-bound-real-2) (auto simp add:x′-def)
  finally show ?thesis by simp
next
  case False
  thus ?thesis using assms fun-bound-real-2 by simp
qed
```

Extend to $y > \frac{1}{2}$ using symmetry around $y = \frac{1}{2}$ axis.

```
lemma fun-bound-real:
  assumes x ∈ {0..<1} y ∈ {0..<1} (x,y) ≠ (0,0)
  shows |cos(pi∗x)|∗φ x y + |cos(pi∗y)|∗φ y x ≤ 2.5 ∗ sqrt 2  (is ?L ≤ ?R)
proof (cases y > 1/2)
  case True
  define y′ where y′ = 1−y

  have |frac (y − 2 ∗ x) − 1 / 2| = |frac (1 − y + 2 ∗ x) − 1 / 2|
  proof (cases y − 2 ∗ x ∈ ℤ)
    case True
    then obtain k where y-eq: y = 2∗x + of-int k using Ints-cases[OF True]
      by (metis add-minus-cancel uminus-add-conv-diff)
    show ?thesis unfolding y-eq frac-def by simp
  next
    case False
    hence 1 − y + 2 ∗ x ∉ ℤ
      using Ints-1 Ints-diff by fastforce
    thus ?thesis
      by (intro abs-rev-cong) (auto intro:frac-cong simp:one-minus-frac)
  qed

  moreover have |frac (y + 2 ∗ x) − 1 / 2| = |frac (1 − y − 2 ∗ x) − 1 / 2|
  proof (cases y + 2 ∗ x ∈ ℤ)
    case True
    then obtain k where y-eq: y = of-int k − 2∗x using Ints-cases[OF True]
      by (metis add.right-neutral add-diff-eq cancel-comm-monoid-add-class.diff-cancel)
    show ?thesis unfolding y-eq frac-def by simp
  next
    case False
    hence 1 − y − 2 ∗ x ∉ ℤ
      using Ints-1 Ints-diff by fastforce
    thus ?thesis
```

**by** (*intro abs-rev-cong*) (*auto intro*:*frac-cong simp*:*one-minus-frac*)
**qed**
**ultimately have** $\varphi\ x\ y = \varphi\ x\ y'$
**unfolding** $\varphi$*-def* $y'$*-def* **by** (*intro* $\gamma'$*-cong add-swap-cong*) *simp-all*

**moreover have** $\varphi\ y\ x = \varphi\ y'\ x$
**unfolding** $\varphi$*-def* $y'$*-def*
**by** (*intro* $\gamma'$*-cong add-swap-cong refl arg-cong*[**where** $f{=}(\lambda x.\ abs\ (x{-}1/2))$] *frac-cong*)
(*simp-all add*:*algebra-simps*)

**moreover have** $|cos(pi{*}y)| = |cos(pi{*}y')|$
**unfolding** $y'$*-def* **by** (*intro abs-rev-cong*) (*simp add*:*algebra-simps*)

**ultimately have** $?L = |cos(pi{*}x)|{*}\varphi\ x\ y' + |cos(pi{*}y')|{*}\varphi\ y'\ x$
**by** *simp*
**also have** ... $\leq$ *?R*
**using** *assms True* **by** (*intro fun-bound-real-1*) (*auto simp add*:$y'$*-def*)
**finally show** *?thesis* **by** *simp*
**next**
**case** *False*
**thus** *?thesis* **using** *assms fun-bound-real-1* **by** *simp*
**qed**

**lemma** *mod-to-frac*:
**fixes** $x$ :: *int*
**shows** *real-of-int* ($x$ *mod* $m$) = $m$ $*$ *frac* ($x/m$) (**is** *?L* = *?R*)
**proof** −
**obtain** $y$ **where** *y-def*: $x$ *mod* $m$ = $x$ + *int* $m{*}$ $y$
**by** (*metis mod-eqE mod-mod-trivial*)

**have** *0*: $x$ *mod int* $m$ < $m$ $x$ *mod int* $m \geq 0$
**using** *m-gt-0* **by** *auto*

**have** *?L* = *real* $m$ $*$ (*of-int* ($x$ *mod* $m$) / $m$ )
**using** *m-gt-0* **by** (*simp add*:*algebra-simps*)
**also have** ... = *real* $m$ $*$ *frac* (*of-int* ($x$ *mod* $m$) / $m$)
**using** *0* **by** (*subst iffD2*[*OF frac-eq*]) *auto*
**also have** ... = *real* $m$ $*$ *frac* ($x$ / $m$ + $y$)
**unfolding** *y-def* **using** *m-gt-0* **by** (*simp add*:*divide-simps mult.commute*)
**also have** ... = *?R*
**unfolding** *frac-def* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *fun-bound*:
**assumes** $v \in$ *verts G* $v \neq (0,0)$
**shows** $\tau(fst\ v){*}(\gamma\ v\ (S_2\ v){+}\gamma\ v\ (T_2\ v)){+}\tau(snd\ v){*}(\gamma\ v\ (S_1\ v){+}\gamma\ v\ (T_1\ v)) \leq 2.5 * sqrt\ 2$
(**is** *?L* $\leq$ *?R*)
**proof** −
**obtain** $x$ $y$ **where** *v-def*: $v = (x,y)$ **by** (*cases* $v$) *auto*
**define** $x'$ **where** $x' = x/real\ m$
**define** $y'$ **where** $y' = y/real\ m$

**have** *0*:$\gamma\ v\ (S_1\ v) = \gamma'\ x'\ (frac(x'{-}2{*}y'))$
**unfolding** $\gamma$*-def* $\gamma'$*-def compare-def v-def* $\gamma$*-aux-def* $T_1$*-def* $S_1$*-def* $x'$*-def* $y'$*-def* **using** *m-gt-0*
**by** (*intro if-cong-direct refl*) (*auto simp add*:*case-prod-beta mod-to-frac divide-simps*)
**have** *1*:$\gamma\ v\ (T_1\ v) = \gamma'\ x'\ (frac(x'{+}2{*}y'))$
**unfolding** $\gamma$*-def* $\gamma'$*-def compare-def v-def* $\gamma$*-aux-def* $T_1$*-def* $x'$*-def* $y'$*-def* **using** *m-gt-0*

**by** (*intro if-cong-direct refl*) (*auto simp add*:*case-prod-beta mod-to-frac divide-simps*)
**have** *2*:$\gamma$ *v* ($S_2$ *v*) = $\gamma'$ *y'* (*frac*($y'$−*2*∗$x'$))
**unfolding** $\gamma$*-def* $\gamma'$*-def compare-def v-def* $\gamma$*-aux-def* $S_2$*-def* $x'$*-def* $y'$*-def* **using** *m-gt-0*
**by** (*intro if-cong-direct refl*) (*auto simp add*:*case-prod-beta mod-to-frac divide-simps*)
**have** *3*:$\gamma$ *v* ($T_2$ *v*) = $\gamma'$ *y'* (*frac*($y'$+*2*∗$x'$))
**unfolding** $\gamma$*-def* $\gamma'$*-def compare-def v-def* $\gamma$*-aux-def* $T_2$*-def* $x'$*-def* $y'$*-def* **using** *m-gt-0*
**by** (*intro if-cong-direct refl*) (*auto simp add*:*case-prod-beta mod-to-frac divide-simps*)
**have** *4*: $\tau$ (*fst v*) = |*cos*(*pi*∗$x'$)| $\tau$ (*snd v*) = |*cos*(*pi*∗$y'$)|
**unfolding** $\tau$*-def v-def* $x'$*-def* $y'$*-def* **by** *auto*

**have** *x* ∈ {*0*..<*int m*} *y* ∈ {*0*..<*int m*} (*x*,*y*) ≠ (*0*,*0*)
**using** *assms* **unfolding** *v-def mgg-graph-def* **by** *auto*
**hence** *5*:$x'$ ∈ {*0*..<*1*} $y'$ ∈ {*0*..<*1*} ($x'$,$y'$) ≠ (*0*,*0*)
**unfolding** $x'$*-def* $y'$*-def* **by** *auto*

**have** *?L* = |*cos*(*pi*∗$x'$)|∗$\varphi$ $x'$ $y'$ + |*cos*(*pi*∗$y'$)|∗$\varphi$ $y'$ $x'$
**unfolding** *0 1 2 3 4* $\varphi$*-def* **by** *simp*
**also have** ... ≤ *?R*
**by** (*intro fun-bound-real 5*)
**finally show** *?thesis* **by** *simp*
**qed**

Equation 15 in Proof of Theorem 8.8

**lemma** *hoory-8-8*:
**fixes** *f* :: *int* × *int* ⇒ *real*
**assumes** ⋀*x*. *f x* ≥ *0*
**assumes** *f* (*0*,*0*) = *0*
**assumes** *periodic f*
**shows** *g-inner f* (λ*x*. *f*($S_2$ *x*)∗$\tau$ (*fst x*)+*f*($S_1$ *x*)∗$\tau$ (*snd x*))≤*1.25*∗ *sqrt 2*∗*g-norm f*^*2*
(**is** *?L* ≤ *?R*)
**proof** −
**have** *0*: *2* ∗ *f x* ∗ *f y* ≤ $\gamma$ *x y* ∗ *f x*^*2* + $\gamma$ *y x* ∗ *f y*^*2* (**is** *?L1* ≤ *?R1*) **for** *x y*
**proof** −
**have** *0* ≤ ((*sqrt* ($\gamma$ *x y*) ∗ *f x*) − (*sqrt* ($\gamma$ *y x*) ∗ *f y*))^*2*
**by** *simp*
**also have** ... = *?R1* − *2* ∗ (*sqrt* ($\gamma$ *x y*) ∗ *f x*) ∗ (*sqrt* ($\gamma$ *y x*) ∗ *f y*)
**unfolding** *power2-diff* **using** $\gamma$*-nonneg assms*(*1*)
**by** (*intro arg-cong2*[**where** *f*=(−)] *arg-cong2*[**where** *f*=(+)]) (*auto simp add*: *power2-eq-square*)
**also have** ... = *?R1* −*2* ∗ *sqrt* ($\gamma$ *x y* ∗ $\gamma$ *y x*) ∗ *f x* ∗ *f y*
**unfolding** *real-sqrt-mult* **by** *simp*
**also have** ... = *?R1* − *?L1*
**unfolding** $\gamma$*-sym* **by** *simp*
**finally have** *0* ≤ *?R1* − *?L1* **by** *simp*
**thus** *?thesis* **by** *simp*
**qed**

**have** [*simp*]: *fst* ($S_2$ *x*) = *fst x* *snd* ($S_1$ *x*) = *snd x* **for** *x*
**unfolding** $S_1$*-def* $S_2$*-def* **by** *auto*

**have** *S-2-inv* [*simp*]: $T_2$ ($S_2$ *x*) = *x* **if** *x* ∈ *verts G* **for** *x*
**using** *that* **unfolding** $T_2$*-def* $S_2$*-def mgg-graph-def*
**by** (*cases x*,*simp add*:*mod-simps*)
**have** *S-1-inv* [*simp*]: $T_1$ ($S_1$ *x*) = *x* **if** *x* ∈ *verts G* **for** *x*
**using** *that* **unfolding** $T_1$*-def* $S_1$*-def mgg-graph-def*
**by** (*cases x*,*simp add*:*mod-simps*)

**have** *S2-inj*: *inj-on* $S_2$ (*verts G*)
**using** *S-2-inv* **by** (*intro inj-on-inverseI*[**where** *g*=$T_2$])

```
  have S1-inj: inj-on S₁ (verts G)
    using S-1-inv by (intro inj-on-inverseI[where g=T₁])

  have S₂ ` verts G ⊆ verts G
    unfolding mgg-graph-def S₂-def
    by (intro image-subsetI) auto
  hence S2-ran: S₂ ` verts G = verts G
    by (intro card-subset-eq card-image S2-inj) auto

  have S₁ ` verts G ⊆ verts G
    unfolding mgg-graph-def S₁-def
    by (intro image-subsetI) auto
  hence S1-ran: S₁ ` verts G = verts G
    by (intro card-subset-eq card-image S1-inj) auto

  have 2: g v ∗ f v^2 ≤ 2.5 ∗ sqrt 2 ∗ f v^2 if g v ≤ 2.5 ∗ sqrt 2 ∨ v = (0,0) for v g
  proof (cases v=(0,0))
    case True
    then show ?thesis using assms(2) by simp
  next
    case False
    then show ?thesis using that by (intro mult-right-mono) auto
  qed

  have 2∗?L=(∑ v∈verts G. τ(fst v)∗(2∗f v ∗f(S₂ v)))+(∑ v∈verts G. τ(snd v) ∗ (2 ∗ f v ∗ f
(S₁ v)))
    unfolding g-inner-def by (simp add: algebra-simps sum-distrib-left sum.distrib)
  also have ... ≤
    (∑ v∈verts G. τ(fst v)∗(γ v (S₂ v) ∗ f v^2 + γ (S₂ v) v ∗ f(S₂ v)^2))+
    (∑ v∈verts G. τ(snd v)∗(γ v (S₁ v) ∗ f v^2 + γ (S₁ v) v ∗ f(S₁ v)^2))
    unfolding τ-def by (intro add-mono sum-mono mult-left-mono 0) auto
  also have ... =
    (∑ v∈verts G. τ(fst v)∗γ v (S₂ v)∗f v^2)+(∑ v∈verts G. τ(fst v) ∗ γ (S₂ v) v ∗ f(S₂ v)^2)+
    (∑ v∈verts G. τ(snd v)∗γ v (S₁ v)∗f v^2)+(∑ v∈verts G. τ(snd v) ∗ γ (S₁ v) v ∗ f(S₁ v)^2)
    by (simp add:sum.distrib algebra-simps)
  also have ... =
    (∑ v∈verts G. τ(fst v)∗γ v (S₂ v)∗f v^2)+
    (∑ v∈verts G. τ(fst (S₂ v)) ∗ γ (S₂ v) (T₂ (S₂ v)) ∗ f(S₂ v)^2)+
    (∑ v∈verts G. τ(snd v)∗γ v (S₁ v)∗f v^2)+
    (∑ v∈verts G. τ(snd (S₁ v)) ∗ γ (S₁ v) (T₁ (S₁ v)) ∗ f(S₁ v)^2)
    by (intro arg-cong2[where f=(+)] sum.cong refl) simp-all
  also have ... =
   (∑ v∈verts G. τ(fst v)∗γ v (S₂ v)∗f v^2)+ (∑ v∈S₂ ` verts G. τ(fst v) ∗ γ v (T₂ v) ∗ f v^2)+
   (∑ v∈verts G. τ(snd v)∗γ v (S₁ v)∗f v^2)+ (∑ v∈S₁ ` verts G. τ(snd v) ∗ γ v (T₁ v) ∗ f v^2)
    using S1-inj S2-inj by (simp add:sum.reindex)
  also have ... =
    (∑ v∈verts G. (τ(fst v)∗(γ v (S₂ v)+γ v (T₂ v))+τ(snd v)∗(γ v (S₁ v)+γ v (T₁ v))) ∗f v^2)
    unfolding S1-ran S2-ran by (simp add:algebra-simps sum.distrib)
  also have ... ≤ (∑ v∈verts G. 2.5 ∗ sqrt 2 ∗ f v^2)
    using fun-bound by (intro sum-mono 2) auto
  also have ... ≤  2.5 ∗ sqrt 2 ∗ g-norm f^2
    unfolding g-norm-sq g-inner-def
    by (simp add:algebra-simps power2-eq-square sum-distrib-left)
  finally have 2 ∗ ?L ≤ 2.5 ∗ sqrt 2 ∗ g-norm f^2 by simp
  thus ?thesis by simp
qed

lemma hoory-8-7:
```

```
  fixes f :: int×int ⇒ complex
  assumes f (0,0) = 0
  assumes periodic f
  shows norm(g-inner f (λx. f (S_2 x) ∗ (1+ω_F (fst x)) + f (S_1 x) ∗ (1+ω_F (snd x))))
    ≤ (2.5 ∗ sqrt 2) ∗ (∑ v ∈ verts G. norm (f v)^2) (is ?L ≤ ?R)
proof −
  define g :: int×int ⇒ real where g x = norm (f x) for x

  have g-zero: g (0,0) = 0
    using assms(1) unfolding g-def by simp
  have g-nonneg: g x ≥ 0 for x
    unfolding g-def by simp
  have g-periodic: periodic g
    unfolding g-def by (intro periodic-comp[OF assms(2)])

  have 0: norm(1+ω_F x) = 2∗τ x for x :: int
  proof −
    have norm(1+ω_F x) = norm(ω_F (−x/2)∗(ω_F 0 + ω_F x))
      unfolding ω_F-def norm-mult by simp
    also have ... = norm (ω_F (0−x/2) + ω_F (x−x/2))
      unfolding ω_F-simps by (simp add: algebra-simps)
    also have ... = norm (ω_F (x/2) + cnj (ω_F (x/2)))
      unfolding ω_F-simps(3) by (simp add:algebra-simps)
    also have ... = |2∗Re (ω_F (x/2))|
      unfolding complex-add-cnj norm-of-real by simp
    also have ... =  2∗|cos(pi∗x/m)|
      unfolding ω_F-def cis.simps by simp
    also have ... = 2∗τ x unfolding τ-def by simp
    finally show ?thesis by simp
  qed

  have ?L≤norm(∑ v∈verts G. f v ∗ cnj(f(S_2 v)∗(1+ω_F (fst v))+f(S_1 v )∗(1+ω_F (snd v))))
    unfolding g-inner-def by (simp add:case-prod-beta)
  also have ...≤(∑ v∈verts G. norm(f v ∗ cnj(f (S_2 v) ∗(1+ω_F (fst v))+f (S_1 v)∗(1+ω_F (snd
v)))))
    by (intro norm-sum)
  also have ...=(∑ v∈verts G. g v ∗ norm(f (S_2 v) ∗(1+ω_F (fst v))+f (S_1 v)∗(1+ω_F (snd v))))
    unfolding norm-mult g-def complex-mod-cnj by simp
  also have ...≤(∑ v∈verts G. g v ∗ (norm (f(S_2 v)∗(1+ω_F (fst v)))+norm(f(S_1 v)∗(1+ω_F(snd
v)))))
    by (intro sum-mono norm-triangle-ineq mult-left-mono g-nonneg)
  also have ...=2∗g-inner g (λx. g (S_2 x)∗τ (fst x)+g(S_1 x)∗τ (snd x))
    unfolding g-def g-inner-def norm-mult 0
    by (simp add:sum-distrib-left algebra-simps case-prod-beta)
  also have ... ≤2∗(1.25∗ sqrt 2∗g-norm g^2)
    by (intro mult-left-mono hoory-8-8 g-nonneg g-zero g-periodic) auto
  also have ... = ?R
    unfolding g-norm-sq g-def g-inner-def by (simp add:power2-eq-square)
  finally show ?thesis by simp
qed

lemma hoory-8-3:
  assumes g-inner f (λ-. 1) = 0
  assumes periodic f
  shows |(∑ (x,y)∈verts G. f(x,y)∗(f(x+2∗y,y)+f(x+2∗y+1,y)+f(x,y+2∗x)+f(x,y+2∗x+1)))|
    ≤ (2.5 ∗ sqrt 2) ∗ g-norm f^2 (is |?L| ≤ ?R)
proof −
  let ?f = (λx. complex-of-real (f x))
```

**define** *Ts* :: (*int* × *int* ⇒ *int* × *int*) *list* **where**
*Ts* = [(λ(*x*,*y*).(*x*+*2*∗*y*,*y*)),(λ(*x*,*y*).(*x*+*2*∗*y*+*1*,*y*)),(λ(*x*,*y*).(*x*,*y*+*2*∗*x*)),(λ(*x*,*y*).(*x*,*y*+*2*∗*x*+*1*))]
**have** *p*: *periodic ?f*
**by** (*intro periodic-comp*[*OF assms*(*2*)])

**have** *0*: (∑ *T*←*Ts*. *FT* (*?f*∘*T*) *v*) = *FT ?f* ($S_2$ *v*)∗(*1*+$\omega_F$ (*fst v*))+*FT ?f* ($S_1$ *v*)∗(*1*+$\omega_F$ (*snd v*))
(**is** *?L1* = *?R1*) **for** *v* :: *int* × *int*
**proof** −
**obtain** *x y* **where** *v-def*: *v* = (*x*,*y*) **by** (*cases v*, *auto*)
**have** *?L1* = (∑ *T*←*Ts*. *FT* (*?f*∘*T*) (*x*,*y*))
**unfolding** *v-def* **by** *simp*
**also have** ... = *FT ?f* (*x*,*y*−*2*∗*x*)∗(*1*+$\omega_F$ *x*) + *FT ?f* (*x*−*2*∗*y*,*y*)∗(*1*+$\omega_F$ *y*)
**unfolding** *Ts-def* **by** (*simp add*:*FT-sheer*[*OF p*] *case-prod-beta comp-def*) (*simp add*:*algebra-simps*)
**also have** ... = *?R1*
**unfolding** *v-def* $S_2$*-def* $S_1$*-def*
**by** (*intro arg-cong2*[**where** *f*=(+)] *arg-cong2*[**where** *f*=(∗)] *periodic-cong*[*OF periodic-FT*]) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**have** *cmod* ((*of-nat m*)^*2*) = *cmod* (*of-real* (*of-nat m*^*2*)) **by** *simp*
**also have** ... = *abs* (*of-nat m*^*2*) **by** (*intro norm-of-real*)
**also have** ... = *real m*^*2* **by** *simp*
**finally have** *1*: *cmod* ((*of-nat m*)$^2$) = (*real m*)$^2$ **by** *simp*

**have** *FT* (λ*x*. *complex-of-real* (*f x*)) (*0*, *0*) = *complex-of-real* (*g-inner f* (λ- . *1*))
**unfolding** *FT-def g-inner-def g-inner-def* $\omega_F$*-def* **by** *simp*
**also have** ... = *0*
**unfolding** *assms* **by** *simp*
**finally have** *2*: *FT* (λ*x*. *complex-of-real* (*f x*)) (*0*, *0*) = *0*
**by** *simp*

**have** *abs ?L* = *norm* (*complex-of-real ?L*)
**unfolding** *norm-of-real* **by** *simp*
**also have** ... = *norm* (∑ *T* ← *Ts*. (*g-inner ?f* (*?f* ∘ *T*)))
**unfolding** *Ts-def* **by** (*simp add*:*algebra-simps g-inner-def sum.distrib comp-def case-prod-beta*)
**also have** ... = *norm* (∑ *T* ← *Ts*. (*g-inner* (*FT ?f*) (*FT* (*?f* ∘ *T*)))/*m*^*2*)
**by** (*subst parseval*) *simp*
**also have** ... = *norm* (*g-inner* (*FT ?f*) (λ*x*. (∑ *T* ← *Ts*. (*FT* (*?f* ∘ *T*) *x*)))/*m*^*2*)
**unfolding** *Ts-def* **by** (*simp add*:*g-inner-simps case-prod-beta add-divide-distrib*)
**also have** ...=*norm*(*g-inner*(*FT ?f*)(λ*x*.(*FT ?f*($S_2$ *x*)∗(*1*+$\omega_F$ (*fst x*))+*FT f*($S_1$ *x*)∗(*1*+$\omega_F$ (*snd x*)))))/*m*^*2*
**by** (*subst 0*) (*simp add*:*norm-divide 1*)
**also have** ... ≤ (*2.5* ∗ *sqrt 2*) ∗ (∑ *v* ∈ *verts G*. *norm* (*FT f v*)^*2*) / *m*^*2*
**by** (*intro divide-right-mono hoory-8-7*[**where** *f*=*FT f*] *2 periodic-FT*) *auto*
**also have** ... = (*2.5* ∗ *sqrt 2*) ∗ (∑ *v* ∈ *verts G*. *cmod* (*f v*)^*2*)
**by** (*subst* (*2*) *plancharel*) *simp*
**also have** ... = (*2.5* ∗ *sqrt 2*) ∗ (*g-inner f f*)
**unfolding** *g-inner-def norm-of-real* **by** (*simp add*: *power2-eq-square*)
**also have** ... = *?R*
**using** *g-norm-sq* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**

Inequality stated before Theorem 8.3 in Hoory.

**lemma** *mgg-numerical-radius-aux*:

**assumes** *g-inner f* (λ-. *1*) = *0*
**shows** $|(\sum a \in arcs\ G.\ f\ (head\ G\ a) * f\ (tail\ G\ a))| \leq (5 * sqrt\ 2) * g\text{-}norm\ f\hat{}2$ (**is** *?L* ≤ *?R*)
**proof** −
**define** *g* **where** *g x* = *f* (*fst x mod m*, *snd x mod m*) **for** *x* :: *int* × *int*
**have** *0*:*g x* = *f x* **if** *x* ∈ *verts G* **for** *x*
**unfolding** *g-def* **using** *that*
**by** (*auto simp add*:*mgg-graph-def mem-Times-iff*)

**have** *g-mod-simps*[*simp*]: *g* (*x*, *y mod m*) = *g* (*x*, *y*) *g* (*x mod m*, *y*) = *g* (*x*, *y*) **for** *x y* :: *int*
**unfolding** *g-def* **by** *auto*

**have** *periodic-g*: *periodic g*
**unfolding** *periodic-def* **by** *simp*

**have** *g-inner g* (λ-. *1*) = *g-inner f* (λ-. *1*)
**by** (*intro g-inner-cong 0*) *auto*
**also have** ... = *0*
**using** *assms* **by** *simp*
**finally have** *1*:*g-inner g* (λ-. *1*) = *0* **by** *simp*

**have** *2*:*g-norm g* = *g-norm f*
**by** (*intro g-norm-cong 0*) (*auto*)

**have** *?L* = $|(\sum a \in arcs\ G.\ g\ (head\ G\ a) * g\ (tail\ G\ a))|$
**using** *wellformed*
**by** (*intro arg-cong*[**where** *f*=*abs*] *sum.cong arg-cong2*[**where** *f*=(∗)] *0*[*symmetric*]) *auto*
**also have** ...=$|(\sum a{\in}arcs\text{-}pos.\ g(head\ G\ a){*}g(tail\ G\ a))+(\sum a{\in}arcs\text{-}neg.\ g(head\ G\ a){*}g(tail\ G\ a))|$
**unfolding** *arcs-sym arcs-pos-def arcs-neg-def*
**by** (*intro arg-cong*[**where** *f*=*abs*] *sum.union-disjoint*) *auto*
**also have** ... = $|2 * (\sum (v,l){\in}verts\ G \times \{..{<}4\}.\ g\ v * g\ (mgg\text{-}graph\text{-}step\ m\ v\ (l,\ 1)))|$
**unfolding** *arcs-pos-def arcs-neg-def*
**by** (*simp add*:*inj-on-def sum.reindex case-prod-beta mgg-graph-def algebra-simps*)
**also have** ... = $2 * |(\sum v \in verts\ G.\ (\sum l \in \{..{<}4\}.\ g\ v * g\ (mgg\text{-}graph\text{-}step\ m\ v\ (l,\ 1))))|$
**by** (*subst sum.cartesian-product*) (*simp add*:*abs-mult*)
**also have** ... = $2{*}|(\sum (x,y){\in}verts\ G.\ (\sum l{\leftarrow}[0..{<}4].\ g(x,y){*}\ g\ (mgg\text{-}graph\text{-}step\ m\ (x,y)\ (l,1))))|$
**by** (*subst interv-sum-list-conv-sum-set-nat*)
(*auto simp add*:*atLeast0LessThan case-prod-beta simp del*:*mgg-graph-step.simps*)
**also have** ... =$2{*}|\sum (x,y){\in}verts\ G.\ g\ (x,y){*}\ (g(x{+}2{*}y,y){+}g(x{+}2{*}y{+}1,y){+}g(x,y{+}2{*}x){+}g(x,y{+}2{*}x{+}1))|$
**by** (*simp add*:*case-prod-beta numeral-eq-Suc algebra-simps*)
**also have** ... ≤ $2{*}\ ((2.5 * sqrt\ 2) * g\text{-}norm\ g\hat{}2)$
**by** (*intro mult-left-mono hoory-8-3 1 periodic-g*) *auto*
**also have** ... ≤ *?R* **unfolding** *2* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**definition** *MGG-bound* :: *real*
**where** *MGG-bound* = *5* ∗ *sqrt 2* / *8*

Main result: Theorem 8.2 in Hoory.

**lemma** *mgg-numerical-radius*: $\Lambda_a \leq$ *MGG-bound*
**proof** −
**have** $\Lambda_a \leq (5 * sqrt\ 2)/real\ d$
**by** (*intro expander-intro mgg-numerical-radius-aux*) *auto*
**also have** ... = *MGG-bound*
**unfolding** *MGG-bound-def d-eq-8* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

```
end

end
```

# 9 Random Walks

```
theory Expander-Graphs-Walks
  imports
    Expander-Graphs-Algebra
    Expander-Graphs-Eigenvalues
    Expander-Graphs-TTS
    Constructive-Chernoff-Bound
begin

unbundle intro-cong-syntax

no-notation Matrix.vec-index (infixl ‹$› 100)
hide-const Matrix.vec-index
hide-const Matrix.vec
no-notation Matrix.scalar-prod (infix ‹·› 70)

fun walks' :: ('a,'b) pre-digraph ⇒ nat ⇒ ('a list) multiset
  where
    walks' G 0 = image-mset (λx. [x]) (mset-set (verts G)) |
    walks' G (Suc n) =
      concat-mset {#{#w @[z].z∈# vertices-from G (last w)#}. w ∈# walks' G n#}

definition walks G l = (case l of 0 ⇒ {#[]#} | Suc pl ⇒ walks' G pl)

lemma Union-image-mono: (⋀x. x ∈ A ⟹ f x ⊆ g x) ⟹ ⋃ (f ` A) ⊆ ⋃ (g ` A)
  by auto

context fin-digraph
begin

lemma count-walks':
  assumes set xs ⊆ verts G
  assumes length xs = l+1
  shows count (walks' G l) xs = (∏ i ∈ {..<l}. count (edges G) (xs ! i, xs ! (i+1)))
proof -
  have a:xs ≠ [] using assms(2) by auto

  have count (walks' G (length xs−1)) xs = (∏ i<length xs −1. count (edges G) (xs ! i, xs ! (i
+ 1)))
    using a assms(1)
  proof (induction xs rule:rev-nonempty-induct)
    case (single x)
    hence x ∈ verts G by simp
    hence count {#[x]. x ∈# mset-set (verts G)#} [x] = 1
      by (subst count-image-mset-inj, auto simp add:inj-def)
    then show ?case by simp
  next
    case (snoc x xs)
    have set-xs: set xs ⊆ verts G using snoc by simp

    define l where l = length xs − 1
```

**have** *l-xs*: *length xs* = *l* + *1* **unfolding** *l-def* **using** *snoc* **by** *simp*
**have** *count* (*walks′ G* (*length* (*xs* @ [*x*]) − *1*)) (*xs* @ [*x*]) =
($\sum$ *ys*∈#*walks′ G l*. *count* {#*ys* @ [*z*]. *z* ∈# *vertices-from G* (*last ys*)#} (*xs* @ [*x*]))
**by** (*simp add*:*l-xs count-concat-mset image-mset.compositionality comp-def*)
**also have** ... = ($\sum$ *ys*∈#*walks′ G l*.
(*if ys* = *xs then count* {#*xs* @ [*z*]. *z* ∈# *vertices-from G* (*last xs*)#} (*xs* @ [*x*]) *else 0*))
**by** (*intro arg-cong*[**where** *f*=*sum-mset*] *image-mset-cong*) (*auto intro*!: *count-image-mset-0-triv*)
**also have** ... = ($\sum$ *ys*∈#*walks′ G l*.(*if ys*=*xs then count* (*vertices-from G* (*last xs*)) *x else 0*))
**by** (*subst count-image-mset-inj*, *auto simp add*:*inj-def*)
**also have** ... = *count* (*walks′ G l*) *xs* ∗ *count* (*vertices-from G* (*last xs*)) *x*
**by** (*subst sum-mset-delta*, *simp*)
**also have** ... = *count* (*walks′ G l*) *xs* ∗ *count* (*edges G*) (*last xs*, *x*)
**unfolding** *vertices-from-def count-mset-exp image-mset-filter-mset-swap*[*symmetric*]
*filter-filter-mset* **by** (*simp add*:*prod-eq-iff*)
**also have** ... = *count* (*walks′ G l*) *xs* ∗ *count* (*edges G*) ((*xs*@[*x*])!*l*, (*xs*@[*x*])!(*l*+*1*))
**using** *snoc*(*1*) **unfolding** *l-def nth-append last-conv-nth*[*OF snoc*(*1*)] **by** *simp*
**also have** ... = ($\prod$ *i*<*l*+*1*. *count* (*edges G*) ((*xs*@[*x*])!*i*, (*xs*@[*x*])!(*i*+*1*)))
**unfolding** *l-def snoc*(*2*)[*OF set-xs*] **by** (*simp add*:*nth-append*)
**finally have** *count* (*walks′ G* (*length* (*xs* @ [*x*]) − *1*)) (*xs* @ [*x*]) =
($\prod$ *i*<*length* (*xs*@[*x*]) − *1*. *count* (*edges G*) ((*xs*@[*x*])!*i*, (*xs*@[*x*])!(*i*+*1*)))
**unfolding** *l-def* **using** *snoc*(*1*) **by** *simp*
**then show** *?case* **by** *simp*
**qed**
**moreover have** *l* = *length xs* − *1* **using** *a assms* **by** *simp*
**ultimately show** *?thesis* **by** *simp*
**qed**

**lemma** *count-walks*:
**assumes** *set xs* ⊆ *verts G*
**assumes** *length xs* = *l l* > *0*
**shows** *count* (*walks G l*) *xs* = ($\prod$ *i* ∈ {..<*l*−*1*}. *count* (*edges G*) (*xs* ! *i*, *xs* ! (*i*+*1*)))
**using** *assms* **unfolding** *walks-def* **by** (*cases l*, *auto simp add*:*count-walks′*)

**lemma** *set-walks′*:
*set-mset* (*walks′ G l*) ⊆ {*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = (*l*+*1*)}
**proof** (*induction l*)
**case** *0*
**then show** *?case* **by** *auto*
**next**
**case** (*Suc l*)

**have** *set-mset* (*walks′ G* (*Suc l*)) =
($\bigcup$ *x*∈*set-mset* (*walks′ G l*). (λ*z*. *x* @ [*z*]) ‘ *set-mset* (*vertices-from G* (*last x*)))
**by** (*simp add*:*set-mset-concat-mset*)
**also have** ... ⊆ ($\bigcup$ *x*∈{*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = *l* + *1*}.
(λ*z*. *x* @ [*z*]) ‘ *set-mset* (*vertices-from G* (*last x*)))
**by** (*intro Union-mono image-mono Suc*)
**also have** ... ⊆ ($\bigcup$ *x*∈{*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = *l* + *1*}. (λ*z*. *x* @ [*z*]) ‘ *verts G*)
**by** (*intro Union-image-mono image-mono set-mset-vertices-from*)
**also have** ... ⊆ {*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = (*Suc l* + *1*)}
**by** (*intro subsetI*) *auto*
**finally show** *?case* **by** *simp*
**qed**

**lemma** *set-walks*:
*set-mset* (*walks G l*) ⊆ {*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = *l*}
**unfolding** *walks-def* **using** *set-walks′* **by** (*cases l*, *auto*)

**lemma** *set-walks-2*:
  **assumes** $xs \in\# walks'\ G\ l$
  **shows** $set\ xs \subseteq verts\ G\ xs \neq []$
**proof** −
  **have** $a{:}xs \in set\text{-}mset\ (walks'\ G\ l)$
    **using** *assms* **by** *simp*
  **thus** $set\ xs \subseteq verts\ G$
    **using** *set-walks'* **by** *auto*
  **have** $length\ xs \neq 0$
    **using** *set-walks' a* **by** *fastforce*
  **thus** $xs \neq []$ **by** *simp*
**qed**

**lemma** *set-walks-3*:
  **assumes** $xs \in\# walks\ G\ l$
  **shows** $set\ xs \subseteq verts\ G\ length\ xs = l$
  **using** *set-walks assms* **by** *auto*
**end**

**lemma** *measure-pmf-of-multiset*:
  **assumes** $A \neq \{\#\}$
  **shows** $measure\ (pmf\text{-}of\text{-}multiset\ A)\ S = real\ (size\ (filter\text{-}mset\ (\lambda x.\ x \in S)\ A))\ /\ size\ A$
    (**is** $?L = ?R$)
**proof** −
  **have** $sum\ (count\ A)\ (S \cap set\text{-}mset\ A) = size\ (filter\text{-}mset\ (\lambda x.\ x \in S \cap set\text{-}mset\ A)\ A)$
    **by** (*intro sum-count-2*) *simp*
  **also have** ... $= size\ (filter\text{-}mset\ (\lambda x.\ x \in S)\ A)$
    **by** (*intro arg-cong*[**where** *f=size*] *filter-mset-cong*) *auto*
  **finally have** *a*: $sum\ (count\ A)\ (S \cap set\text{-}mset\ A) = size\ (filter\text{-}mset\ (\lambda x.\ x \in S)\ A)$
    **by** *simp*

  **have** $?L = measure\ (pmf\text{-}of\text{-}multiset\ A)\ (S \cap set\text{-}mset\ A)$
    **using** *assms* **by** (*intro measure-eq-AE AE-pmfI*) *auto*
  **also have** ... $= sum\ (pmf\ (pmf\text{-}of\text{-}multiset\ A))\ (S \cap set\text{-}mset\ A)$
    **by** (*intro measure-measure-pmf-finite*) *simp*
  **also have** ... $= (\sum x \in S \cap set\text{-}mset\ A.\ count\ A\ x\ /\ size\ A)$
    **using** *assms* **by** (*intro sum.cong*, *auto*)
  **also have** ... $= (\sum x \in S \cap set\text{-}mset\ A.\ count\ A\ x)\ /\ size\ A$
    **by** (*simp add*:*sum-divide-distrib*)
  **also have** ... $= ?R$
    **using** *a* **by** *simp*
  **finally show** *?thesis*
    **by** *simp*
**qed**

**lemma** *pmf-of-multiset-image-mset*:
  **assumes** $A \neq \{\#\}$
  **shows** $pmf\text{-}of\text{-}multiset\ (image\text{-}mset\ f\ A) = map\text{-}pmf\ f\ (pmf\text{-}of\text{-}multiset\ A)$
  **using** *assms* **by** (*intro pmf-eqI*) (*simp add*:*pmf-map measure-pmf-of-multiset count-mset-exp*
    *image-mset-filter-mset-swap*[*symmetric*])

**context** *regular-graph*
**begin**

**lemma** *size-walks'*:
  $size\ (walks'\ G\ l) = card\ (verts\ G) * d\hat{\ }l$
**proof** (*induction l*)

**case** *0*
**then show** *?case* **by** *simp*
**next**
**case** (*Suc l*)
**have** *a*:*out-degree G* (*last x*) = *d* **if** *x* ∈# *walks' G l* **for** *x*
**proof** −
**have** *last x* ∈ *verts G*
**using** *set-walks-2 that* **by** *fastforce*
**thus** *?thesis*
**using** *reg* **by** *simp*
**qed**

**have** *size* (*walks' G* (*Suc l*)) = ($\sum$ *x*∈#*walks' G l*. *out-degree G* (*last x*))
**by** (*simp add*:*size-concat-mset image-mset.compositionality comp-def verts-from-alt out-degree-def*)
**also have** ... = ($\sum$ *x*∈#*walks' G l*. *d*)
**by** (*intro arg-cong*[**where** *f*=*sum-mset*] *image-mset-cong a*) *simp*
**also have** ... = *size* (*walks' G l*) ∗ *d* **by** *simp*
**also have** ... = *card* (*verts G*) ∗ *d*^(*Suc l*) **using** *Suc* **by** *simp*
**finally show** *?case* **by** *simp*
**qed**

**lemma** *size-walks*:
*size* (*walks G l*) = (*if l* > *0 then n* ∗ *d*^(*l*−*1*) *else 1*)
**using** *size-walks'* **unfolding** *walks-def n-def* **by** (*cases l*, *auto*)

**lemma** *walks-nonempty*:
*walks G l* ≠ {#}
**proof** −
**have** *size* (*walks G l*) > *0*
**unfolding** *size-walks* **using** *d-gt-0 n-gt-0* **by** *auto*
**thus** *walks G l* ≠ {#}
**by** *auto*
**qed**

**end**

**context** *regular-graph-tts*
**begin**

**lemma** *g-step-remains-orth*:
**assumes** *g-inner f* (λ-. *1*) = *0*
**shows** *g-inner* (*g-step f*) (λ-. *1*) = *0* (**is** *?L* = *?R*)
**proof** −
**have** *?L* = (*A* ∗*v* (χ *i*. *f* (*enum-verts i*))) · *1*
**unfolding** *g-inner-conv g-step-conv one-vec-def* **by** *simp*
**also have** ... = (χ *i*. *f* (*enum-verts i*)) · *1*
**by** (*intro markov-orth-inv markov*)
**also have** ... = *g-inner f* (λ-. *1*)
**unfolding** *g-inner-conv one-vec-def* **by** *simp*
**also have** ... = *0* **using** *assms* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *spec-bound*:
*spec-bound A* $\Lambda_a$
**proof** −
**have** *norm* (*A* ∗*v v*) ≤ $\Lambda_a$ ∗ *norm v* **if** *v* · *1* = (*0*::*real*) **for** *v*::*real*^'*n*
**unfolding** $\Lambda_e$-*eq*-Λ

```
    by (intro γ_a-real-bound that)
  thus ?thesis
    unfolding spec-bound-def using Λ-ge-0 by auto
qed
```

A spectral expansion rule that does not require orthogonality of the vector for the stationary distribution:

```
lemma expansionD3:
  |g-inner f (g-step f)| ≤ Λ_a ∗ g-norm f^2 + (1−Λ_a) ∗ g-inner f (λ-. 1)^2 / n (is ?L ≤ ?R)
proof −
  define v where v = (χ i. f (enum-verts i))
  define v1 :: real^'n where v1 = ((v · 1) / n) ∗_R 1
  define v2 :: real^'n where v2 = v − v1
  have v-eq: v = v1 + v2
    unfolding v2-def by simp

  have 0: A ∗v v1 = v1
    unfolding v1-def using markov-apply[OF markov]
    by (simp add:algebra-simps)
  have 1: v1 v∗ A = v1
    unfolding v1-def using markov-apply[OF markov]
    by (simp add:algebra-simps scaleR-vector-matrix-assoc)

  have v2 · 1 = v · 1 − v1 · 1
    unfolding v2-def by (simp add:algebra-simps)
  also have ... = v · 1  − v · 1 ∗ real CARD('n) / real n
    unfolding v1-def by (simp add:inner-1-1)
  also have ... = 0
    using verts-non-empty unfolding card n-def by simp
  finally have 4:v2 · 1 = 0 by simp
  hence 2: v1 · v2 = 0
    unfolding v1-def by (simp add:inner-commute)

  define f2 where f2 i = v2 $ (enum-verts-inv i) for i
  have f2-def: v2 = (χ i. f2 (enum-verts i))
    unfolding f2-def Rep-inverse by simp

  have 6: g-inner f2 (λ-. 1) = 0
    unfolding g-inner-conv f2-def[symmetric] one-vec-def[symmetric] 4 by simp

  have |v2 · (A ∗v v2)| = |g-inner f2 (g-step f2)|
    unfolding f2-def g-inner-conv g-step-conv by simp
  also have ... ≤ Λ_a ∗ (g-norm f2)²
    by (intro expansionD1 6)
  also have ... = Λ_a ∗ (norm v2)^2
    unfolding g-norm-conv f2-def by simp
  finally have 5:|v2 · (A ∗v v2)| ≤ Λ_a ∗ (norm v2)² by simp

  have 3: norm (1 :: real^'n)^2 = n
    unfolding power2-norm-eq-inner inner-1-1 card n-def by presburger

  have ?L = |v · (A ∗v v)|
    unfolding g-inner-conv g-step-conv v-def by simp
  also have ... = |v1 · (A ∗v v1) + v2 · (A ∗v v1) + v1 · (A ∗v v2) + v2 · (A ∗v v2)|
    unfolding v-eq by (simp add:algebra-simps)
  also have ... = |v1 · v1 + v2 · v1 + v1 · v2 + v2 · (A ∗v v2)|
    unfolding dot-lmul-matrix[where x=v1,symmetric] 0 1 by simp
  also have ... = |v1 · v1 + v2 · (A ∗v v2)|
```

**using** *2* **by** (*simp add*:*inner-commute*)
**also have** ... $\leq$ |*norm v1^2*| + |*v2* $\cdot$ (*A* $*v$ *v2*)|
**unfolding** *power2-norm-eq-inner* **by** (*intro abs-triangle-ineq*)
**also have** ... $\leq$ *norm v1^2* + $\Lambda_a$ $*$ *norm v2^2*
**by** (*intro add-mono 5*) *auto*
**also have** ... = $\Lambda_a$ $*$ (*norm v1^2* + *norm v2^2*) + (*1* − $\Lambda_a$) $*$ *norm v1^2*
**by** (*simp add*:*algebra-simps*)
**also have** ... = $\Lambda_a$ $*$ *norm v^2* + (*1* − $\Lambda_a$) $*$ *norm v1^2*
**unfolding** *v-eq pythagoras*[*OF 2*] **by** *simp*
**also have** ... = $\Lambda_a$ $*$ *norm v^2* + ((*1* − $\Lambda_a$)) $*$ ((*v* $\cdot$ *1*)^*2*$*$*n*)/*n^2*
**unfolding** *v1-def* **by** (*simp add*:*power-divide power-mult-distrib 3*)
**also have** ... = $\Lambda_a$ $*$ *norm v^2* + ((*1* − $\Lambda_a$)/*n*) $*$ (*v* $\cdot$ *1*)^*2*
**by** (*simp add*:*power2-eq-square*)
**also have** ... = *?R*
**unfolding** *g-norm-conv g-inner-conv v-def one-vec-def* **by** (*simp add*:*field-simps*)
**finally show** *?thesis* **by** *simp*
**qed**

**definition** *ind-mat* **where** *ind-mat S* = *diag* (*ind-vec* (*enum-verts* −‘ *S*))

**lemma** *walk-distr*:
*measure* (*pmf-of-multiset* (*walks G l*)) {$\omega$. ($\forall$ *i*<*l*. $\omega$ ! *i* $\in$ *S i*)} =
*foldl* ($\lambda$*x M*. *M* $*v$ *x*) *stat* (*intersperse A* (*map* ($\lambda$*i*. *ind-mat* (*S i*)) [*0*..<*l*]))$\cdot$*1*
(**is** *?L* = *?R*)
**proof** (*cases l* > *0*)
**case** *True*
**let** *?n* = *real n*
**let** *?d* = *real d*
**let** *?W* = {(*w*::'*a list*). *set w* $\subseteq$ *verts G* $\wedge$ *length w* = *l*}
**let** *?V* = {(*w*::'*n list*). *length w* = *l*}

**have** *a*: *set-mset* (*walks G l*) $\subseteq$ *?W*
**using** *set-walks* **by** *auto*
**have** *b*: *finite ?W*
**by** (*intro finite-lists-length-eq*) *auto*

**define** *lp* **where** *lp* = *l* − *1*

**define** *xs* **where** *xs* = *map* ($\lambda$*i*. *ind-mat* (*S i*)) [*0*..<*l*]
**have** *xs* $\neq$ [] **unfolding** *xs-def* **using** *True* **by** *simp*
**then obtain** *xh xt* **where** *xh-xt*: *xh#xt=xs* **by** (*cases xs*, *auto*)

**have** *length xs* = *l*
**unfolding** *xs-def* **by** *simp*
**hence** *len-xt*: *length xt* = *lp*
**using** *True* **unfolding** *xh-xt*[*symmetric*] *lp-def* **by** *simp*

**have** *xh* = *xs* ! *0*
**unfolding** *xh-xt*[*symmetric*] **by** *simp*
**also have** ... = *ind-mat* (*S 0*)
**using** *True* **unfolding** *xs-def* **by** *simp*
**finally have** *xh-eq*: *xh* = *ind-mat* (*S 0*)
**by** *simp*

**have** *inj-map-enum-verts*: *inj-on* (*map enum-verts*) *?V*
**using** *bij-betw-imp-inj-on*[*OF enum-verts*] *inj-on-subset*
**by** (*intro inj-on-mapI*) *auto*

**have** *card ?W = card (verts G)^l*
**by** (*intro card-lists-length-eq*) *simp*
**also have** ... = *card* {*w. set w ⊆ (UNIV :: 'n set) ∧ length w = l*}
**unfolding** *card*[*symmetric*] **by** (*intro card-lists-length-eq*[*symmetric*]) *simp*
**also have** ... = *card ?V*
**by** (*intro arg-cong*[**where** *f=card*]) *auto*
**also have** ... = *card* (*map enum-verts ‘ ?V*)
**by** (*intro card-image*[*symmetric*] *inj-map-enum-verts*)
**finally have** *card ?W = card* (*map enum-verts ‘ ?V*)
**by** *simp*
**hence** *map enum-verts ‘ ?V = ?W*
**using** *bij-betw-apply*[*OF enum-verts*]
**by** (*intro card-subset-eq b image-subsetI*) *auto*

**hence** *bij-map-enum-verts*: *bij-betw* (*map enum-verts*) *?V ?W*
**using** *inj-map-enum-verts* **unfolding** *bij-betw-def* **by** *auto*

**have** *?L = size* {# *w* ∈# *walks G l.* ∀ *i<l. w ! i* ∈ *S i* #} / (*?n* ∗ *?d*^(*l−1*))
**using** *True* **unfolding** *size-walks measure-pmf-of-multiset*[*OF walks-nonempty*] **by** *simp*
**also have** ... = (∑ *w*∈*?W. real* (*count* (*walks G l*) *w*) ∗ *of-bool* (∀ *i<l. w*!*i* ∈ *S i*))/(*?n*∗*?d*^(*l−1*))
**unfolding** *size-filter-mset-conv sum-mset-conv-2*[*OF a b*] **by** *simp*
**also have** ... = (∑ *w*∈*?W.* (∏ *i<l−1. real* (*count* (*edges G*) (*w*!*i*,*w*!(*i+1*)))) ∗
(∏ *i<l. of-bool* (*w*!*i* ∈ *S i*)))/(*?n*∗*?d*^(*l−1*))
**using** *True* **by** (*intro sum.cong arg-cong2*[**where** *f*=(/)]) (*auto simp add*: *count-walks*)
**also have** ... =
(∑ *w*∈*?W.* (∏ *i<l−1. real* (*count* (*edges G*) (*w*!*i*,*w*!(*i+1*)))/*?d*)∗(∏ *i<l. of-bool* (*w*!*i* ∈ *S i*)))/*?n*
**using** *True* **unfolding** *prod-dividef* **by** (*simp add*:*sum-divide-distrib algebra-simps*)
**also have** ... =
(∑ *w*∈*?V.* (∏ *i<l−1. count* (*edges G*) (*map enum-verts w*!*i*,*map enum-verts w*!(*i+1*)) / *?d*) ∗
(∏ *i<l. of-bool* (*map enum-verts w*!*i* ∈ *S i*)))/*?n*
**by** (*intro sum.reindex-bij-betw*[*symmetric*] *arg-cong2*[**where** *f*=(/)] *refl bij-map-enum-verts*)
**also have** ... =
(∑ *w*∈*?V.* (∏ *i<lp. A* $ *w*!(*i+1*) $ *w*!*i*) ∗ (∏ *i<Suc lp. of-bool*(*enum-verts* (*w*!*i*) ∈ *S i*)))/*?n*
**unfolding** *A-def lp-def* **using** *True* **by** *simp*
**also have** ... = (∑ *w*∈*?V.* (∏ *i<lp. A* $ *w*!(*i+1*) $ *w*!*i*) ∗
(∏ *i*∈*insert 0* (*Suc ‘* {..<*lp*}). *of-bool*(*enum-verts* (*w*!*i*) ∈ *S i*)))/*?n*
**using** *lessThan-Suc-eq-insert-0*
**by** (*intro sum.cong arg-cong2*[**where** *f*=(/)] *arg-cong2*[**where** *f*=(∗)] *prod.cong*) *auto*
**also have** ... = (∑ *w*∈*?V.* (∏ *i<lp. of-bool*(*enum-verts* (*w*!(*i+1*))∈*S*(*i+1*))∗ *A*$ *w*!(*i+1*) $ *w*!*i*)
∗ *of-bool*(*enum-verts*(*w*!*0*)∈*S 0*))/*?n*
**by** (*simp add*:*prod.reindex algebra-simps prod.distrib*)
**also have** ... =
(∑ *w*∈*?V.* (∏ *i<lp.* (*ind-mat* (*S* (*i+1*))∗∗*A*) $ *w*!(*i+1*) $ *w*!*i*) ∗ *of-bool*(*enum-verts* (*w*!*0*)∈*S 0*))/*?n*
**unfolding** *diag-def ind-vec-def matrix-matrix-mult-def ind-mat-def*
**by** (*intro sum.cong arg-cong2*[**where** *f*=(/)] *arg-cong2*[**where** *f*=(∗)] *prod.cong refl*)
(*simp add*:*if-distrib if-distribR sum.If-cases*)
**also have** ... =
(∑ *w*∈*?V.* (∏ *i<lp.* (*xs*!(*i+1*)∗∗*A*) $ *w*!(*i+1*) $ *w*!*i*) ∗ *of-bool*(*enum-verts* (*w*!*0*)∈*S 0*))/*?n*
**unfolding** *xs-def lp-def True*
**by** (*intro sum.cong arg-cong2*[**where** *f*=(/)] *arg-cong2*[**where** *f*=(∗)] *prod.cong refl*) *auto*
**also have** ... =
(∑ *w*∈*?V.* (∏ *i<lp.* (*xt ! i* ∗∗ *A*) $ *w*!(*i+1*) $ *w*!*i*) ∗ *of-bool*(*enum-verts* (*w*!*0*)∈*S 0*))/*?n*
**unfolding** *xh-xt*[*symmetric*] **by** *auto*
**also have** ... = (∑ *w*∈*?V.* (∏ *i<lp.* (*xt*!*i*∗∗*A*)$ *w*!(*i+1*) $ *w*!*i*)∗(*ind-mat*(*S 0*)∗*v stat*) $*w*!*0*)
**using** *n-def* **unfolding** *matrix-vector-mult-def diag-def stat-def ind-vec-def ind-mat-def card*
**by** (*simp add*:*sum.If-cases if-distrib if-distribR sum-divide-distrib*)

also have ... = $(\sum w\in ?V.\ (\prod i<lp.\ (xt\ !\ i\ **\ A)\ \$\ w!(i+1)\ \$\ w!i) * (xh *v\ stat)\ \$\ w\ !\ 0)$
  **unfolding** *xh-eq* **by** *simp*
also have ... = $foldl\ (\lambda x\ M.\ M *v\ x)\ (xh *v\ stat)\ (map\ (\lambda x.\ x\ **\ A)\ xt) \cdot 1$
  **using** *True* **unfolding** *foldl-matrix-mult-expand-2* **by** (*simp add*:*len-xt lp-def*)
also have ... = $foldl\ (\lambda x\ M.\ M *v\ (A *v\ x))\ (xh *v\ stat)\ xt \cdot 1$
  **by** (*simp add*: *matrix-vector-mul-assoc foldl-map*)
also have ... = $foldl\ (\lambda x\ M.\ M *v\ x)\ stat\ (intersperse\ A\ (xh\#xt)) \cdot 1$
  **by** (*subst foldl-intersperse-2*, *simp*)
also have ... = *?R* **unfolding** *xh-xt xs-def* **by** *simp*
finally show *?thesis* **by** *simp*
**next**
  **case** *False*
  **hence** $l = 0$ **by** *simp*
  **thus** *?thesis* **unfolding** *stat-def* **by** (*simp add*: *inner-1-1*)
**qed**

**lemma** *hitting-property*:
  **assumes** $S \subseteq verts\ G$
  **assumes** $I \subseteq \{..<l\}$
  **defines** $\mu \equiv real\ (card\ S)\ /\ card\ (verts\ G)$
  **shows** $measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ set\ (nths\ w\ I) \subseteq S\} \le (\mu+\Lambda_a*(1-\mu))\hat{}card\ I$
    (**is** $?L \le ?R$)
**proof** −
  **define** *T* **where** $T = (\lambda i.\ if\ i \in I\ then\ S\ else\ UNIV)$

  **have** *0*: $ind\text{-}mat\ UNIV = mat\ 1$
    **unfolding** *ind-mat-def diag-def ind-vec-def Finite-Cartesian-Product.mat-def* **by** *vector*

  **have** $\Lambda$-*range*: $\Lambda_a \in \{0..1\}$
    **using** $\Lambda$-*ge-0* $\Lambda$-*le-1* **by** *simp*

  **have** $S \subseteq range\ enum\text{-}verts$
    **using** *assms*(*1*) *enum-verts* **unfolding** *bij-betw-def* **by** *simp*
  **moreover have** *inj enum-verts*
    **using** *bij-betw-imp-inj-on*[*OF enum-verts*] **by** *simp*
  **ultimately have** $\mu$-*alt*: $\mu = real\ (card\ (enum\text{-}verts\ -`\ S))\ /\ CARD\ ('n)$
    **unfolding** $\mu$-*def card* **by** (*subst card-vimage-inj*) *auto*

  **have** $?L = measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ \forall\ i<l.\ w\ !\ i \in T\ i\}$
    **using** *walks-nonempty set-walks-3* **unfolding** *T-def set-nths*
    **by** (*intro measure-eq-AE AE-pmfI*) *auto*
  **also have** ... = $foldl\ (\lambda x\ M.\ M *v\ x)\ stat$
    $(intersperse\ A\ (map\ (\lambda i.\ (if\ i \in I\ then\ ind\text{-}mat\ S\ else\ mat\ 1))\ [0..<l])) \cdot 1$
    **unfolding** *walk-distr T-def* **by** (*simp add*:*if-distrib if-distribR 0 cong*:*if-cong*)
  **also have** ... $\le ?R$
    **unfolding** $\mu$-*alt ind-mat-def*
    **by** (*intro hitting-property-alg-2*[*OF* $\Lambda$-*range assms*(*2*) *spec-bound markov*])
  **finally show** *?thesis* **by** *simp*
**qed**

**lemma** *uniform-property*:
  **assumes** $i < l\ x \in verts\ G$
  **shows** $measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ w\ !\ i = x\} = 1/real\ (card\ (verts\ G))$
    (**is** $?L = ?R$)
**proof** −
  **obtain** *xi* **where** *xi-def*: *enum-verts xi = x*
    **using** *assms*(*2*) *bij-betw-imp-surj-on*[*OF enum-verts*] **by** *force*

**define** *T* **where** $T = (\lambda j.\ if\ j = i\ then\ \{x\}\ else\ UNIV)$

**have** $diag\ (ind\text{-}vec\ UNIV) = mat\ 1$
**unfolding** *diag-def ind-vec-def Finite-Cartesian-Product.mat-def* **by** *vector*
**moreover have** $enum\text{-}verts\ -`\ \{x\} = \{xi\}$
**using** *bij-betw-imp-inj-on*[*OF enum-verts*]
**unfolding** *vimage-def xi-def*[*symmetric*] **by** (*auto simp add*:*inj-on-def*)
**ultimately have** *0*: $ind\text{-}mat\ (T\ j) = (if\ j = i\ then\ diag\ (ind\text{-}vec\ \{xi\})\ else\ mat\ 1)$ **for** *j*
**unfolding** *T-def ind-mat-def* **by** (*cases* $j = i$, *auto*)

**have** $?L = measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ \forall j < l.\ w\ !\ j \in T\ j\}$
**unfolding** *T-def* **using** *assms*(*1*) **by** *simp*
**also have** ... $= foldl\ (\lambda x\ M.\ M *v\ x)\ stat\ (intersperse\ A\ (map\ (\lambda j.\ ind\text{-}mat\ (T\ j))\ [0..<l])) \cdot 1$
**unfolding** *walk-distr* **by** *simp*
**also have** ... $= 1/CARD('n)$
**unfolding** *0 uniform-property-alg*[*OF assms*(*1*) *markov*] **by** *simp*
**also have** ... = *?R*
**unfolding** *card* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**end**

**context** *regular-graph*
**begin**

**lemmas** *expansionD3* =
*regular-graph-tts.expansionD3*[*OF eg-tts-1*,
*internalize-sort 'n* :: *finite*, *OF - regular-graph-axioms*,
*unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**lemmas** *g-step-remains-orth* =
*regular-graph-tts.g-step-remains-orth*[*OF eg-tts-1*,
*internalize-sort 'n* :: *finite*, *OF - regular-graph-axioms*,
*unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**lemmas** *hitting-property* =
*regular-graph-tts.hitting-property*[*OF eg-tts-1*,
*internalize-sort 'n* :: *finite*, *OF - regular-graph-axioms*,
*unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**lemmas** *uniform-property-2* =
*regular-graph-tts.uniform-property*[*OF eg-tts-1*,
*internalize-sort 'n* :: *finite*, *OF - regular-graph-axioms*,
*unfolded remove-finite-premise*, *cancel-type-definition*, *OF verts-non-empty*]

**theorem** *uniform-property*:
**assumes** $i < l$
**shows** $map\text{-}pmf\ (\lambda w.\ w\ !\ i)\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l)) = pmf\text{-}of\text{-}set\ (verts\ G)$ (**is** $?L = ?R$)
**proof** (*rule pmf-eqI*)
**fix** $x :: 'a$
**have** *a*:$measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ w\ !\ i = x\} = 0$ (**is** $?L1 = ?R1$)
**if** $x \notin verts\ G$
**proof** −
**have** $?L1 \le measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{w.\ set\ w \subseteq verts\ G \wedge x \in set\ w\}$
**using** *walks-nonempty set-walks-3 assms*(*1*)
**by** (*intro pmf-mono*) *auto*
**also have** ... $\le measure\ (pmf\text{-}of\text{-}multiset\ (walks\ G\ l))\ \{\}$

```
      using that by (intro pmf-mono) auto
    also have ... = 0 by simp
    finally have ?L1 ≤ 0 by simp
    thus ?thesis using measure-le-0-iff by blast
  qed

  have pmf ?L x = measure (pmf-of-multiset (walks G l)) {w. w ! i = x}
    unfolding pmf-map by (simp add:vimage-def)
  also have ... = indicator (verts G) x/real (card (verts G))
    using uniform-property-2[OF assms(1)] a
    by (cases x ∈ verts G, auto)
  also have ... = pmf ?R x
    using verts-non-empty by (intro pmf-of-set[symmetric]) auto
  finally show pmf ?L x = pmf ?R x by simp
qed

lemma uniform-property-gen:
  fixes S :: 'a set
  assumes S ⊆ verts G i < l
  defines μ ≡ real (card S) / card (verts G)
  shows measure (pmf-of-multiset (walks G l)) {w. w ! i ∈ S} = μ (is ?L = ?R)
proof −

  have ?L = measure (map-pmf (λw. w ! i) (pmf-of-multiset (walks G l))) S
    unfolding measure-map-pmf by (simp add:vimage-def)
  also have ... = measure (pmf-of-set (verts G)) S
    unfolding uniform-property[OF assms(2)] by simp
  also have ... = ?R
    using verts-non-empty Int-absorb1[OF assms(1)]
    unfolding μ-def by (subst measure-pmf-of-set) auto
  finally show ?thesis by simp
qed

theorem kl-chernoff-property:
  assumes l > 0
  assumes S ⊆ verts G
  defines μ ≡ real (card S) / card (verts G)
  assumes γ ≤ 1 μ + Λ_a ∗ (1−μ) ∈ {0<..γ}
  shows measure (pmf-of-multiset (walks G l)) {w. real (card {i ∈ {..<l}. w ! i ∈ S}) ≥ γ∗l}
    ≤ exp (− real l ∗ KL-div γ (μ+Λ_a∗(1−μ))) (is ?L ≤ ?R)
proof −
  let ?δ = (∑i<l. μ+Λ_a∗(1−μ))/l

  have a: measure (pmf-of-multiset (walks G l)) {w. ∀i∈T. w ! i ∈ S} ≤ (μ + Λ_a∗(1−μ)) ^ card
T
    (is ?L1 ≤ ?R1) if T ⊆ {..<l} for T
  proof −
    have ?L1 = measure (pmf-of-multiset (walks G l)) {w. set (nths w T) ⊆ S}
      unfolding set-nths setcompr-eq-image using that set-walks-3 walks-nonempty
      by (intro measure-eq-AE AE-pmfI) (auto simp add:image-subset-iff)
    also have ... ≤ ?R1
      unfolding μ-def by (intro hitting-property[OF assms(2) that])
    finally show ?thesis by simp
  qed

  have ?L ≤ exp ( − real l ∗ KL-div γ ?δ)
    using assms(1,4,5) a by (intro impagliazzo-kabanets-pmf) simp-all
  also have ... = ?R by simp
```

**finally show** *?thesis* **by** *simp*
**qed**

**end**

**unbundle** *no intro-cong-syntax*

**end**

# 10 Graph Powers

**theory** *Expander-Graphs-Power-Construction*
**imports**
*Expander-Graphs-Walks*
*Graph-Theory.Arc-Walk*
**begin**

**unbundle** *intro-cong-syntax*

**fun** *is-arc-walk* :: ('a, 'b) *pre-digraph* ⇒ 'a ⇒ 'b *list* ⇒ *bool*
**where**
*is-arc-walk G - [] = True* |
*is-arc-walk G y (x#xs) = (is-arc-walk G (head G x) xs ∧ tail G x = y ∧ x ∈ arcs G)*

**definition** *arc-walk-head* :: ('a, 'b) *pre-digraph* ⇒ ('a × 'b *list*) ⇒ 'a
**where**
*arc-walk-head G x = (if snd x = [] then fst x else head G (last (snd x)))*

**lemma** *is-arc-walk-snoc*:
*is-arc-walk G y (xs@[x]) ⟷ is-arc-walk G y xs ∧ x ∈ out-arcs G (arc-walk-head G (y,xs))*
**by** (*induction xs arbitrary*: *y*, *simp-all add*:*ac-simps arc-walk-head-def*)

**lemma** *is-arc-walk-set*:
**assumes** *is-arc-walk G u w*
**shows** *set w ⊆ arcs G*
**using** *assms* **by** (*induction w arbitrary*: *u*, *auto*)

**lemma** (**in** *wf-digraph*) *awalk-is-arc-walk*:
**assumes** *u ∈ verts G*
**shows** *is-arc-walk G u w ⟷ awalk u w (awlast u w)*
**using** *assms* **unfolding** *awalk-def* **by** (*induction w arbitrary*: *u*, *auto*)

**definition** *arc-walks* :: ('a, 'b) *pre-digraph* ⇒ *nat* ⇒ ('a × 'b *list*) *set*
**where**
*arc-walks G l = {(u,w). u ∈ verts G ∧ is-arc-walk G u w ∧ length w = l}*

**lemma** *arc-walks-len*:
**assumes** *x ∈ arc-walks G l*
**shows** *length (snd x) = l*
**using** *assms* **unfolding** *arc-walks-def* **by** *auto*

**lemma** (**in** *wf-digraph*) *awhd-of-arc-walk*:
**assumes** *w ∈ arc-walks G l*
**shows** *awhd (fst w) (snd w) = fst w*
**using** *assms* **unfolding** *arc-walks-def awalk-verts-def*
**by** (*cases snd w*, *auto*)

**lemma** (**in** *wf-digraph*) *awlast-of-arc-walk*:
  **assumes** $w \in$ *arc-walks G l*
  **shows** *awlast* (*fst w*) (*snd w*) = *arc-walk-head G w*
  **unfolding** *awalk-verts-conv arc-walk-head-def* **by** *simp*

**lemma** (**in** *wf-digraph*) *arc-walk-head-wellformed*:
  **assumes** $w \in$ *arc-walks G l*
  **shows** *arc-walk-head G w* $\in$ *verts G*
**proof** (*cases snd w* = [])
  **case** *True*
  **then show** *?thesis*
    **using** *assms* **unfolding** *arc-walks-def arc-walk-head-def* **by** *auto*
**next**
  **case** *False*
  **have** *0*:*is-arc-walk G* (*fst w*) (*snd w*) **using** *assms* **unfolding** *arc-walks-def* **by** *auto*
  **have** *last* (*snd w*) $\in$ *set* (*snd w*)
    **using** *False last-in-set* **by** *auto*
  **also have** ... $\subseteq$ *arcs G*
    **by** (*intro is-arc-walk-set*[*OF 0*])
  **finally have** *last* (*snd w*) $\in$ *arcs G* **by** *simp*
  **thus** *?thesis* **unfolding** *arc-walk-head-def* **using** *False* **by** *simp*
**qed**

**lemma** (**in** *wf-digraph*) *arc-walk-tail-wellformed*:
  **assumes** $w \in$ *arc-walks G l*
  **shows** *fst w* $\in$ *verts G*
  **using** *assms* **unfolding** *arc-walks-def* **by** *auto*

**lemma** (**in** *fin-digraph*) *arc-walks-fin*:
  *finite* (*arc-walks G l*)
**proof** −
  **have** *0*:*finite* (*verts G* $\times$ {*w. set w* $\subseteq$ *arcs G* $\wedge$ *length w* = *l*})
    **by** (*intro finite-cartesian-product finite-lists-length-eq*) *auto*
  **show** *finite* (*arc-walks G l*)
    **unfolding** *arc-walks-def* **using** *is-arc-walk-set*[**where** *G*=*G*]
    **by** (*intro finite-subset*[*OF - 0*] *subsetI*) *auto*
**qed**

**lemma** (**in** *wf-digraph*) *awalk-verts-unfold*:
  **assumes** $w \in$ *arc-walks G l*
  **shows** *awalk-verts* (*fst w*) (*snd w*) = *fst w#map* (*head G*) (*snd w*) (**is** *?L* = *?R*)
**proof** −
  **obtain** *u v* **where** *w-def*: *w* = (*u*,*v*) **by** *fastforce*

  **have** *awalk u v* (*awlast u v*)
    **using** *assms* **unfolding** *w-def arc-walks-def*
    **by** (*intro iffD1*[*OF awalk-is-arc-walk*]) *auto*
  **hence** *cas*: *cas u v* (*awlast u v*)
    **unfolding** *awalk-def* **by** *simp*

  **have** *0*: *tail G* (*hd v*) = *u* **if** *v* $\neq$ []
    **using** *cas that* **by** (*cases v*) *auto*

  **have** *?L* = *awalk-verts u v*
    **unfolding** *w-def* **by** *simp*
  **also have** ... = (*if v* = [] *then* [*u*] *else tail G* (*hd v*) # *map* (*head G*) *v*)
    **by** (*intro awalk-verts-conv'*[*OF cas*])
  **also have** ... = *u# map* (*head G*) *v*

**using** *0* **by** *simp*
**also have** ... = *?R*
**unfolding** *w-def* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** (**in** *fin-digraph*) *arc-walks-map-walks′*:
*walks′ G l = image-mset (case-prod awalk-verts) (mset-set (arc-walks G l))*
**proof** (*induction l*)
**case** *0*
**let** *?g = λx. fst x#map (head G) (snd x)*

**have** *walks′ G 0 = {#[x]. x ∈# mset-set (verts G)#}*
**by** *simp*
**also have** ... = *image-mset ?g (image-mset (λx. (x,[])) (mset-set (verts G)))*
**unfolding** *image-mset.compositionality* **by** (*simp add:comp-def*)
**also have** ... = *image-mset ?g (mset-set ((λx. (x,[])) ‘ verts G))*
**by** (*intro arg-cong2*[**where** *f=image-mset*] *image-mset-mset-set inj-onI*) *auto*
**also have** ... = *image-mset ?g (mset-set ({(u, w). u ∈ verts G ∧ w = []}))*
**by** (*intro-cong* [$\sigma_2$ *image-mset*]) *auto*
**also have** ... = *image-mset ?g (mset-set (arc-walks G 0))*
**unfolding** *arc-walks-def* **by** (*intro-cong* [$\sigma_2$ *image-mset*,$\sigma_1$ *mset-set*]) *auto*
**also have** ... = *image-mset (case-prod awalk-verts) (mset-set (arc-walks G 0))*
**using** *arc-walks-fin* **by** (*intro image-mset-cong*) (*simp add:case-prod-beta awalk-verts-unfold*)
**finally show** *?case* **by** *simp*
**next**
**case** (*Suc l*)
**let** *?f = λ(u,w) a. (u,w@[a])*
**let** *?g = λx. fst x#map (head G) (snd x)*

**have** *arc-walks G (l+1) = case-prod ?f ‘ {(x,y). ?f x y ∈ arc-walks G (l+1)}*
**using** *arc-walks-len*[**where** *G=G* **and** *l=Suc l, THEN iffD1[OF length-Suc-conv-rev]*]
**by** *force*
**also have** ... = *case-prod ?f ‘ {(x,y). x ∈ arc-walks G l ∧ y ∈ out-arcs G (arc-walk-head G x)}*
**unfolding** *arc-walks-def* **using** *is-arc-walk-snoc*[**where** *G=G*]
**by** (*intro-cong* [$\sigma_2$ *image*]) *auto*
**also have** ... = (⋃*w ∈ arc-walks G l. ?f w ‘ out-arcs G (arc-walk-head G w)*)
**by** (*auto simp add:image-iff*)
**finally have** *0:arc-walks G (l+1) = (⋃w ∈ arc-walks G l. ?f w ‘ out-arcs G (arc-walk-head G w))*
**by** *simp*

**have** *mset-set (arc-walks G (l+1)) = concat-mset (image-mset (mset-set ∘ (λw. ?f w ‘ out-arcs G (arc-walk-head G w))) (mset-set (arc-walks G l)))*
**unfolding** *0* **by** (*intro concat-disjoint-union-mset arc-walks-fin finite-imageI*) *auto*
**also have** ... = *concat-mset {# mset-set (?f x ‘ out-arcs G (arc-walk-head G x)). x∈#mset-set(arc-walks G l)#}*
**by** (*simp add:comp-def case-prod-beta*)
**also have** ... = *concat-mset {# {# ?f x y. y ∈# mset-set (out-arcs G (arc-walk-head G x))#}. x ∈# mset-set (arc-walks G l)#}*
**by** (*intro-cong* [$\sigma_1$ *concat-mset*] *more:image-mset-cong image-mset-mset-set[symmetric] inj-onI*) *auto*
**finally have** *1:mset-set (arc-walks G (l+1)) = concat-mset {# {# ?f x y. y ∈# mset-set (out-arcs G (arc-walk-head G x))#}. x ∈# mset-set (arc-walks G l)#}*
**by** *simp*

**have** *walks′ G (l+1)=concat-mset {#{#w @ [z]. z∈# vertices-from G (last w)#}. w ∈# walks′*

*G l*#}
    **by** *simp*
  **also have** ... = *concat-mset* {#
      {#*awalk-verts* (*fst x*) (*snd x*) @ [*z*]. *z* ∈# *vertices-from G* (*awlast* (*fst x*) (*snd x*))#}.
      *x* ∈# *mset-set* (*arc-walks G l*)#}
    **unfolding** *Suc* **by** (*simp add*:*image-mset.compositionality comp-def case-prod-beta*)
  **also have** ... = *concat-mset* {#
      {#*?g x* @ [*z*]. *z* ∈# *vertices-from G* (*awlast* (*fst x*) (*snd x*))#}.
      *x* ∈# *mset-set* (*arc-walks G l*)#}
    **using** *arc-walks-fin*
    **by** (*intro-cong* [$\sigma_1$ *concat-mset*] *more*:*image-mset-cong*) (*auto simp*: *awalk-verts-unfold*)
  **also have** ... = *concat-mset* {# {#*?g x* @ [*z*]. *z* ∈# *vertices-from G* (*arc-walk-head G x*)#}.
      *x* ∈# *mset-set* (*arc-walks G l*)#}
    **using** *arc-walks-fin awlast-of-arc-walk*
    **by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*] *more*: *image-mset-cong*) *auto*
  **also have** ... = (*concat-mset* {# {# *?g* (*fst x*, *snd x*@[*y*]).
    *y* ∈# *mset-set* (*out-arcs G* (*arc-walk-head G x*))#}. *x* ∈# *mset-set* (*arc-walks G l*)#})
    **unfolding** *verts-from-alt* **by** (*simp add*:*image-mset.compositionality comp-def*)
  **also have** ... = *image-mset ?g* (*concat-mset* {# {# *?f x y*.
    *y* ∈# *mset-set* (*out-arcs G* (*arc-walk-head G x*))#}. *x* ∈# *mset-set* (*arc-walks G l*)#})
    **unfolding** *image-concat-mset*
    **by** (*auto simp add*:*comp-def case-prod-beta image-mset.compositionality*)
  **also have** ... = *image-mset ?g* (*mset-set* (*arc-walks G* (*l*+*1*)))
    **unfolding** *1* **by** *simp*
  **also have** ... = *image-mset* (*case-prod awalk-verts*) (*mset-set* (*arc-walks G* (*l*+*1*)))
    **using** *arc-walks-fin* **by** (*intro image-mset-cong*) (*simp add*:*case-prod-beta awalk-verts-unfold*)
  **finally show** *?case* **by** *simp*
**qed**

**lemma** (**in** *fin-digraph*) *arc-walks-map-walks*:
  *walks G* (*l*+*1*) = *image-mset* (*case-prod awalk-verts*) (*mset-set* (*arc-walks G l*))
  **using** *arc-walks-map-walks′* **unfolding** *walks-def* **by** *simp*

**lemma** (**in** *wf-digraph*)
  **assumes** *awalk u a v  length a* = *l l* > *0*
  **shows** *awalk-ends*: *tail G* (*hd a*) = *u head G* (*last a*) = *v*
**proof** −
  **have** *0*:*cas u a v*
    **using** *assms* **unfolding** *awalk-def* **by** *simp*
  **have** *1*: *a* ≠ [] **using** *assms*(*2*,*3*) **by** *auto*

  **show** *tail G* (*hd a*) = *u*
    **using** *0* **unfolding** *cas-simp*[*OF 1*] **by** *auto*

  **show** *head G* (*last a*) = *v*
    **using** *1 0* **by** (*induction a arbitrary*:*u rule*:*list-nonempty-induct*) *auto*
**qed**

**definition** *graph-power* :: (*′a*, *′b*) *pre-digraph* ⇒ *nat* ⇒ (*′a*, (*′a* × *′b list*)) *pre-digraph*
  **where** *graph-power G l* =
    ⦇ *verts* = *verts G*, *arcs* = *arc-walks G l*, *tail* = *fst*, *head* = *arc-walk-head G* ⦈

**lemma** (**in** *wf-digraph*) *graph-power-wf*:
  *wf-digraph* (*graph-power G l*)
**proof** −
  **have** *tail* (*graph-power G l*) *a* ∈ *verts* (*graph-power G l*)
       *head* (*graph-power G l*) *a* ∈ *verts* (*graph-power G l*)
       **if** *a* ∈ *arcs* (*graph-power G l*) **for** *a*

```
    using that arc-walk-head-wellformed arc-walk-tail-wellformed
    unfolding graph-power-def by simp-all
  thus ?thesis
    unfolding wf-digraph-def by auto
qed

lemma (in fin-digraph) graph-power-fin:
  fin-digraph (graph-power G l)
proof –
  interpret H:wf-digraph graph-power G l
    using graph-power-wf by auto

  have finite (arcs (graph-power G l))
    using arc-walks-fin
    unfolding graph-power-def by simp

  moreover have finite (verts (graph-power G l))
    unfolding graph-power-def by simp
  ultimately show ?thesis
    by unfold-locales auto
qed

lemma (in fin-digraph) graph-power-count-edges:
  fixes l v w
  defines S ≡ {x. length x=l+1∧set x⊆verts G∧hd x=v∧last x=w}
  shows count (edges (graph-power G l)) (v,w) = (∑ x ∈ S.(∏ i<l. count(edges G)(x!i,x!(i+1))))
    (is ?L = ?R)
proof –
  interpret H:fin-digraph graph-power G l
    using graph-power-fin by auto

  have 0:finite {x. set x ⊆ verts G ∧ length x = l+1}
    by (intro finite-lists-length-eq) auto
  have fin-S: finite S
    unfolding S-def by (intro finite-subset[OF - 0]) auto

  have ?L = size {#x ∈# mset-set (arc-walks G l). fst x = v ∧ arc-walk-head G x = w#}
    unfolding graph-power-def edges-def arc-to-ends-def
    by (simp add:count-mset-exp image-mset-filter-mset-swap[symmetric])
  also have ... = size
    {#x ∈# mset-set (arc-walks G l). awhd (fst x) (snd x) = v ∧ awlast (fst x) (snd x) = w#}
    using awlast-of-arc-walk awhd-of-arc-walk arc-walks-fin
    by (intro arg-cong[where f=size] filter-mset-cong refl) simp
  also have ... = size {#x ∈# walks G (l+1). hd x = v ∧ last x = w#}
    unfolding arc-walks-map-walks
    by (simp add:image-mset-filter-mset-swap[symmetric] case-prod-beta)
  also have ...=size{#x∈# walks G (l+1).x ∈ S#}
    unfolding S-def using set-walks-3
    by (intro arg-cong[where f=size] filter-mset-cong refl) auto
  also have ...=sum (count (walks G (l+1))) S
    by (intro sum-count-2[symmetric] fin-S)
  also have ...=(∑ x∈S.(∏ i<l+1−1. count (edges G) (x!i,x!(i+1))))
    unfolding S-def
    by (intro sum.cong refl count-walks) auto
  also have ...=?R
    by simp
  finally show ?thesis by simp
qed
```

```
lemma (in fin-digraph) graph-power-sym-aux:
  assumes symmetric-multi-graph G
  assumes v ∈ verts (graph-power G l)  w ∈ verts (graph-power G l)
  shows card (arcs-betw (graph-power G l) v w) = card (arcs-betw (graph-power G l) w v)
    (is ?L = ?R)
proof −
  interpret H:fin-digraph graph-power G l
    using graph-power-fin by auto

  define S where S v w = {x. length x=l+1 ∧ set x ⊆ verts G ∧ hd x = v ∧ last x = w} for v w

  have 0: bij-betw rev (S w v) (S v w)
    unfolding S-def by (intro bij-betwI[where g=rev]) (auto simp add:hd-rev last-rev)

  have 1: bij-betw ((−) (l − 1)) {..<l} {..<l}
    by (intro bij-betwI[where g=λx. (l−1−x)]) auto

  have ?L = count (edges (graph-power G l)) (v, w)
    unfolding H.count-edges by simp
  also have ... = (∑x ∈ S v w. (∏i<l. count (edges G) (x!i,x!(i+1))))
    unfolding S-def graph-power-count-edges by simp
  also have ... = (∑x ∈ S w v. (∏i<l. count (edges G) (rev x!i,rev x!(i+1))))
    by (intro sum.reindex-bij-betw[symmetric] 0)
  also have ... = (∑x ∈ S w v. (∏i<l. count (edges G) (x!((l−1−i)+1),x!(l−1−i))))
    unfolding S-def by (intro sum.cong refl prod.cong) (simp-all add: rev-nth Suc-diff-Suc)
  also have ... = (∑x ∈ S w v. (∏i<l. count (edges G) (x!(i+1),x!i)))
    by (intro sum.cong prod.reindex-bij-betw refl 1)
  also have ... = (∑x ∈ S w v. (∏i<l. count (edges G) (x!i,x!(i+1))))
    by (intro sum.cong prod.cong count-edges-sym[OF assms(1)] refl)
  also have ... = count (edges (graph-power G l)) (w, v)
    unfolding S-def graph-power-count-edges by simp
  also have ... = ?R
    unfolding H.count-edges by simp
  finally show ?thesis by simp
qed

lemma (in fin-digraph) graph-power-sym:
  assumes symmetric-multi-graph G
  shows symmetric-multi-graph (graph-power G l)
proof −
  interpret H:fin-digraph graph-power G l
    using graph-power-fin by auto

  show ?thesis
    using graph-power-sym-aux[OF assms]
    unfolding symmetric-multi-graph-def by auto
qed

lemma (in fin-digraph) graph-power-out-degree':
  assumes reg: ⋀v. v ∈ verts G ⟹ out-degree G v = d
  assumes v ∈ verts (graph-power G l)
  shows out-degree (graph-power G l) v = d ^ l  (is ?L = ?R)
proof −
  interpret H:fin-digraph graph-power G l
    using graph-power-fin by auto

  have v-vert: v ∈ verts G
```

**using** *assms* **unfolding** *graph-power-def* **by** *simp*

**have** *?L* = *size* (*vertices-from* (*graph-power* *G* *l*) *v*)
**unfolding** *out-degree-def* *H.verts-from-alt* **by** *simp*
**also have** ... = *size* ({# *e* ∈# *edges* (*graph-power* *G* *l*). *fst* *e* = *v* #})
**unfolding** *vertices-from-def* **by** *simp*
**also have** ... = *size* {#*w* ∈# *mset-set* (*arc-walks* *G* *l*). *fst* *w* = *v*#}
**unfolding** *graph-power-def* *edges-def* *arc-to-ends-def*
**by** (*simp* *add*:*count-mset-exp* *image-mset-filter-mset-swap*[*symmetric*])
**also have** ... = *size* {#*w* ∈# *mset-set* (*arc-walks* *G* *l*). *awhd* (*fst* *w*) (*snd* *w*) = *v*#}
**using** *awlast-of-arc-walk* *awhd-of-arc-walk* *arc-walks-fin*
**by** (*intro* *arg-cong*[**where** *f*=*size*] *filter-mset-cong* *refl*) *simp*
**also have** ... = *size* {#*x* ∈# *walks'* *G* *l*. *hd* *x* = *v* #}
**unfolding** *arc-walks-map-walks'*
**by** (*simp* *add*:*image-mset-filter-mset-swap*[*symmetric*] *case-prod-beta*)
**also have** ... = *d*^*l*
**proof** (*induction* *l*)
**case** *0*
**have** *size* {#*x* ∈# *walks'* *G* *0*. *hd* *x* = *v*#} = *card* {*x*. *x* = *v* ∧ *x* ∈ *verts* *G*}
**by** (*simp* *add*:*image-mset-filter-mset-swap*[*symmetric*])
**also have** ... = *card* {*v*}
**using** *v-vert* **by** (*intro* *arg-cong*[**where** *f*=*card*]) *auto*
**also have** ... = *d*^*0* **by** *simp*
**finally show** *?case* **by** *simp*
**next**
**case** (*Suc* *l*)
**have** *size* {#*x* ∈# *walks'* *G* (*Suc* *l*). *hd* *x* = *v*#} =
(∑*x*∈#*walks'* *G* *l*. *size* {#*y* ∈# *vertices-from* *G* (*last* *x*). *hd* (*x* @ [*y*]) = *v*#})
**by** (*simp* *add*:*size-concat-mset* *image-mset-filter-mset-swap*[*symmetric*]
*filter-concat-mset* *image-mset.compositionality* *comp-def*)
**also have** ... = (∑*x*∈#*walks'* *G* *l*. *size* {#*y* ∈# *vertices-from* *G* (*last* *x*). *hd* *x* = *v*#})
**using** *set-walks-2*
**by** (*intro-cong* [$\sigma_1$ *sum-mset*, $\sigma_1$ *size*] *more*:*image-mset-cong* *filter-mset-cong*) *auto*
**also have** ... = (∑*x*∈#*walks'* *G* *l*. (*if* *hd* *x* = *v* *then* *out-degree* *G* (*last* *x*) *else* *0*))
**unfolding** *verts-from-alt* *out-degree-def*
**by** (*simp* *add*:*filter-mset-const* *if-distribR* *if-distrib* *cong*:*if-cong*)
**also have** ... = (∑*x*∈#*walks'* *G* *l*. *d* ∗ *of-bool* (*hd* *x* = *v*))
**using** *set-walks-2*[**where** *l*=*l*] *last-in-set*
**by** (*intro* *arg-cong*[**where** *f*=*sum-mset*] *image-mset-cong*) (*auto* *intro*!:*reg*)
**also have** ... = *d* ∗ (∑*x*∈#*walks'* *G* *l*. *of-bool* (*hd* *x* = *v*))
**by** (*simp* *add*:*sum-mset-distrib-left* *image-mset.compositionality* *comp-def*)
**also have** ... = *d* ∗ (*size* {#*x* ∈# *walks'* *G* *l*. *hd* *x* = *v*#})
**by** (*simp* *add*:*size-filter-mset-conv*)
**also have** ... = *d* ∗ *d* ^ *l*
**using** *Suc* **by** *simp*
**also have** ... = *d*^*Suc* *l*
**by** *simp*
**finally show** *?case* **by** *simp*
**qed**

**finally show** *?thesis* **by** *simp*
**qed**

**lemma** (**in** *regular-graph*) *graph-power-out-degree*:
**assumes** *v* ∈ *verts* (*graph-power* *G* *l*)
**shows** *out-degree* (*graph-power* *G* *l*) *v* = *d* ^ *l* (**is** *?L* = *?R*)
**by** (*intro* *graph-power-out-degree'* *assms* *reg*) *auto*

```
lemma (in regular-graph) graph-power-regular:
  regular-graph (graph-power G l)
proof −
  interpret H:fin-digraph graph-power G l
    using graph-power-fin by auto

  have verts (graph-power G l) ≠ {}
    using verts-non-empty unfolding graph-power-def by simp

  moreover have 0 < d^l
    using d-gt-0 by simp

  ultimately show ?thesis
    using graph-power-out-degree
    by (intro regular-graphI[where d=d^l] graph-power-sym sym)
qed

lemma (in regular-graph) graph-power-degree:
  regular-graph.d (graph-power G l) = d^l (is ?L = ?R)
proof −
  interpret H:regular-graph graph-power G l
    using graph-power-regular by auto
  obtain v where v-set: v ∈ verts (graph-power G l)
    using H.verts-non-empty by auto
  hence ?L = out-degree (graph-power G l) v
    using v-set H.reg by auto
  also have ... =?R
    by (intro graph-power-out-degree[OF v-set])
  finally show ?thesis by simp
qed

lemma (in regular-graph) graph-power-step:
  assumes x ∈ verts G
  shows regular-graph.g-step (graph-power G l) f x = (g-step^^l) f x
  using assms
proof (induction l arbitrary: x)
  case 0
  let ?H = graph-power G 0
  interpret H:regular-graph ?H
    using graph-power-regular by auto
  have regular-graph.g-step (graph-power G 0) f x = H.g-step f x
    by simp
  have H.g-step f x = (∑x∈in-arcs ?H x. f (tail ?H x))
    unfolding H.g-step-def graph-power-degree by simp
  also have ... = (∑v∈{e ∈ arc-walks G 0. arc-walk-head G e = x}. f (fst v))
    unfolding in-arcs-def graph-power-def by (simp add:case-prod-beta)
  also have ... = (∑v∈{x}. f v)
    unfolding arc-walks-def using 0
    by (intro sum.reindex-bij-betw bij-betwI[where g=(λx. (x,[]))])
      (auto simp add:arc-walk-head-def)
  also have ... = f x
    by simp
  also have ... = (g-step^^0) f x
    by simp
  finally show ?case by simp
next
  case (Suc l)
  let ?H = graph-power G l
```

**interpret** *H*:*regular-graph ?H*
**using** *graph-power-regular* **by** *auto*
**let** *?HS* = *graph-power G* (*l*+*1*)
**interpret** *HS*:*regular-graph ?HS*
**using** *graph-power-regular* **by** *auto*

**let** *?bij* = (λ(*x*,(*y1*,*y2*)). (*y1*,*y2*@[*x*]))
**let** *?bijr* = (λ(*y1*,*y2*). (*last y2*, (*y1*,*butlast y2*)))

**define** *S* **where** *S* = {*y*. *fst y* ∈ *in-arcs G x* ∧ *snd y* ∈ *in-arcs ?H* (*tail G* (*fst y*))}

**have** *S* = {(*u*,*v*). *u* ∈ *arcs G* ∧ *head G u* = *x* ∧ *v* ∈ *arc-walks G l* ∧ *arc-walk-head G v* = *tail G u*}
**unfolding** *S-def graph-power-def in-arcs-def* **by** *auto*
**also have** ... = {(*u*,*v*). (*fst v*,*snd v*@[*u*]) ∈ *arc-walks G* (*l*+*1*) ∧ *arc-walk-head G* (*fst v*,*snd v*@[*u*]) = *x*}
**unfolding** *arc-walks-def* **by** (*intro iffD2*[*OF set-eq-iff*] *allI*)
(*auto simp add*: *is-arc-walk-snoc case-prod-beta arc-walk-head-def*)
**also have** ... = {(*u*,*v*). (*fst v*,*snd v*@[*u*]) ∈ *in-arcs ?HS x*}
**unfolding** *in-arcs-def graph-power-def* **by** *auto*
**finally have** *S-alt*: *S* = {(*u*,*v*). (*fst v*,*snd v*@[*u*]) ∈ *in-arcs ?HS x*} **by** *simp*

**have** *len-in-arcs*: *a* ∈ *in-arcs ?HS x* ⟹ *snd a* ≠ [] **for** *a*
**unfolding** *in-arcs-def graph-power-def arc-walks-def* **by** *auto*

**have** *0*:*bij-betw ?bij S* (*in-arcs ?HS x*)
**unfolding** *S-alt* **using** *len-in-arcs*
**by** (*intro bij-betwI*[**where** *g*=*?bijr*]) *auto*

**have** *HS*.*g-step f x* = (∑ *y*∈*in-arcs ?HS x*. *f* (*tail ?HS y*)/ *d*^(*l*+*1*))
**unfolding** *HS*.*g-step-def graph-power-degree* **by** *simp*
**also have** ... = (∑ *y*∈*in-arcs ?HS x*. *f* (*fst y*)/ *d*^(*l*+*1*))
**unfolding** *graph-power-def* **by** *simp*
**also have** ... = (∑ *y* ∈ *S*. *f* (*fst* (*?bij y*))/ *d*^(*l*+*1*))
**by** (*intro sum.reindex-bij-betw*[*symmetric*] *0*)
**also have** ... = (∑ *y* ∈ *S*. *f* (*fst* (*snd y*))/ *d*^(*l*+*1*))
**by** (*intro-cong* [$\sigma_2$ (/),$\sigma_1$ *f*] *more*: *sum.cong*) (*simp add*:*case-prod-beta*)
**also have** ...=(∑ *y*∈(⋃ *a*∈*in-arcs G x*. (*Pair a*)‘*in-arcs ?H* (*tail G a*)). *f* (*fst* (*snd y*))/ *d*^(*l*+*1*))
**unfolding** *S-def* **by** (*intro sum.cong*) *auto*
**also have** ...=(∑ *a*∈*in-arcs G x*. (∑ *y*∈(*Pair a*)‘*in-arcs ?H* (*tail G a*). *f* (*fst* (*snd y*))/ *d*^(*l*+*1*)))
**by** (*intro sum.UNION-disjoint*) *auto*
**also have** ... = (∑ *a* ∈ *in-arcs G x*. (∑ *b* ∈ *in-arcs ?H* (*tail G a*). *f* (*fst b*) / *d*^(*l*+*1*)))
**by** (*intro sum.cong sum.reindex-bij-betw*) (*auto simp add*:*bij-betw-def inj-on-def image-iff*)
**also have** ... = (∑ *a* ∈ *in-arcs G x*. (∑ *b* ∈ *in-arcs ?H* (*tail G a*). *f* (*tail ?H b*) / *d*^*l*)/*d*)
**unfolding** *graph-power-def*
**by** (*simp add*:*sum-divide-distrib algebra-simps*)
**also have** ... = (∑ *a* ∈ *in-arcs G x*. *H*.*g-step f* (*tail G a*)/ *d*)
**unfolding** *H*.*g-step-def graph-power-degree* **by** *simp*
**also have** ... = (∑ *a* ∈ *in-arcs G x*. (*g-step*^^*l*) *f* (*tail G a*)/ *d*)
**by** (*intro sum.cong refl arg-cong2*[**where** *f*=(/)] *Suc*) *auto*
**also have** ... = *g-step* ((*g-step*^^*l*) *f*) *x*
**unfolding** *g-step-def* **by** *simp*
**also have** ... = (*g-step*^^(*l*+*1*)) *f x*
**by** *simp*
**finally show** *?case* **by** *simp*
**qed**

**lemma** (**in** *regular-graph*) *graph-power-expansion*:

```
  regular-graph.Λa (graph-power G l) ≤ Λa^l
proof -
  interpret H:regular-graph graph-power G l
    using graph-power-regular by auto

  have |H.g-inner f (H.g-step f)| ≤ Λa ^ l * (H.g-norm f)^2 (is ?L ≤ ?R)
    if H.g-inner f (λ-. 1) = 0  for f
  proof -
    have g-inner f (λ-. 1) = H.g-inner f (λ-.1)
      unfolding g-inner-def H.g-inner-def
      by (intro sum.cong) (auto simp add:graph-power-def)
    also have ... = 0 using that by simp
    finally have 1:g-inner f (λ-. 1) = 0 by simp

    have 2: g-inner ((g-step^^l) f) (λ-. 1) = 0 for l
      using g-step-remains-orth 1 by (induction l, auto)

    have 0: g-norm ((g-step^^l) f) ≤ Λa ^ l * g-norm f
    proof (induction l)
      case 0
      then show ?case by simp
    next
      case (Suc l)
      have g-norm ((g-step ^^ Suc l) f) = g-norm (g-step ((g-step ^^ l) f))
        by simp
      also have ... ≤ Λa * g-norm (((g-step ^^ l) f))
        by (intro expansionD2 2)
      also have ... ≤ Λa * (Λa^l * g-norm f)
        by (intro mult-left-mono Λ-ge-0 Suc)
      also have ... = Λa^(l+1) * g-norm f by simp
      finally show ?case by simp
    qed

    have ?L = |g-inner f (H.g-step f)|
      unfolding H.g-inner-def g-inner-def
      by (intro-cong [σ1 abs] more:sum.cong) (auto simp add:graph-power-def)
    also have ... = |g-inner f ((g-step^^l) f)|
      by (intro-cong [σ1 abs] more:g-inner-cong graph-power-step) auto
    also have ... ≤ g-norm f * g-norm ((g-step^^l) f)
      by (intro g-inner-cauchy-schwartz)
    also have ... ≤ g-norm f * (Λa ^ l * g-norm f)
      by (intro mult-left-mono 0 g-norm-nonneg)
    also have ... = Λa ^ l * g-norm f^2
      by (simp add:power2-eq-square)
    also have ... = ?R
      unfolding g-norm-sq H.g-norm-sq g-inner-def H.g-inner-def
      by (intro-cong [σ2 (*)] more:sum.cong) (auto simp add:graph-power-def)
    finally show ?thesis by simp
  qed
  moreover have  0 ≤ Λa ^ l
    using Λ-ge-0 by simp

  ultimately show ?thesis
    by (intro H.expander-intro-1) auto
qed

unbundle no intro-cong-syntax
```

**end**

## 11 Strongly Explicit Expander Graphs

In some applications, representing an expander graph using a data structure (for example as an adjacency lists) would be prohibitive. For such cases strongly explicit expander graphs (SEE) are relevant. These are expander graphs, which can be represented implicitly using a function that computes for each vertex its neighbors in space and time logarithmic w.r.t. to the size of the graph. An application can for example sample a random walk, from a SEE using such a function efficiently. An example of such a graph is the Margulis construction from Section 8. This section presents the latter as a SEE but also shows that two graph operations that preserve the SEE property, in particular the graph power construction from Section 10 and a compression scheme introduced by Murtagh et al. [9, Theorem 20]. Combining all of the above it is possible to construct strongly explicit expander graphs of *every size* and spectral gap.

**theory** *Expander-Graphs-Strongly-Explicit*
  **imports** *Expander-Graphs-Power-Construction Expander-Graphs-MGG*
**begin**

**unbundle** *intro-cong-syntax*
**no-notation** *Digraph.dominates* (‹- →ı -› [*100*,*100*] *40*)

**record** *strongly-explicit-expander* =
  *see-size* :: *nat*
  *see-degree* :: *nat*
  *see-step* :: *nat* ⇒ *nat* ⇒ *nat*

**definition** *graph-of* :: *strongly-explicit-expander* ⇒ (*nat*, (*nat*,*nat*) *arc*) *pre-digraph*
  **where** *graph-of e* =
    (| *verts* = {..<*see-size e*},
      *arcs* = (λ(*v*, *i*). *Arc v* (*see-step e i v*) *i*) ‘ ({..<*see-size e*} × {..<*see-degree e*}),
      *tail* = *arc-tail*,
      *head* = *arc-head* |)

**definition** *is-expander e* $\Lambda_a$ ⟷
  *regular-graph* (*graph-of e*) ∧ *regular-graph*.$\Lambda_a$ (*graph-of e*) ≤ $\Lambda_a$

**lemma** *is-expander-mono*:
  **assumes** *is-expander e a a* ≤ *b*
  **shows** *is-expander e b*
  **using** *assms* **unfolding** *is-expander-def* **by** *auto*

**lemma** *graph-of-finI*:
  **assumes** *see-step e* ∈ ({..<*see-degree e*} → ({..<*see-size e*} → {..<*see-size e*}))
  **shows** *fin-digraph* (*graph-of e*)
**proof** −
  **let** *?G* = *graph-of e*

  **have** *head ?G a* ∈ *verts ?G* ∧ *tail ?G a* ∈ *verts ?G* **if** *a* ∈ *arcs ?G* **for** *a*
    **using** *assms that* **unfolding** *graph-of-def* **by** (*auto simp add*:*Pi-def*)

  **hence** *0*: *wf-digraph ?G*
    **unfolding** *wf-digraph-def* **by** *auto*

  **have** *1*: *finite* (*verts ?G*)

**unfolding** *graph-of-def* **by** *simp*

**have** *2*: *finite* (*arcs ?G*)
**unfolding** *graph-of-def* **by** *simp*
**show** *?thesis*
**using** *0 1 2* **unfolding** *fin-digraph-def fin-digraph-axioms-def* **by** *auto*
**qed**

**lemma** *edges-graph-of*:
*edges*(*graph-of e*)={#(*v*,*see-step e i v*). (*v*,*i*)∈#*mset-set* ({..<*see-size e*}×{..<*see-degree e*})#}
**proof** −
**have** *0*:*mset-set* ((λ(*v*, *i*). *Arc v* (*see-step e i v*) *i*) ‘ ({..<*see-size e*} × {..<*see-degree e*}))
= {# *Arc v* (*see-step e i v*) *i*. (*v*,*i*) ∈# *mset-set* ( {..<*see-size e*} × {..<*see-degree e*})#}
**by** (*intro image-mset-mset-set*[*symmetric*] *inj-onI*) *auto*

**have** *edges* (*graph-of e*) =
{#(*fst p*, *see-step e* (*snd p*) (*fst p*)). *p* ∈# *mset-set* ({..<*see-size e*} × {..<*see-degree e*})#}
**unfolding** *edges-def graph-of-def arc-to-ends-def* **using** *0*
**by** (*simp add*:*image-mset.compositionality comp-def case-prod-beta*)
**also have** ... = {#(*v*, *see-step e i v*). (*v*,*i*) ∈# *mset-set* ({..<*see-size e*} × {..<*see-degree e*})#}
**by** (*intro image-mset-cong*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *out-degree-see*:
**assumes** *v* ∈ *verts* (*graph-of e*)
**shows** *out-degree* (*graph-of e*) *v* = *see-degree e* (**is** *?L* = *?R*)
**proof** −
**let** *?d* = *see-degree e*
**let** *?n* = *see-size e*
**have** *0*: *v* < *?n*
**using** *assms* **unfolding** *graph-of-def* **by** *simp*

**have** *?L* = *card* {*a*. (∃ *x*∈{..<*?n*}. ∃ *y*∈{..<*?d*}. *a* = *Arc x* (*see-step e y x*) *y*) ∧ *arc-tail a* = *v*}
**unfolding** *out-degree-def out-arcs-def graph-of-def* **by** (*simp add*:*image-iff*)
**also have** ... = *card* {*a*. (∃ *y*∈{..<*?d*}. *a* = *Arc v* (*see-step e y v*) *y*)}
**using** *0* **by** (*intro arg-cong*[**where** *f*=*card*]) *auto*
**also have** ... = *card* ((λ*y*. *Arc v* (*see-step e y v*) *y*) ‘ {..<*?d*})
**by** (*intro arg-cong*[**where** *f*=*card*] *iffD2*[*OF set-eq-iff*]) (*simp add*:*image-iff*)
**also have** ... = *card* {..<*?d*}
**by** (*intro card-image inj-onI*) *auto*
**also have** ... = *?d* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *card-arc-walks-see*:
**assumes** *fin-digraph* (*graph-of e*)
**shows** *card* (*arc-walks* (*graph-of e*) *n*) = *see-degree e*^*n* ∗ *see-size e* (**is** *?L* = *?R*)
**proof** −
**let** *?G* = *graph-of e*
**interpret** *fin-digraph ?G*
**using** *assms* **by** *auto*
**have** *?L* = *card* (⋃ *v* ∈ *verts ?G*. {*x*. *fst x* = *v* ∧ *is-arc-walk ?G v* (*snd x*) ∧ *length* (*snd x*) = *n*})
**unfolding** *arc-walks-def* **by** (*intro arg-cong*[**where** *f*=*card*]) *auto*
**also have** ... = (∑ *v* ∈ *verts ?G*. *card* {*x*. *fst x*=*v*∧*is-arc-walk ?G v* (*snd x*)∧*length* (*snd x*) = *n*})
**using** *is-arc-walk-set*[**where** *G*=*?G*]

```
    by (intro card-UN-disjoint ballI finite-cartesian-product subsetI finite-lists-length-eq
        finite-subset[where B=verts ?G × {x. set x ⊆ arcs ?G ∧ length x = n}]) force+
  also have ... = (∑ v ∈ verts ?G. out-degree (graph-power ?G n) v)
    unfolding out-degree-def graph-power-def out-arcs-def arc-walks-def
    by (intro sum.cong arg-cong[where f=card]) auto
  also have ... = (∑ v ∈ verts ?G. see-degree e^n)
   by (intro sum.cong graph-power-out-degree' out-degree-see refl) (simp-all add: graph-power-def)
  also have ... = ?R
    by (simp add:graph-of-def)
  finally show ?thesis by simp
qed

lemma regular-graph-degree-eq-see-degree:
  assumes regular-graph (graph-of e)
  shows regular-graph.d (graph-of e) = see-degree e (is ?L = ?R)
proof −
  interpret regular-graph graph-of e
    using assms(1) by simp
  obtain v where v-set: v ∈ verts (graph-of e)
    using verts-non-empty by auto
  hence ?L = out-degree (graph-of e) v
    using v-set reg by auto
  also have ... = see-degree e
    by (intro out-degree-see v-set)
  finally show ?thesis by simp
qed
```

The following introduces the compression scheme, described in [9, Theorem 20].

```
fun see-compress :: nat ⇒ strongly-explicit-expander ⇒ strongly-explicit-expander
  where see-compress m e =
    (| see-size = m, see-degree = see-degree e * 2
    , see-step = (λk v.
      if k < see-degree e
        then (see-step e k v) mod m
        else (if v+m < see-size e then (see-step e (k−see-degree e) (v+m)) mod m else v)) |)

lemma edges-of-compress:
  fixes e m
  assumes 2*m ≥ see-size e m ≤ see-size e
  defines A ≡ {# (x mod m, y mod m). (x,y) ∈# edges (graph-of e)#}
  defines B ≡ repeat-mset (see-degree e) {# (x,x). x ∈# (mset-set {see-size e − m..<m})#}
  shows edges (graph-of (see-compress m e)) = A + B (is ?L = ?R)
proof −
  let ?d = see-degree e
  let ?c = see-step (see-compress m e)
  let ?n = see-size e
  let ?s = see-step e

  have 7:m ≤ v ⟹ v < ?n ⟹ v − m = v mod m for v
    using assms by (simp add: le-mod-geq)

  let ?M = mset-set ({..<m}×{..<2*?d})
  define M1 where M1 = mset-set ({..<m} × {..<?d})
  define M2 where M2 = mset-set ({..<?n−m} × {?d..<2*?d})
  define M3 where M3 = mset-set ({?n−m..<m} × {?d..<2*?d})

  have M2 = mset-set ((λ(x,y). (x−m,y+?d)) ` ({m..<?n} × {..<?d}))
    using assms(2) unfolding M2-def map-prod-def[symmetric] atLeast0LessThan[symmetric]
```

**by** (*intro arg-cong*[**where** *f*=*mset-set*] *map-prod-surj-on*[*symmetric*])
(*simp-all add*: *image-minus-const-atLeastLessThan-nat mult-2*)
**also have** ... = *image-mset* ($\lambda$(*x*,*y*). (*x*−*m*,*y*+*?d*)) (*mset-set* ({*m*..<*?n*} × {..<*?d*}))
**by** (*intro image-mset-mset-set*[*symmetric*] *inj-onI*) *auto*
**finally have** *M2-eq*: *M2* = *image-mset* ($\lambda$(*x*,*y*). (*x*−*m*,*y*+*?d*)) (*mset-set* ({*m*..<*?n*} × {..<*?d*}))
**by** *simp*

**have** *?M* = *mset-set* ({..<*m*}×{..<*?d*} ∪ {..<*?n*−*m*}×{*?d*..<2∗*?d*} ∪ {*?n*−*m*..<*m*}×{*?d*..<2∗*?d*})
**using** *assms*(*1*,*2*) **by** (*intro arg-cong*[**where** *f*=*mset-set*]) *auto*
**also have** ... = *mset-set* ({..<*m*}×{..<*?d*} ∪ {..<*?n*−*m*}×{*?d*..<2∗*?d*}) + *M3*
**unfolding** *M3-def* **by** (*intro mset-set-Union*) *auto*
**also have** ... = *M1* + *M2* + *M3*
**unfolding** *M1-def M2-def*
**by** (*intro arg-cong2*[**where** *f*=(+)] *mset-set-Union*) *auto*
**finally have** *0*:*mset-set* ({..<*m*} × {..<2∗*?d*}) = *M1* + *M2* + *M3* **by** *simp*

**have** *1*:{#(*v*,*?c i v*). (*v*,*i*)∈#*M1*#}={#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set* ({..<*m*}×{..<*?d*})#}
**unfolding** *M1-def* **by** (*intro image-mset-cong*) *auto*

**have** {#(*v*,*?c i v*).(*v*,*i*)∈#*M2*#}={#(*fst x*−*m*, *?c*(*snd x*+*?d*)(*fst x*−*m*)).*x*∈#*mset-set*({*m*..<*?n*}×{..<*?d*})#}
**unfolding** *M2-eq*
**by** (*simp add*:*image-mset.compositionality comp-def case-prod-beta del*:*see-compress.simps*)
**also have** ... = {#(*v* − *m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set* ({*m*..<*?n*}×{..<*?d*})#}
**by** (*intro image-mset-cong*) *auto*
**also have** ... = {#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set* ({*m*..<*?n*}×{..<*?d*})#}
**using** *7* **by** (*intro image-mset-cong*) *auto*
**finally have** *2*:
{#(*v*,*?c i v*). (*v*,*i*)∈#*M2*#}={#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set* ({*m*..<*?n*}×{..<*?d*})#}
**by** *simp*

**have** {#(*v*,*?c i v*). (*v*,*i*)∈#*M3*#} = {#(*v*,*v*). (*v*,*i*) ∈# *mset-set* ({*?n*−*m*..<*m*} × {*?d*..<2∗*?d*})#}
**unfolding** *M3-def* **by** (*intro image-mset-cong*) *auto*
**also have** ... = *concat-mset* {#{#(*x*, *x*). *xa* ∈# *mset-set* {*?d*..<2 ∗ *?d*}#}. *x* ∈# *mset-set* {*?n* − *m*..<*m*}#}
**by** (*subst mset-prod-eq*) (*auto simp*:*image-mset.compositionality image-concat-mset comp-def*)
**also have** ... = *concat-mset* {#*replicate-mset ?d* (*x*, *x*). *x* ∈# *mset-set* {*?n* − *m*..<*m*}#}
**unfolding** *image-mset-const-eq* **by** *simp*
**also have** ... = *B*
**unfolding** *B-def repeat-image-concat-mset* **by** *simp*
**finally have** *3*:{#(*v*,*?c i v*). (*v*,*i*)∈#*M3*#}=*B* **by** *simp*

**have** *A* = {#(*fst x mod m*, *?s* (*snd x*) (*fst x*) *mod m*). *x* ∈# *mset-set* ({..<*?n*} × {..<*?d*})#}
**unfolding** *A-def edges-graph-of* **by** (*simp add*:*image-mset.compositionality comp-def case-prod-beta*)
**also have** ... = {#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set*({..<*?n*}×{..<*?d*})#}
**by** (*intro image-mset-cong*) *auto*
**finally have** *4*: *A* = {#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set*({..<*?n*}×{..<*?d*})#}
**by** *simp*

**have** *?L* = {# (*v*, *?c i v*). (*v*,*i*) ∈# *?M* #}
**unfolding** *edges-graph-of* **by** (*simp add*:*ac-simps*)
**also have** ... = {#(*v*,*?c i v*). (*v*,*i*)∈#*M1*#}+{#(*v*,*?c i v*). (*v*,*i*)∈#*M2*#}+{#(*v*,*?c i v*). (*v*,*i*)∈#*M3*#}
**unfolding** *0 image-mset-union* **by** *simp*
**also have** ...={#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set*({..<*m*}×{..<*?d*}∪{*m*..<*?n*}×{..<*?d*})#}+*B*
**unfolding** *1 2 3 image-mset-union*[*symmetric*]
**by** (*intro-cong* [$\sigma_2$ (+), $\sigma_2$ *image-mset*] *more*: *mset-set-Union*[*symmetric*]) *auto*
**also have** ...={#(*v mod m*,*?s i v mod m*). (*v*,*i*)∈#*mset-set*({..<*?n*}×{..<*?d*})#}+*B*
**using** *assms*(*2*) **by** (*intro-cong* [$\sigma_2$ (+), $\sigma_2$ *image-mset*, $\sigma_1$ *mset-set*]) *auto*
**also have** ... = *A* + *B*

**unfolding** *4* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *see-compress-sym*:
**assumes** *2∗m ≥ see-size e m ≤ see-size e*
**assumes** *symmetric-multi-graph* (*graph-of e*)
**shows** *symmetric-multi-graph* (*graph-of* (*see-compress m e*))
**proof** −
**let** *?c* = *see-compress m e*
**let** *?d* = *see-degree e*
**let** *?G* = *graph-of e*
**let** *?H* = *graph-of* (*see-compress m e*)

**interpret** *G*:*fin-digraph ?G*
**by** (*intro symmetric-multi-graphD2*[*OF assms*(*3*)])
**interpret** *H*:*fin-digraph ?H*
**by** (*intro graph-of-finI*) *simp*

**have** *deg-compres*: *see-degree ?c* = *2* ∗ *see-degree e*
**by** *simp*

**have** *1*: *card* (*arcs-betw ?H v w*) = *card* (*arcs-betw ?H w v*) (**is** *?L* = *?R*)
**if** *v* ∈ *verts ?H w* ∈ *verts ?H* **for** *v w*
**proof** −
**define** *b* **where** *b* =*count* {#(*x*, *x*). *x* ∈# *mset-set* {*see-size e* − *m*..<*m*}#} (*v*, *w*)

**have** *b-alt-def*: *b* = *count* {#(*x*, *x*). *x* ∈# *mset-set* {*see-size e* − *m*..<*m*}#} (*w*, *v*)
**unfolding** *b-def count-mset-exp*
**by** (*simp add*:*case-prod-beta image-mset-filter-mset-swap*[*symmetric*] *ac-simps*)

**have** *?L* = *count* (*edges ?H*) (*v*,*w*)
**unfolding** *H*.*count-edges* **by** *simp*
**also have** ... = *count* {#(*x mod m*, *y mod m*). (*x*, *y*) ∈# *edges* (*graph-of e*)#} (*v*, *w*) + *?d* ∗ *b*
**unfolding** *edges-of-compress*[*OF assms*(*1*,*2*)] *b-def* **by** *simp*
**also have** ... = *count* {#(*snd e mod m*, *fst e mod m*). *e* ∈# *edges* (*graph-of e*)#} (*v*, *w*) + *?d* ∗ *b*
**by** (*subst G*.*edges-sym*[*OF assms*(*3*),*symmetric*])
(*simp add*:*image-mset*.*compositionality comp-def case-prod-beta*)
**also have** ... = *count* {#(*x mod m*, *y mod m*). (*x*,*y*) ∈# *edges* (*graph-of e*)#} (*w*, *v*) + *?d* ∗ *b*
**unfolding** *count-mset-exp*
**by** (*simp add*:*image-mset-filter-mset-swap*[*symmetric*] *ac-simps case-prod-beta*)
**also have** ... = *count* (*edges ?H*) (*w*,*v*)
**unfolding** *edges-of-compress*[*OF assms*(*1*,*2*)] *b-alt-def* **by** *simp*
**also have** ... = *?R*
**unfolding** *H*.*count-edges* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**show** *?thesis*
**using** *1 H*.*fin-digraph-axioms*
**unfolding** *symmetric-multi-graph-def* **by** *auto*
**qed**

**lemma** *see-compress*:
**assumes** *is-expander e* $\Lambda_a$
**assumes** *2∗m ≥ see-size e m ≤ see-size e*
**shows** *is-expander* (*see-compress m e*) ($\Lambda_a$/*2* + *1*/*2*)

```
proof −
  let ?H = graph-of (see-compress m e)
  let ?G = graph-of e
  let ?d = see-degree e
  let ?n = see-size e

  interpret G:regular-graph graph-of e
    using assms(1) is-expander-def by simp

  have d-eq: ?d = G.d
    using regular-graph-degree-eq-see-degree[OF G.regular-graph-axioms] by simp

  have n-eq: G.n = ?n
    unfolding G.n-def by (simp add:graph-of-def)

  have n-gt-1: ?n > 0
    using G.n-gt-0 n-eq by auto

  have symmetric-multi-graph (graph-of (see-compress m e))
    by (intro see-compress-sym assms(2,3) G.sym)
  moreover have see-size e > 0
    using G.verts-non-empty unfolding graph-of-def by auto
  hence m > 0 using assms(2) by simp
  hence verts (graph-of (see-compress m e)) ≠ {}
    unfolding graph-of-def by auto
  moreover have 1:0 < see-degree e
    using d-eq G.d-gt-0 by auto
  hence 0 < see-degree (see-compress m e) by simp
  ultimately have 0:regular-graph ?H
    by (intro regular-graphI[where d=see-degree (see-compress m e)] out-degree-see) auto
  interpret H:regular-graph ?H
    using 0 by auto

  have |∑a∈arcs ?H. f (head ?H a) ∗ f (tail ?H a)| ≤ (real G.d ∗ G.Λa + G.d) ∗ (H.g-norm f)^2
    (is ?L ≤ ?R) if H.g-inner f (λ-. 1) = 0 for f
  proof −
    define f' where f' x = f (x mod m) for x
    let ?L1 = G.g-norm f'^2 + |∑x=?n−m..<m. f x^2|
    let ?L2 = G.g-inner f' (λ-.1)^2/ G.n + |∑x=?n−m..<m. f x^2|

    have ?L1 = (∑x<?n. f (x mod m)^2) + |∑x=?n−m..<m. f x^2|
      unfolding G.g-norm-sq G.g-inner-def f'-def by (simp add:graph-of-def power2-eq-square)
    also have ... = (∑x∈{0..<m} ∪ {m..<?n}. f (x mod m)^2) + (∑x=?n−m..<m. f x^2)
      using assms(3) by (intro-cong [σ2 (+)] more:sum.cong abs-of-nonneg sum-nonneg) auto
   also have ...=(∑x=0..<m. f (x mod m)^2) + (∑x=m..<?n. f (x mod m)^2) + (∑x=?n−m..<m.
f x^2)
      by (intro-cong [σ2 (+)] more:sum.union-disjoint) auto
   also have ... = (∑x=0..<m. f (x mod m)^2) + (∑x=0..<?n−m. f x^2) + (∑x=?n−m..<m.
f x^2)
      using assms(2,3)
      by (intro-cong [σ2 (+)] more: sum.reindex-bij-betw bij-betwI[where g=(λx. x+m)])
       (auto simp add:le-mod-geq)
    also have ... = (∑x=0..<m. f x^2) + (∑x=0..<?n−m. f x^2) + (∑x=?n−m..<m. f x^2)
      by (intro sum.cong arg-cong2[where f=(+)]) auto
    also have ... = (∑x=0..<m. f x^2) + ((∑x=0..<?n−m. f x^2) + (∑x=?n−m..<m. f x^2))
      by simp
    also have ... = (∑x=0..<m. f x^2) + (∑x∈{0..<?n−m}∪{?n−m..<m}. f x^2)
      by (intro sum.union-disjoint[symmetric] arg-cong2[where f=(+)]) auto
```

**also have** ... = (∑ *x<m. f x^2*) + (∑ *x<m. f x^2*)
  **using** *assms(2,3)* **by** (*intro arg-cong2*[**where** *f*=(+)] *sum.cong*) *auto*
**also have** ... = *2* ∗ *H.g-norm f^2*
  **unfolding** *mult-2 H.g-norm-sq H.g-inner-def* **by** (*simp add*:*graph-of-def power2-eq-square*)
**finally have** *2*:*?L1* = *2* ∗ *H.g-norm f^2* **by** *simp*

**have** *?L2* = (∑ *x*∈{..<*m*}∪{*m*..<*?n*}. *f* (*x mod m*))^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **unfolding** *G.g-inner-def f'-def* **using** *assms(2,3)*
  **by** (*intro-cong* [σ₂ (+), σ₂ (/), σ₂ (*power*)] *more*: *sum.cong abs-of-nonneg sum-nonneg*)
   (*auto simp add*:*graph-of-def*)
**also have** ...=((∑ *x*<*m. f* (*x mod m*))+(∑ *x*=*m*..<*?n. f* (*x mod m*)))^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m.*
*f x^2*)
  **by** (*intro-cong* [σ₂ (+), σ₂ (/), σ₂ (*power*)] *more*:*sum.union-disjoint*) *auto*
**also have** ...=((∑ *x*<*m. f* (*x mod m*))+(∑ *x*=*0*..<*?n*−*m. f x*))^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m.*
*f x^2*)
  **using** *assms(2,3)* **by** (*intro-cong* [σ₂ (+), σ₂ (/), σ₂ (*power*)]
    *more*:*sum.reindex-bij-betw bij-betwI*[**where** *g*=(λ*x. x*+*m*)]) (*auto simp add*:*le-mod-geq*)
**also have** ...=(*H.g-inner f* (λ-. *1*) +(∑ *x*<*?n*−*m. f x*))^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **unfolding** *H.g-inner-def*
  **by** (*intro-cong* [σ₂ (+), σ₂ (/), σ₂ (*power*)] *more*: *sum.cong*) (*auto simp*:*graph-of-def*)
**also have** ...=(∑ *x*<*?n*−*m. f x*)^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **unfolding** *that* **by** *simp*
**also have** ...≤ (∑ *x*<*?n*−*m.* |*f x*| ∗ |*1*|)^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **by** (*intro add-mono divide-right-mono iffD1*[*OF abs-le-square-iff*]) *auto*
**also have** ... ≤ (*L2-set f* {..<*?n*−*m*} ∗ *L2-set* (λ-. *1*) {..<*?n*−*m*})^*2*/*G.n* + (∑ *x*=*?n*−*m*..<*m.*
*f x^2*)
  **by** (*intro add-mono divide-right-mono power-mono L2-set-mult-ineq sum-nonneg*) *auto*
**also have** ... = ((∑ *x* <*?n*−*m. f x^2*) ∗ (*?n*−*m*))/*G.n* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **unfolding** *power-mult-distrib L2-set-def real-sqrt-mult*
  **by** (*intro-cong* [σ₂ (+), σ₂ (/),σ₂ (∗)] *more*:*real-sqrt-pow2 sum-nonneg*) *auto*
**also have** ... = (∑ *x* <*?n*−*m. f x^2*) ∗ ((*?n*−*m*)/*?n*) + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **unfolding** *n-eq* **by** *simp*
**also have** ... ≤ (∑ *x* <*?n*−*m. f x^2*) ∗ *1* + (∑ *x*=*?n*−*m*..<*m. f x^2*)
  **using** *assms(3) n-gt-1* **by** (*intro mult-left-mono add-mono sum-nonneg*) *auto*
**also have** ... = (∑ *x*∈{..<*?n*−*m*}∪{*?n*−*m*..<*m*}. *f x^2*)
  **unfolding** *mult-1-right* **by** (*intro sum.union-disjoint*[*symmetric*]) *auto*
**also have** ... = *H.g-norm f^2*
  **using** *assms(2,3)* **unfolding** *H.g-norm-sq H.g-inner-def*
  **by** (*intro sum.cong*) (*auto simp add*:*graph-of-def power2-eq-square*)
**finally have** *3*:*?L2* ≤ *H.g-norm f^2* **by** *simp*

**have** *?L* = |∑ (*u, v*)∈#*edges ?H. f v* ∗ *f u*|
  **unfolding** *edges-def arc-to-ends-def sum-unfold-sum-mset*
  **by** (*simp add*:*image-mset.compositionality comp-def del*:*see-compress.simps*)
**also have** ...=|(∑ *x* ∈# *edges ?G.f*(*snd x mod m*)∗*f*(*fst x mod m*))+(∑ *x*=*?n*−*m*..<*m.?d*∗(*f*
*x^2*))|
  **unfolding** *edges-of-compress*[*OF assms(2,3)*] *sum-unfold-sum-mset*
  **by** (*simp add*:*image-mset.compositionality sum-mset-repeat comp-def*
    *case-prod-beta power2-eq-square del*:*see-compress.simps*)
**also have** ...=|(∑ (*u,v*) ∈# *edges ?G.f*(*u mod m*)∗*f*(*v mod m*))+(∑ *x*=*?n*−*m*..<*m.?d*∗(*f x^2*))|
  **by** (*intro-cong* [σ₁ *abs*, σ₂ (+), σ₁ *sum-mset*] *more*:*image-mset-cong*)
   (*simp-all add*:*case-prod-beta*)
**also have** ... ≤ |∑ (*u,v*) ∈# *edges ?G.f*(*u mod m*)∗*f*(*v mod m*)|+|∑ *x*=*?n*−*m*..<*m.?d*∗(*f x^2*)|

  **by** (*intro abs-triangle-ineq*)
**also have** ... = *?d* ∗ (|∑ (*u,v*) ∈# *edges ?G.f*(*v mod m*)∗*f*(*u mod m*)|/*G.d*+|∑ *x*=*?n*−*m*..<*m.*(*f*
*x^2*)|)
  **unfolding** *d-eq* **using** *G.d-gt-0*

**by** (*simp add*:*divide-simps ac-simps sum-distrib-left*[*symmetric*] *abs-mult*)
**also have** ... = *?d* ∗ (|*G*.*g-inner f′* (*G*.*g-step f′*)| + |∑ *x*=*?n*−*m*..<*m*. *f x*^*2*|)
**unfolding** *G*.*g-inner-step-eq sum-unfold-sum-mset edges-def arc-to-ends-def f′-def*
**by** (*simp add*:*image-mset*.*compositionality comp-def del*:*see-compress*.*simps*)
**also have** ...≤ *?d* ∗ ((*G*.$\Lambda_a$ ∗ *G*.*g-norm f′*^*2* + (*1*−*G*.$\Lambda_a$)∗*G*.*g-inner f′* (λ-.*1*)^*2*/ *G*.*n*)
+ |∑ *x*=*?n*−*m*..<*m*. *f x*^*2*|)
**by** (*intro add-mono G*.*expansionD3 mult-left-mono*) *auto*
**also have** ... = *?d* ∗ (*G*.$\Lambda_a$ ∗ *?L1* + (*1* − *G*.$\Lambda_a$) ∗ *?L2*)
**by** (*simp add*:*algebra-simps*)
**also have** ... ≤ *?d* ∗ (*G*.$\Lambda_a$ ∗ (*2* ∗ *H*.*g-norm f*^*2*) + (*1*−*G*.$\Lambda_a$) ∗ *H*.*g-norm f*^*2*)
**unfolding** *2* **using** *G*.Λ*-ge-0 G*.Λ*-le-1* **by** (*intro mult-left-mono add-mono 3*) *auto*
**also have** ... = *?R*
**unfolding** *d-eq*[*symmetric*] **by** (*simp add*:*algebra-simps*)
**finally show** *?thesis* **by** *simp*
**qed**

**hence** *H*.$\Lambda_a$ ≤ (*G*.*d*∗*G*.$\Lambda_a$+*G*.*d*)/*H*.*d*
**using** *G*.*d-gt-0 G*.Λ*-ge-0* **by** (*intro H*.*expander-intro*) (*auto simp del*:*see-compress*.*simps*)
**also have** ... = (*see-degree e* ∗ *G*.$\Lambda_a$ + *see-degree e*) / (*2*∗ *see-degree e*)
**unfolding** *d-eq*[*symmetric*] *regular-graph-degree-eq-see-degree*[*OF H*.*regular-graph-axioms*]
**by** *simp*
**also have** ... = *G*.$\Lambda_a$/*2* + *1*/*2*
**using** *1* **by** (*simp add*:*field-simps*)
**also have** ... ≤ $\Lambda_a$/*2* + *1*/*2*
**using** *assms*(*1*) **unfolding** *is-expander-def* **by** *simp*
**finally have** *H*.$\Lambda_a$ ≤ $\Lambda_a$/*2* + *1*/*2* **by** *simp*
**thus** *?thesis* **unfolding** *is-expander-def* **using** *0* **by** *simp*
**qed**

The graph power of a strongly explicit expander graph is itself a strongly explicit expander graph.

**fun** *to-digits* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat list*
**where**
*to-digits* - *0* - = [] |
*to-digits b* (*Suc l*) *k* = (*k mod b*)# *to-digits b l* (*k div b*)

**fun** *from-digits* :: *nat* ⇒ *nat list* ⇒ *nat*
**where**
*from-digits b* [] = *0* |
*from-digits b* (*x*#*xs*) = *x* + *b* ∗ *from-digits b xs*

**lemma** *to-from-digits*:
**assumes** *length xs* = *n set xs* ⊆ {..<*b*}
**shows** *to-digits b n* (*from-digits b xs*) = *xs*
**proof** −
**have** *to-digits b* (*length xs*) (*from-digits b xs*) = *xs*
**using** *assms*(*2*) **by** (*induction xs*, *auto*)
**thus** *?thesis* **unfolding** *assms*(*1*) **by** *auto*
**qed**

**lemma** *from-digits-range*:
**assumes** *length xs* = *n set xs* ⊆ {..<*b*}
**shows** *from-digits b xs* < *b*^*n*
**proof** (*cases b* > *0*)
**case** *True*
**have** *from-digits b xs* ≤ *b*^*length xs* − *1*
**using** *assms*(*2*)
**proof** (*induction xs*)

```
    case Nil
    then show ?case by simp
  next
    case (Cons a xs)
    have from-digits b (a # xs) =  a + b * from-digits b xs
      by simp
    also have ... ≤ (b−1) + b * from-digits b xs
      using Cons by (intro add-mono) auto
    also have ... ≤ (b−1) + b * (b^length xs−1)
      using Cons(2) by (intro add-mono mult-left-mono Cons(1)) auto
    also have ... = b^length (a#xs) − 1
      using True by (simp add:algebra-simps)
    finally show from-digits b (a # xs) ≤ b^length (a#xs) − 1 by simp
  qed
  also have ... < b^n
    using True assms(1) by simp
  finally show ?thesis by simp
next
  case False
  hence b = 0 by simp
  hence xs = []
    using assms(2) by simp
  thus ?thesis using assms(1) by simp
qed

lemma from-digits-inj:
  inj-on (from-digits b) {xs. set xs ⊆ {..<b} ∧ length xs = n}
  by (intro inj-on-inverseI[where g=to-digits b n] to-from-digits) auto

fun see-power :: nat ⇒ strongly-explicit-expander ⇒ strongly-explicit-expander
  where see-power l e =
    (| see-size = see-size e, see-degree = see-degree e^l
    , see-step = (λk v. foldl (λy x. see-step e x y) v (to-digits (see-degree e) l k)) |)

lemma graph-power-iso-see-power:
  assumes fin-digraph (graph-of e)
  shows digraph-iso (graph-power (graph-of e) n) (graph-of (see-power n e))
proof −
  let ?G = graph-of e
  let ?P = graph-power (graph-of e) n
  let ?H = graph-of (see-power n e)
  let ?d = see-degree e
  let ?n = see-size e

  interpret fin-digraph (graph-of e)
    using assms by auto

  interpret P:fin-digraph ?P
    by (intro graph-power-fin)

  define φ where
    φ = (λ(u,v). Arc u (arc-walk-head ?G (u, v)) (from-digits ?d (map arc-label v)))

  define iso where iso =
    (| iso-verts = id, iso-arcs = φ, iso-head = arc-head, iso-tail = arc-tail |)

  have xs = ys if length xs = length ys map arc-label xs = map arc-label ys
    is-arc-walk ?G u xs ∧ is-arc-walk ?G u ys ∧ u ∈ verts ?G for xs ys u
```

```
  using that
proof (induction xs ys arbitrary: u rule:list-induct2)
  case Nil
  then show ?case by simp
next
  case (Cons x xs y ys)
  have arc-label x = arc-label y u ∈ verts ?G x ∈ out-arcs ?G u y ∈ out-arcs ?G u
    using Cons by auto
  hence a:x = y
    unfolding graph-of-def by auto
  moreover have head ?G y ∈ verts ?G using Cons by auto
  ultimately have xs = ys
    using Cons(3,4) by (intro Cons(2)[of head ?G y]) auto
  thus ?case using a by auto
qed
hence 5:inj-on (λ(u,v). (u, map arc-label v)) (arc-walks ?G n)
  unfolding arc-walks-def by (intro inj-onI) auto
have 3:set (map arc-label (snd xs)) ⊆ {..<?d} length (snd xs) = n
  if xs ∈ arc-walks ?G n for xs
proof −
  show length (snd xs) = n
    using subsetD[OF is-arc-walk-set[where G=?G]] that unfolding arc-walks-def by auto
  have set (snd xs) ⊆ arcs ?G
    using subsetD[OF is-arc-walk-set[where G=?G]] that unfolding arc-walks-def by auto
  thus set (map arc-label (snd xs)) ⊆ {..<?d}
    unfolding graph-of-def by auto
qed

hence 7:inj-on (λ(u,v). (u, from-digits ?d (map arc-label v))) (arc-walks ?G n)
  using inj-onD[OF 5] inj-onD[OF from-digits-inj] by (intro inj-onI) auto

hence inj-on φ (arc-walks ?G n)
  unfolding inj-on-def φ-def by auto
hence inj-on (iso-arcs iso) (arcs (graph-power (graph-of e) n))
  unfolding iso-def graph-power-def by simp
moreover have inj-on (iso-verts iso) (verts (graph-power (graph-of e) n))
  unfolding iso-def by simp
moreover have
  iso-verts iso (tail ?P a) = iso-tail iso (iso-arcs iso a)
  iso-verts iso (head ?P a) = iso-head iso (iso-arcs iso a) if a ∈ arcs ?P for a
  unfolding φ-def iso-def graph-power-def by (simp-all add:case-prod-beta)
ultimately have 0:P.digraph-isomorphism iso
  unfolding P.digraph-isomorphism-def by (intro conjI ballI P.wf-digraph-axioms) auto

have card((λ(u, v).(u,from-digits ?d (map arc-label v)))‘arc-walks ?G n)=card(arc-walks ?G n)
  by (intro card-image 7)
also have ... = ?d^n ∗ ?n
  by (intro card-arc-walks-see fin-digraph-axioms)
finally have card((λ(u, v).(u,from-digits ?d (map arc-label v)))‘arc-walks ?G n) = ?d^n ∗ ?n
  by simp
moreover have fst v ∈ {..<?n} if v ∈ arc-walks ?G n for v
  using that unfolding arc-walks-def graph-of-def by auto
moreover have from-digits ?d (map arc-label (snd v)) < ?d ^ n if v ∈ arc-walks ?G n for v
  using 3[OF that] by (intro from-digits-range) auto

ultimately have 2:
  {..<?n}×{..<?d^n} = (λ(u,v). (u, from-digits ?d (map arc-label v))) ‘ arc-walks ?G n
  by (intro card-subset-eq[symmetric]) auto
```

**have** *foldl* (*λy x. see-step e x y*) *u* (*map arc-label w*) = *arc-walk-head ?G* (*u*,*w*)
  **if** *is-arc-walk ?G u w u* ∈ *verts ?G* **for** *u w*
  **using** *that*
**proof** (*induction w rule*:*rev-induct*)
  **case** *Nil*
  **then show** *?case* **by** (*simp add*:*arc-walk-head-def*)
**next**
  **case** (*snoc x xs*)
  **hence** *x* ∈ *arcs ?G* **by** (*simp add*:*is-arc-walk-snoc*)
  **hence** *see-step e* (*arc-label x*) (*tail ?G x*) = (*head ?G x*)
    **unfolding** *graph-of-def* **by** (*auto simp add*:*image-iff*)
  **also have** ... = *arc-walk-head* (*graph-of e*) (*u*, *xs* @ [*x*])
    **unfolding** *arc-walk-head-def* **by** *simp*
  **finally have** *see-step e* (*arc-label x*) (*tail ?G x*) = *arc-walk-head* (*graph-of e*) (*u*, *xs* @ [*x*])
    **by** *simp*
  **thus** *?case* **using** *snoc* **by** (*simp add*:*is-arc-walk-snoc*)
**qed**

**hence** *4*: *foldl* (*λy x. see-step e x y*) (*fst x*) (*map arc-label* (*snd x*)) = *arc-walk-head ?G x*
  **if** *x* ∈ *arc-walks* (*graph-of e*) *n* **for** *x*
  **using** *that* **unfolding** *arc-walks-def* **by** (*simp add*:*case-prod-beta*)

**have** *arcs ?H* = (*λ*(*v*, *i*). *Arc v* (*see-step* (*see-power n e*) *i v*) *i*) ' ({..<*?n*}×{..<*?d*^*n*})
  **unfolding** *graph-of-def* **by** *simp*
**also have** ... = (*λ*(*v*,*w*). *Arc v* (*see-step* (*see-power n e*) (*from-digits ?d* (*map arc-label w*)) *v*)
  (*from-digits ?d* (*map arc-label w*))) ' *arc-walks ?G n*
  **unfolding** *2 image-image* **by** (*simp del*:*see-power.simps add*: *case-prod-beta comp-def*)
**also have** ... = (*λ*(*v*,*w*). *Arc v* (*foldl* (*λy x. see-step e x y*) *v* (*map arc-label w*))
  (*from-digits ?d* (*map arc-label w*))) ' *arc-walks ?G n*
  **using** *3* **by** (*intro image-cong refl*) (*simp add*:*case-prod-beta to-from-digits*)
**also have** ... = *φ* ' *arc-walks ?G n*
  **unfolding** *φ-def* **using** *4* **by** (*simp add*:*case-prod-beta*)
**also have** ... = *iso-arcs iso* ' *arcs ?P*
  **unfolding** *iso-def graph-power-def* **by** *simp*
**finally have** *arcs ?H* = *iso-arcs iso* ' *arcs ?P*
  **by** *simp*
**moreover have** *verts ?H* = *iso-verts iso* ' *verts ?P*
  **unfolding** *iso-def graph-of-def graph-power-def* **by** *simp*
**moreover have** *tail ?H* = *iso-tail iso*
  **unfolding** *iso-def graph-of-def* **by** *simp*
**moreover have** *head ?H* = *iso-head iso*
  **unfolding** *iso-def graph-of-def* **by** *simp*
**ultimately have** *1*:*?H* = *app-iso iso ?P*
  **unfolding** *app-iso-def*
  **by** (*intro pre-digraph.equality*) (*simp-all del*:*see-power.simps*)

**show** *?thesis*
  **using** *0 1* **unfolding** *digraph-iso-def* **by** *auto*
**qed**

**lemma** *see-power*:
  **assumes** *is-expander e* $\Lambda_a$
  **shows** *is-expander* (*see-power n e*) ($\Lambda_a$^*n*)
**proof** −
  **interpret** *G*: *regular-graph graph-of e*
    **using** *assms* **unfolding** *is-expander-def* **by** *auto*

**interpret** *H*:*regular-graph graph-power* (*graph-of e*) *n*
  **by** (*intro G.graph-power-regular*)

**have** *0*:*digraph-iso* (*graph-power* (*graph-of e*) *n*) (*graph-of* (*see-power n e*))
  **by** (*intro graph-power-iso-see-power*) *auto*

**have** *regular-graph*.$\Lambda_a$ (*graph-of* (*see-power n e*)) = *H*.$\Lambda_a$
  **using** *H.regular-graph-iso-expansion*[*OF 0*] **by** *auto*
**also have** ... ≤ *G*.$\Lambda_a$^*n*
  **by** (*intro G.graph-power-expansion*)
**also have** ... ≤ $\Lambda_a$^*n*
  **using** *assms*(*1*) **unfolding** *is-expander-def*
  **by** (*intro power-mono G*.Λ-*ge-0*) *auto*
**finally have** *regular-graph*.$\Lambda_a$ (*graph-of* (*see-power n e*)) ≤ $\Lambda_a$^*n*
  **by** *simp*
**moreover have** *regular-graph* (*graph-of* (*see-power n e*))
  **using** *H.regular-graph-iso*[*OF 0*] **by** *auto*
**ultimately show** *?thesis*
  **unfolding** *is-expander-def* **by** *auto*
**qed**

The Margulis Construction from Section 8 is a strongly explicit expander graph.

**definition** *mgg-vert* :: *nat* ⇒ *nat* ⇒ (*int* × *int*)
  **where** *mgg-vert n x* = (*x mod n*, *x div n*)

**definition** *mgg-vert-inv* :: *nat* ⇒ (*int* × *int*) ⇒ *nat*
  **where** *mgg-vert-inv n x* = *nat* (*fst x*) + *nat* (*snd x*) ∗ *n*

**lemma** *mgg-vert-inv*:
  **assumes** *n* > *0 x* ∈ {*0*..<*int n*}×{*0*..<*int n*}
  **shows** *mgg-vert n* (*mgg-vert-inv n x*) = *x*
  **using** *assms* **unfolding** *mgg-vert-def mgg-vert-inv-def* **by** *auto*

**definition** *mgg-arc* :: *nat* ⇒ (*nat* × *int*)
  **where** *mgg-arc k* = (*k mod 4*, *if k* ≥ *4 then* (−*1*) *else 1*)

**definition** *mgg-arc-inv* :: (*nat* × *int*) ⇒ *nat*
  **where** *mgg-arc-inv x* = (*nat* (*fst x*) + *4* ∗ *of-bool* (*snd x* < *0*))

**lemma** *mgg-arc-inv*:
  **assumes** *x* ∈ {..<*4*}×{−*1*,*1*}
  **shows** *mgg-arc* (*mgg-arc-inv x*) = *x*
  **using** *assms* **unfolding** *mgg-arc-def mgg-arc-inv-def* **by** *auto*

**definition** *see-mgg* :: *nat* ⇒ *strongly-explicit-expander* **where**
  *see-mgg n* = (| *see-size* = *n*^*2*, *see-degree* = *8*,
    *see-step* = (λ*i v*. *mgg-vert-inv n* (*mgg-graph-step n* (*mgg-vert n v*) (*mgg-arc i*))) |)

**lemma** *mgg-graph-iso*:
  **assumes** *n* > *0*
  **shows** *digraph-iso* (*mgg-graph n*) (*graph-of* (*see-mgg n*))
**proof** −
  **let** *?v* = *mgg-vert n* **let** *?vi* = *mgg-vert-inv n*
  **let** *?a* = *mgg-arc* **let** *?ai* = *mgg-arc-inv*
  **let** *?G* = *graph-of* (*see-mgg n*) **let** *?s* = *mgg-graph-step n*

  **define** φ **where** φ *a* = *Arc* (*?vi* (*arc-tail a*)) (*?vi* (*arc-head a*)) (*?ai* (*arc-label a*)) **for** *a*

```
define iso where iso =
  ⦇ iso-verts = mgg-vert-inv n, iso-arcs = φ, iso-head = arc-head, iso-tail = arc-tail ⦈

interpret M: margulis-gaber-galil n
  using assms by unfold-locales

have inj-vi: inj-on ?vi (verts M.G)
  unfolding mgg-graph-def mgg-vert-inv-def
  by (intro inj-on-inverseI[where g=mgg-vert n]) (auto simp:mgg-vert-def)
have card (?vi ‘ verts M.G) = card (verts M.G)
  by (intro card-image inj-vi)
moreover have card (verts M.G) = n²
  unfolding mgg-graph-def by (auto simp:power2-eq-square)
moreover have mgg-vert-inv n x ∈ {..<n²} if x ∈ verts M.G for x
proof −
  have mgg-vert-inv n x = nat (fst x) + nat (snd x) ∗ n
    unfolding mgg-vert-inv-def by simp
  also have ... ≤ (n−1) + (n−1) ∗ n
    using that unfolding mgg-graph-def
    by (intro add-mono mult-right-mono) auto
  also have ... = n ∗ n − 1 using assms by (simp add:algebra-simps)
  also have ... < n^2
    using assms by (simp add: power2-eq-square)
  finally have mgg-vert-inv n x < n^2 by simp
  thus ?thesis by simp
qed
ultimately have 0:{..<n^2} = ?vi ‘ verts M.G
  by (intro card-subset-eq[symmetric] image-subsetI) auto

have inj-ai: inj-on ?ai ({..<4} × {−1,1})
  unfolding mgg-arc-inv-def by (intro inj-onI) auto
have card (?ai ‘ ({..<4} × {− 1, 1})) = card ({..<4::nat} × {−1,1::int})
  by (intro card-image inj-ai)
hence 1:{..<8} = ?ai ‘ ({..<4} × {−1,1})
  by (intro card-subset-eq[symmetric] image-subsetI) (auto simp add:mgg-arc-inv-def)

have arcs ?G = (λ(v, i). Arc v (?vi (?s (?v v) (?a i))) i) ‘ ({..<n²} × {..<8})
  by (simp add:see-mgg-def graph-of-def)
also have ... = (λ(v, i). Arc (?vi v) (?vi (?s (?v (?vi v)) (?a (?ai i)))) (?ai i)) ‘
  (verts M.G × ({..<4} × {−1,1}))
  unfolding 0 1 mgg-arc-inv by (auto simp add:image-iff)
also have ... = (λ(v, i). Arc (?vi v) (?vi (?s v i)) (?ai i)) ‘ (verts M.G × ({..<4} × {−1,1}))
   using mgg-vert-inv[OF assms] mgg-arc-inv unfolding mgg-graph-def by (intro image-cong)
auto
also have ... = (φ ∘ (λ(t, l). Arc t (?s t l) l)) ‘ (verts M.G × ({..<4} × {−1,1}))
  unfolding φ-def by (intro image-cong refl) ( simp add:comp-def case-prod-beta )
also have ... = φ ‘ arcs M.G
  unfolding mgg-graph-def by (simp add:image-image)
also have ... = iso-arcs iso ‘ arcs (mgg-graph n)
  unfolding iso-def by simp
finally have arcs (graph-of (see-mgg n)) = iso-arcs iso ‘ arcs (mgg-graph n)
  by simp
moreover have verts ?G = iso-verts iso ‘ verts (mgg-graph n)
  unfolding iso-def graph-of-def see-mgg-def using 0 by simp
moreover have tail ?G = iso-tail iso
  unfolding iso-def graph-of-def by simp
moreover have head ?G = iso-head iso
  unfolding iso-def graph-of-def by simp
```

**ultimately have** *0*:*?G* = *app-iso iso* (*mgg-graph n*)
  **unfolding** *app-iso-def* **by** (*intro pre-digraph.equality*) *simp-all*

**have** *inj-on* $\varphi$ (*arcs M.G*)
**proof** (*rule inj-onI*)
  **fix** *x y* **assume** *assms'*: $x \in$ *arcs M.G* $y \in$ *arcs M.G* $\varphi$ $x = \varphi$ $y$

  **have** *?vi* (*head M.G x*) = *?vi* (*head M.G y*)
    **using** *assms'*(*3*) **unfolding** $\varphi$*-def mgg-graph-def* **by** *auto*
  **hence** *head M.G x* = *head M.G y*
    **using** *assms'*(*1*,*2*) **by** (*intro inj-onD*[*OF inj-vi*]) *auto*
  **hence** *arc-head x* = *arc-head y*
    **unfolding** *mgg-graph-def* **by** *simp*

  **moreover have** *?vi* (*tail M.G x*) = *?vi* (*tail M.G y*)
    **using** *assms'*(*3*) **unfolding** $\varphi$*-def mgg-graph-def* **by** *auto*
  **hence** *tail M.G x* = *tail M.G y*
    **using** *assms'*(*1*,*2*) **by** (*intro inj-onD*[*OF inj-vi*]) *auto*
  **hence** *arc-tail x* = *arc-tail y*
    **unfolding** *mgg-graph-def* **by** *simp*

  **moreover have** *?ai* (*arc-label x*) = *?ai* (*arc-label y*)
    **using** *assms'*(*3*) **unfolding** $\varphi$*-def* **by** *auto*
  **hence** *arc-label x* = *arc-label y*
    **using** *assms'*(*1*,*2*) **unfolding** *mgg-graph-def*
    **by** (*intro inj-onD*[*OF inj-ai*]) (*auto simp del*:*mgg-graph-step.simps*)

  **ultimately show** $x = y$
    **by** (*intro arc.expand*) *auto*
**qed**
**hence** *inj-on* (*iso-arcs iso*) (*arcs M.G*)
  **unfolding** *iso-def* **by** *simp*
**moreover have** *inj-on* (*iso-verts iso*) (*verts M.G*)
  **using** *inj-vi* **unfolding** *iso-def* **by** *simp*
**moreover have**
  *iso-verts iso* (*tail M.G a*) = *iso-tail iso* (*iso-arcs iso a*)
  *iso-verts iso* (*head M.G a*) = *iso-head iso* (*iso-arcs iso a*) **if** $a \in$ *arcs M.G* **for** *a*
  **unfolding** *iso-def* $\varphi$*-def mgg-graph-def* **by** *auto*
**ultimately have** *1*:*M.digraph-isomorphism iso*
  **unfolding** *M.digraph-isomorphism-def* **by** (*intro conjI ballI M.wf-digraph-axioms*) *auto*

**show** *?thesis* **unfolding** *digraph-iso-def* **using** *0 1* **by** *auto*
**qed**

**lemma** *see-mgg*:
  **assumes** $n > 0$
  **shows** *is-expander* (*see-mgg n*) (*5*∗ *sqrt 2* / *8*)
**proof** −
  **interpret** *G*: *margulis-gaber-galil n*
    **using** *assms* **by** *unfold-locales auto*

  **note** *0* = *mgg-graph-iso*[*OF assms*]

  **have** *regular-graph.*$\Lambda_a$ (*graph-of* (*see-mgg n*)) = *G.*$\Lambda_a$
    **using** *G.regular-graph-iso-expansion*[*OF 0*] **by** *auto*
  **also have** ... $\leq$ (*5*∗ *sqrt 2* / *8*)
    **using** *G.mgg-numerical-radius* **unfolding** *G.MGG-bound-def* **by** *simp*
  **finally have** *regular-graph.*$\Lambda_a$ (*graph-of* (*see-mgg n*)) $\leq$ (*5*∗ *sqrt 2* / *8*)

**by** *simp*
**moreover have** *regular-graph* (*graph-of* (*see-mgg n*))
**using** *G.regular-graph-iso*[*OF 0*] **by** *auto*
**ultimately show** *?thesis*
**unfolding** *is-expander-def* **by** *auto*
**qed**

Using all of the above it is possible to construct strongly explicit expanders of every size and spectral gap with asymptotically optimal degree.

**definition** *see-standard-aux*
**where** *see-standard-aux n* = *see-compress n* (*see-mgg* (*nat* ⌈*sqrt n*⌉))

**lemma** *see-standard-aux*:
**assumes** *n* > *0*
**shows**
*is-expander* (*see-standard-aux n*) ((*8*+*5* ∗ *sqrt 2*) / *16*) (**is** *?A*)
*see-degree* (*see-standard-aux n*) = *16* (**is** *?B*)
*see-size* (*see-standard-aux n*) = *n* (**is** *?C*)
**proof** −
**have** *2*:*sqrt* (*real n*) > −*1*
**by** (*rule less-le-trans*[**where** *y*=*0*]) *auto*

**have** *0*:*real n* ≤ *of-int* ⌈*sqrt* (*real n*)⌉ˆ*2*
**by** (*simp add*:*sqrt-le-D*)

**consider** (*a*) *n* = *1* | (*b*) *n* ≥ *2* ∧ *n* ≤ *4* | (*c*) *n* ≥ *5* ∧ *n* ≤ *9* | (*d*) *n* ≥ *10*
**using** *assms* **by** *linarith*
**hence** *1*:*of-int* ⌈*sqrt* (*real n*)⌉ˆ*2* ≤ *2* ∗ *real n*
**proof** (*cases*)
**case** *a* **then show** *?thesis* **by** *simp*
**next**
**case** *b*
**hence** *real-of-int* ⌈*sqrt* (*real n*)⌉ˆ*2* ≤ *of-int* ⌈*sqrt* (*real 4*)⌉ˆ*2*
**using** *2*
**by** (*intro power-mono iffD2*[*OF of-int-le-iff*] *ceiling-mono iffD2*[*OF real-sqrt-le-iff*]) *auto*
**also have** ... = *2* ∗ *real 2* **by** *simp*
**also have** ... ≤ *2* ∗ *real n*
**using** *b* **by** (*intro mult-left-mono*) *auto*
**finally show** *?thesis* **by** *simp*
**next**
**case** *c*
**hence** *real-of-int* ⌈*sqrt* (*real n*)⌉ˆ*2* ≤ *of-int* ⌈*sqrt* (*real 9*)⌉ˆ*2*
**using** *2*
**by** (*intro power-mono iffD2*[*OF of-int-le-iff*] *ceiling-mono iffD2*[*OF real-sqrt-le-iff*]) *auto*
**also have** ... = *9* **by** *simp*
**also have** ... ≤ *2* ∗ *real 5* **by** *simp*
**also have** ... ≤ *2* ∗ *real n*
**using** *c* **by** (*intro mult-left-mono*) *auto*
**finally show** *?thesis* **by** *simp*
**next**
**case** *d*
**have** *real-of-int* ⌈*sqrt* (*real n*)⌉ˆ*2* ≤ (*sqrt* (*real n*)+*1*)ˆ*2*
**using** *2* **by** (*intro power-mono*) *auto*
**also have** ... = *real n* + *sqrt* (*4* ∗ *real n* + *0*) + *1*
**using** *real-sqrt-pow2* **by** (*simp add*:*power2-eq-square algebra-simps real-sqrt-mult*)
**also have** ... ≤ *real n* + *sqrt* (*4* ∗ *real n* + (*real n* ∗ (*real n* − *6*) + *1*)) + *1*
**using** *d* **by** (*intro add-mono iffD2*[*OF real-sqrt-le-iff*]) *auto*
**also have** ... = *real n* + *sqrt* ((*real n*−*1*)ˆ*2*) + *1*

```
      by (intro-cong [σ₂ (+), σ₁ sqrt]) (auto simp add:power2-eq-square algebra-simps)
    also have ... = 2 ∗ real n
      using d by simp
    finally show ?thesis by simp
  qed

  have nat ⌈sqrt (real n)⌉^2 ∈ {n..2∗n}
    by (simp add: approximation-preproc-nat(13) sqrt-le-D 1)
  hence see-size (see-mgg (nat ⌈sqrt (real n)⌉)) ∈ {n..2∗n}
    by (simp add:see-mgg-def)
  moreover have sqrt (real n) > 0 using assms by simp
  hence 0 < nat ⌈sqrt (real n)⌉ by simp
  ultimately have is-expander (see-standard-aux n) ((5∗ sqrt 2 / 8)/2 + 1/2)
    unfolding see-standard-aux-def by (intro see-compress see-mgg) auto
  thus ?A
    by (auto simp add:field-simps)
  show ?B
    unfolding see-standard-aux-def by (simp add:see-mgg-def)
  show ?C
    unfolding see-standard-aux-def by simp
qed

definition see-standard-power
  where see-standard-power x = (if x ≤ (0::real) then 0 else nat ⌈ln x / ln 0.95⌉)

lemma see-standard-power:
  assumes Λa > 0
  shows 0.95^(see-standard-power Λa) ≤ Λa (is ?L ≤ ?R)
proof (cases Λa ≤ 1)
  case True
  hence 0 ≤ ln Λa / ln 0.95
    using assms by (intro divide-nonpos-neg) auto
  hence 1:0 ≤ ⌈ln Λa / ln 0.95⌉
    by simp
  have ?L = 0.95^nat ⌈ln Λa / ln 0.95⌉
    using assms unfolding see-standard-power-def by simp
  also have ... = 0.95 powr (of-nat (nat (⌈ln Λa / ln 0.95⌉)))
    by (subst powr-realpow) auto
  also have ... = 0.95 powr ⌈ln Λa / ln 0.95⌉
    using 1 by (subst of-nat-nat) auto
  also have ... ≤ 0.95 powr (ln Λa / ln 0.95)
    by (intro powr-mono-rev) auto
  also have ... = ?R
    using assms unfolding powr-def by simp
  finally show ?thesis by simp
next
  case False
  hence ln Λa / ln 0.95 ≤ 0
    by (subst neg-divide-le-eq) auto
  hence see-standard-power Λa = 0
    unfolding see-standard-power-def by simp
  then show ?thesis using False by simp
qed

lemma see-standard-power-eval[code]:
  see-standard-power x = (if x ≤ 0 ∨ x ≥ 1 then 0 else (1+see-standard-power (x/0.95)))
proof (cases x ≤ 0 ∨ x ≥ 1)
  case True
```

**have** *ln x / ln (19 / 20) ≤ 0* **if** *x > 0*
**proof** −
**have** *x ≥ 1* **using** *that True* **by** *auto*
**thus** *?thesis*
**by** (*intro divide-nonneg-neg*) *auto*
**qed**
**then show** *?thesis* **using** *True* **unfolding** *see-standard-power-def* **by** *simp*
**next**
**case** *False*
**hence** *x-range*: *x > 0 x < 1* **by** *auto*

**have** *ln (x / 0.95) < ln (1/0.95)*
**using** *x-range* **by** (*intro iffD2*[*OF ln-less-cancel-iff*]) *auto*
**also have** ... = − *ln 0.95*
**by** (*subst ln-div*) *auto*
**finally have** *ln (x / 0.95) < − ln 0.95* **by** *simp*
**hence** *0*: −*1 < ln (x / 0.95) / ln 0.95*
**by** (*subst neg-less-divide-eq*) *auto*

**have** *see-standard-power x = nat ⌈ln x / ln 0.95⌉*
**using** *x-range* **unfolding** *see-standard-power-def* **by** *simp*
**also have** ... = *nat ⌈ln (x/0.95) / ln 0.95 + 1⌉*
**by** (*subst ln-divide-pos*[*OF x-range*(*1*)]) (*simp-all add*:*field-simps* )
**also have** ... = *nat* (⌈*ln (x/0.95) / ln 0.95*⌉+*1*)
**by** (*intro arg-cong*[**where** *f*=*nat*]) *simp*
**also have** ... = *1 + nat ⌈ln (x/0.95) / ln 0.95⌉*
**using** *0* **by** (*subst nat-add-distrib*) *auto*
**also have** ... = (*if x ≤ 0 ∨ 1 ≤ x then 0 else 1 + see-standard-power* (*x/0.95*))
**unfolding** *see-standard-power-def* **using** *x-range* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**definition** *see-standard* :: *nat ⇒ real ⇒ strongly-explicit-expander*
**where** *see-standard n* $\Lambda_a$ = *see-power* (*see-standard-power* $\Lambda_a$) (*see-standard-aux n*)

**theorem** *see-standard*:
**assumes** *n > 0* $\Lambda_a$ *> 0*
**shows** *is-expander* (*see-standard n* $\Lambda_a$) $\Lambda_a$
**and** *see-size* (*see-standard n* $\Lambda_a$) = *n*
**and** *see-degree* (*see-standard n* $\Lambda_a$) = *16* ^ (*nat* ⌈*ln* $\Lambda_a$ */ ln 0.95*⌉) (**is** *?C*)
**proof** −
**have** *0*:*is-expander* (*see-standard-aux n*) *0.95*
**by** (*intro see-standard-aux*(*1*)[*OF assms*(*1*)] *is-expander-mono*[**where** *a*=(*8+5 ∗ sqrt 2*) / *16*])
(*approximation 10*)

**show** *is-expander* (*see-standard n* $\Lambda_a$) $\Lambda_a$
**unfolding** *see-standard-def*
**by** (*intro see-power 0 is-expander-mono*[**where** *a*=*0.95*^(*see-standard-power* $\Lambda_a$)]
*see-standard-power assms*(*2*))
**show** *see-size* (*see-standard n* $\Lambda_a$) = *n*
**unfolding** *see-standard-def* **using** *see-standard-aux*[*OF assms*(*1*)] **by** *simp*

**have** *see-degree* (*see-standard n* $\Lambda_a$) = *16* ^ (*see-standard-power* $\Lambda_a$)
**unfolding** *see-standard-def* **using** *see-standard-aux*[*OF assms*(*1*)] **by** *simp*
**also have** ... = *16* ^ (*nat* ⌈*ln* $\Lambda_a$ */ ln 0.95*⌉)
**unfolding** *see-standard-power-def* **using** *assms*(*2*) **by** *simp*
**finally show** *?C* **by** *simp*

**qed**

**fun** *see-sample-walk* :: *strongly-explicit-expander* ⇒ *nat* ⇒ *nat* ⇒ *nat list*
**where**
*see-sample-walk e 0 x* = [*x*] |
*see-sample-walk e* (*Suc l*) *x* = (*let w* = *see-sample-walk e l* (*x div* (*see-degree e*)) *in*
*w*@[*see-step e* (*x mod* (*see-degree e*)) (*last w*)])

**theorem** *see-sample-walk*:
**fixes** *e l*
**assumes** *fin-digraph* (*graph-of e*)
**defines** *r* ≡ *see-size e* ∗ *see-degree e* ^*l*
**shows** {# *see-sample-walk e l k*. *k* ∈# *mset-set* {..<*r*} #} = *walks'* (*graph-of e*) *l*
**unfolding** *r-def*
**proof** (*induction l*)
**case** *0*
**then show** *?case* **unfolding** *graph-of-def* **by** *simp*
**next**
**case** (*Suc l*)
**interpret** *fin-digraph graph-of e*
**using** *assms*(*1*) **by** *auto*

**let** *?d* = *see-degree e*
**let** *?n* = *see-size e*
**let** *?w* = *see-sample-walk e*
**let** *?G* = *graph-of e*
**define** *r* **where** *r* = *?n* ∗ *?d*^*l*

**have** *1*: {*i* ∗ *?d*..<(*i* + *1*) ∗ *?d*} ∩ {*j* ∗ *?d*..<(*j* + *1*) ∗ *?d*} = {} **if** *i* ≠ *j* **for** *i j*
**using** *that index-div-eq* **by** *blast*

**have** *2*:*vertices-from ?G x* = {# *see-step e i x*. *i* ∈# *mset-set* {..<*?d*}#} (**is** *?L* = *?R*)
**if** *x* ∈ *verts ?G* **for** *x*
**proof** −
**have** *x* < *?n*
**using** *that* **unfolding** *graph-of-def* **by** *simp*
**hence** *1*:*out-arcs ?G x* = (λ*i*. *Arc x* (*see-step e i x*) *i*) ' {..<*?d*}
**unfolding** *out-arcs-def graph-of-def* **by** (*auto simp add*:*image-iff set-eq-iff*)

**have** *?L* = {# *arc-head a*. *a* ∈# *mset-set* (*out-arcs ?G x*) #}
**unfolding** *verts-from-alt* **by** (*simp add*:*graph-of-def*)
**also have** ... = {# *arc-head a*. *a* ∈# {# *Arc x* (*see-step e i x*) *i*. *i* ∈# *mset-set* {..<*?d*}#}#}

**unfolding** *1*
**by** (*intro arg-cong2*[**where** *f*= *image-mset*] *image-mset-mset-set*[*symmetric*] *inj-onI*) *auto*
**also have** ... = *?R*
**by** (*simp add*:*image-mset.compositionality comp-def*)
**finally show** *?thesis* **by** *simp*
**qed**

**have** *card* (⋃*w*<*r*. {*w* ∗ *?d*..<(*w* + *1*) ∗*?d*}) = (∑ *w* < *r*. *card* {*w* ∗ *?d*..<(*w* + *1*) ∗*?d*})
**using** *1* **by** (*intro card-UN-disjoint*) *auto*
**also have** ... = *r* ∗ *?d* **by** *simp*
**finally have** *card* (⋃*w*<*r*. {*w* ∗ *?d*..<(*w* + *1*) ∗*?d*}) = *card* {..<*?d* ∗ *r*} **by** *simp*
**moreover have** *?d* + *z* ∗ *?d* ≤ *?d* ∗ *r* **if** *z* < *r* **for** *z*
**proof** −
**have** *?d* + *z* ∗ *?d* = *?d* ∗ (*z* + *1*) **by** *simp*
**also have** ... ≤ *?d* ∗ *r*

**using** *that* **by** (*intro mult-left-mono*) *auto*
**finally show** *?thesis* **by** *simp*
**qed**
**ultimately have** *0*: $(\bigcup w<r.\ \{w * ?d..<(w + 1) * ?d\}) = \{..<?d * r\}$
**using** *order-less-le-trans* **by** (*intro card-subset-eq subsetI*) *auto*

**have** {# *?w* (*l+1*) *k*. *k* ∈# *mset-set* {..<*?n* ∗ *?d*^(*l+1*)} #} = {#*?w* (*l+1*) *k*. *k* ∈# *mset-set* {..<*?d* ∗ *r*}#}
**unfolding** *r-def* **by** (*simp add*:*ac-simps*)
**also have** ... = {# *?w* (*l+1*) *x*. *x* ∈# *mset-set* (⋃*w*<*r*. {*w* ∗ *?d*..<(*w* + *1*) ∗ *?d*})#}
**unfolding** *0* **by** *simp*
**also have** ... = *image-mset* (*?w* (*l+1*)) (*concat-mset*
(*image-mset* (*mset-set* ∘ (λ*w*. {*w* ∗ *?d*..<(*w* + *1*) ∗ *?d*})) (*mset-set* {..<*r*})))
**by** (*intro arg-cong2*[**where** *f*=*image-mset*] *concat-disjoint-union-mset refl 1*) *auto*
**also have** ... = *concat-mset*{#{#*?w* (*l+1*) *i*. *i*∈#*mset-set* {*w*∗*?d*..<(*w*+*1*)∗*?d*}#}. *w*∈#*mset-set* {..<*r*}#}
**by** (*simp add*:*image-concat-mset image-mset.compositionality comp-def del*:*see-sample-walk.simps*)
**also have** ...=*concat-mset* {#{#*?w*(*l+1*)*i*. *i*∈#*mset-set* ((+)(*w*∗*?d*)‘{..<*?d*})#}. *w*∈#*mset-set* {..<*r*}#}
**by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*, $\sigma_1$ *mset-set*] *more*:*ext*)
(*simp add*: *atLeast0LessThan*[*symmetric*])
**also have** ... = *concat-mset*
{#{#*?w* (*l+1*) *i*. *i*∈#*image-mset* ((+) (*w*∗*?d*)) (*mset-set* {..<*?d*})#}. *w*∈#*mset-set* {..<*r*}#}
**by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*] *more*:*image-mset-cong*
*image-mset-mset-set*[*symmetric*] *inj-onI*) *auto*
**also have** ... = *concat-mset* {#{#*?w* (*l+1*) (*w*∗*?d*+*i*).*i*∈#*mset-set* {..<*?d*}#}. *w*∈#*mset-set* {..<*r*}#}
**by** (*simp add*:*image-mset.compositionality comp-def del*:*see-sample-walk.simps*)
**also have** ... = *concat-mset*
{#{#*?w l w*@[*see-step e i* (*last* (*?w l w*))].*i*∈#*mset-set* {..<*?d*}#}.*w*∈#*mset-set* {..<*r*}#}
**by** (*intro-cong* [$\sigma_1$ *concat-mset*] *more*:*image-mset-cong*) (*simp add*:*Let-def*)
**also have** ... = *concat-mset* {#{#*w*@[*see-step e i* (*last w*)].*i*∈#*mset-set* {..<*?d*}#}.*w*∈#*walks′* *?G l*#}
**unfolding** *r-def Suc*[*symmetric*] *image-mset.compositionality comp-def* **by** *simp*
**also have** ... = *concat-mset*
{#{#*w*@[*x*].*x*∈#{# *see-step e i* (*last w*). *i*∈#*mset-set* {..<*?d*}#}#}. *w* ∈# *walks′* *?G l*#}
**unfolding** *image-mset.compositionality comp-def* **by** *simp*
**also have** ... = *concat-mset* {#{#*w*@[*x*].*x*∈#*vertices-from* *?G* (*last w*)#}. *w* ∈# *walks′* *?G l*#}
**using** *last-in-set set-walks-2*(*1*,*2*)
**by** (*intro-cong* [$\sigma_1$ *concat-mset*, $\sigma_2$ *image-mset*] *more*:*image-mset-cong 2*[*symmetric*]) *blast*
**also have** ... = *walks′* (*graph-of e*) (*l+1*)
**by** (*simp add*:*image-mset.compositionality comp-def*)
**finally show** *?case* **by** *simp*
**qed**

**unbundle** *no intro-cong-syntax*

**end**

# 12 Expander Walks as Pseudorandom Objects

**theory** *Pseudorandom-Objects-Expander-Walks*
**imports**
*Universal-Hash-Families.Pseudorandom-Objects*
*Expander-Graphs.Expander-Graphs-Strongly-Explicit*
**begin**

**unbundle** *intro-cong-syntax*
**hide-const** (**open**) *Quantum.T*
**hide-fact** (**open**) *SN-Orders.of-nat-mono*
**hide-fact** *Missing-Ring.mult-pos-pos*

**definition** *expander-pro* ::
  *nat* ⇒ *real* ⇒ (*'a,'b*) *pseudorandom-object-scheme* ⇒ (*nat* ⇒ *'a*) *pseudorandom-object*
  **where** *expander-pro l* Λ *S* = (
    *let e* = *see-standard* (*pro-size S*) Λ *in*
      (| *pro-last* = *see-size e* ∗ *see-degree e*^(*l*−*1*) − *1*,
        *pro-select* = (λ*i j*. *pro-select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j mod pro-size S*)) |)
    )

**context**
  **fixes** *l* :: *nat*
  **fixes** Λ :: *real*
  **fixes** *S* :: (*'a,'b*) *pseudorandom-object-scheme*
  **assumes** *l-gt-0*: *l* > *0*
  **assumes** Λ*-gt-0*: Λ > *0*
**begin**

**private definition** *e* **where** *e* = *see-standard* (*pro-size S*) Λ

**private lemma** *expander-pro-alt*: *expander-pro l* Λ *S* = (| *pro-last* = *see-size e* ∗ *see-degree e*^(*l*−*1*) − *1*,
        *pro-select* = (λ*i j*. *pro-select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j mod pro-size S*)) |)
  **unfolding** *expander-pro-def e-def*[*symmetric*] **by** (*auto simp*:*Let-def*)

**private lemmas** *see-standard* = *see-standard* [*OF pro-size-gt-0*[**where** *S*=*S*] Λ*-gt-0*]

**interpretation** *E*: *regular-graph graph-of e*
  **using** *see-standard*(*1*) **unfolding** *is-expander-def e-def* **by** *auto*

**private lemma** *e-deg-gt-0*: *see-degree e* > *0*
  **unfolding** *e-def see-standard* **by** *simp*

**private lemma** *e-size-gt-0*: *see-size e* > *0*
  **unfolding** *e-def* **using** *see-standard pro-size-gt-0* **by** *simp*

**private lemma** *expander-sample-size*: *pro-size* (*expander-pro l* Λ *S*) = *see-size e* ∗ *see-degree e*^(*l*−*1*)
  **using** *e-deg-gt-0 e-size-gt-0* **unfolding** *expander-pro-alt pro-size-def* **by** *simp*

**private lemma** *sample-pro-expander-walks*:
  **defines** *R* ≡ *map-pmf* (λ*xs i*. *pro-select S* (*xs* ! *i mod pro-size S*))
    (*pmf-of-multiset* (*walks* (*graph-of e*) *l*))
  **shows** *sample-pro* (*expander-pro l* Λ *S*) = *R*
**proof** −
  **let** *?S* = {..<*see-size e* ∗ *see-degree e* ^ (*l*−*1*)}
  **let** *?T* = (*map-pmf* (*see-sample-walk e* (*l*−*1*)) (*pmf-of-set ?S*))

  **have** *0* ∈ *?S*
    **using** *e-size-gt-0 e-deg-gt-0* **by** *auto*
  **hence** *?S* ≠ {}
    **by** *blast*
  **hence** *?T* = *pmf-of-multiset* {#*see-sample-walk e* (*l*−*1*) *i*. *i* ∈# *mset-set ?S*#}
    **by** (*subst map-pmf-of-set*) *simp-all*
  **also have** ... = *pmf-of-multiset* (*walks'* (*graph-of e*) (*l*−*1*))

by (*subst see-sample-walk*) *auto*
**also have** ... = *pmf-of-multiset* (*walks* (*graph-of e*) *l*)
**unfolding** *walks-def* **using** *l-gt-0* **by** (*cases l*, *simp-all*)
**finally have** *0*:*?T* = *pmf-of-multiset* (*walks* (*graph-of e*) *l*)
**by** *simp*

**have** *sample-pro* (*expander-pro l* Λ *S*) = *map-pmf* (λ*xs j*. *pro-select S* (*xs* ! *j mod pro-size S*)) *?T*
**unfolding** *expander-sample-size sample-pro-alt* **unfolding** *map-pmf-comp expander-pro-alt* **by** *simp*
**also have** ... = *R* **unfolding** *0 R-def* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expander-pro-range*: *pro-select* (*expander-pro l* Λ *S*) *i j* ∈ *pro-set S*
**unfolding** *expander-pro-alt* **by** (*simp add*:*pro-select-in-set*)

**lemma** *expander-uniform-property*:
**assumes** *i* < *l*
**shows** *map-pmf* (λ*w*. *w i*) (*sample-pro* (*expander-pro l* Λ *S*)) = *sample-pro S* (**is** *?L* = *?R*)
**proof** −
**have** *?L* = *map-pmf* (λ*x*. *pro-select S* (*x mod pro-size S*)) (*map-pmf* (λ*xs*. (*xs* ! *i*)) (*pmf-of-multiset* (*walks* (*graph-of e*) *l*)))
**unfolding** *sample-pro-expander-walks* **by** (*simp add*: *map-pmf-comp*)
**also have** ... = *map-pmf* (λ*x*. *pro-select S* (*x mod pro-size S*)) (*pmf-of-set* (*verts* (*graph-of e*)))
**unfolding** *E.uniform-property*[*OF assms*] **by** *simp*
**also have** ... = *?R*
**using** *pro-size-gt-0* **unfolding** *sample-pro-alt*
**by** (*intro map-pmf-cong*) (*simp-all add*:*e-def graph-of-def see-standard select-def*)
**finally show** *?thesis*
**by** *simp*
**qed**

**lemma** *expander-kl-chernoff-bound*:
**assumes** *measure* (*sample-pro S*) {*w*. *T w*} ≤ μ
**assumes** γ ≤ *1* μ + Λ ∗ (*1*−μ) ≤ γ μ ≤ *1*
**shows** *measure* (*sample-pro* (*expander-pro l* Λ *S*)) {*w*. *real* (*card* {*i* ∈ {..<*l*}. *T* (*w i*)}) ≥ γ∗*l*}
≤ *exp* (− *real l* ∗ *KL-div* γ (μ + Λ∗(*1*−μ))) (**is** *?L* ≤ *?R*)
**proof** (*cases measure* (*sample-pro S*) {*w*. *T w*} > *0*)
**case** *True*
**let** *?w* = *pmf-of-multiset* (*walks* (*graph-of e*) *l*)
**define** *V* **where** *V* = {*v*∈ *verts* (*graph-of e*). *T* (*pro-select S v*)}
**define** ν **where** ν = *measure* (*sample-pro S*) {*w*. *T w*}

**have** ν*-gt-0*: ν > *0* **unfolding** ν*-def* **using** *True* **by** *simp*
**have** ν*-le-1*: ν ≤ *1* **unfolding** ν*-def* **by** *simp*
**have** ν*-le-*μ: ν ≤ μ **unfolding** ν*-def* **using** *assms*(*1*) **by** *simp*

**have** *0*: *card* {*i* ∈ {..<*l*}. *T* (*pro-select S* (*w* ! *i mod pro-size S*))} = *card* {*i* ∈ {..<*l*}. *w* ! *i* ∈ *V*}
**if** *w* ∈ *set-pmf* (*pmf-of-multiset* (*walks* (*graph-of e*) *l*)) **for** *w*
**proof** −
**have** *a0*: *w* ∈# *walks* (*graph-of e*) *l* **using** *that E.walks-nonempty* **by** *simp*
**have** *a1*:*w* ! *i* ∈ *verts* (*graph-of e*) **if** *i* < *l* **for** *i*
**using** *that E.set-walks-3*[*OF a0*] **by** *auto*
**moreover have** *w* ! *i mod pro-size S* = *w* ! *i* **if** *i* < *l* **for** *i*
**using** *a1*[*OF that*] *see-standard*(*2*) *e-def* **by** (*simp add*:*graph-of-def*)
**ultimately show** *?thesis*

**unfolding** *V-def*
**by** (*intro arg-cong*[**where** *f*=*card*] *restr-Collect-cong*) *auto*
**qed**

**have** *1*:$E.\Lambda_a \le \Lambda$
**using** *see-standard*(*1*) **unfolding** *is-expander-def e-def* **by** *simp*

**have** *2*: $V \subseteq$ *verts* (*graph-of e*)
**unfolding** *V-def* **by** *simp*

**have** $\nu$ = *measure* (*pmf-of-set* {..<*pro-size S*}) ({*v*. *T* (*pro-select S v*)})
**unfolding** $\nu$*-def sample-pro-alt* **by** *simp*
**also have** ... = *real* (*card* ({*v*∈{..<*pro-size S*}. *T* (*pro-select S v*)})) / *real* (*pro-size S*)
**using** *pro-size-gt-0* **by** (*subst measure-pmf-of-set*) (*auto simp add*:*Int-def*)
**also have** ... = *real* (*card V*) / *card* (*verts* (*graph-of e*))
**unfolding** *V-def graph-of-def e-def* **using** *see-standard* **by** (*simp add*:*Int-commute*)
**finally have** $\nu$*-eq*: $\nu$ = *real* (*card V*) / *card* (*verts* (*graph-of e*))
**by** *simp*

**have** *3*: $0 < \nu + E.\Lambda_a * (1 - \nu)$
**using** $\nu$*-le-1* **by** (*intro add-pos-nonneg* $\nu$*-gt-0 mult-nonneg-nonneg* $E.\Lambda$*-ge-0*) *auto*

**have** $\nu + E.\Lambda_a * (1 - \nu) = \nu * (1 - E.\Lambda_a) + E.\Lambda_a$ **by** (*simp add*:*algebra-simps*)
**also have** ... $\le \mu * (1- E.\Lambda_a) + E.\Lambda_a$ **using** $E.\Lambda$*-le-1*
**by** (*intro add-mono mult-right-mono* $\nu$*-le-*$\mu$) *auto*
**also have** ... $= \mu + E.\Lambda_a * (1 - \mu)$ **by** (*simp add*:*algebra-simps*)
**also have** ... $\le \mu + \Lambda * (1 - \mu)$ **using** *assms*(*4*) **by** (*intro add-mono mult-right-mono 1*) *auto*
**finally have** *4*: $\nu + E.\Lambda_a * (1 - \nu) \le \mu + \Lambda * (1 - \mu)$ **by** *simp*

**have** *5*: $\nu + E.\Lambda_a*(1-\nu) \le \gamma$ **using** *4 assms*(*3*) **by** *simp*

**have** *?L* = *measure ?w* {*y*. $\gamma$ ∗ *real l* ≤ *real* (*card* {*i* ∈ {..<*l*}. *T* (*pro-select S* (*y* ! *i mod pro-size S*))})}
**unfolding** *sample-pro-expander-walks* **by** *simp*
**also have** ... = *measure ?w* {*y*. $\gamma$ ∗ *real l* ≤ *real* (*card* {*i* ∈ {..<*l*}. *y* ! *i* ∈ *V*})}
**using** *0* **by** (*intro measure-pmf-cong*) (*simp*)
**also have** ... ≤ *exp* (− *real l* ∗ *KL-div* $\gamma$ ($\nu + E.\Lambda_a*(1-\nu)$) )
**using** *assms*(*2*) *3 5* **unfolding** $\nu$*-eq* **by** (*intro E.kl-chernoff-property l-gt-0 2*) *auto*
**also have** ... ≤ *exp* (− *real l* ∗ *KL-div* $\gamma$ ($\mu + \Lambda*(1-\mu)$))
**using** *l-gt-0* **by** (*intro iffD2*[*OF exp-le-cancel-iff*] *iffD2*[*OF mult-le-cancel-left-neg*]
*KL-div-mono-right*[*OF disjI2*] *conjI 3 4 assms*(*2*,*3*)) *auto*
**finally show** *?thesis* **by** *simp*
**next**
**case** *False*
**hence** *0*:*measure* (*sample-pro S*) {*w*. *T w*} = *0* **using** *zero-less-measure-iff* **by** *blast*
**hence** *1*:*T w* = *False* **if** *w* ∈ *pro-set S* **for** *w* **using** *that measure-pmf-posI* **by** *force*

**have** $\mu + \Lambda * (1-\mu) > 0$
**proof** (*cases* $\mu = 0$)
**case** *True* **then show** *?thesis* **using** $\Lambda$*-gt-0* **by** *auto*
**next**
**case** *False*
**then show** *?thesis* **using** *assms*(*1*,*4*) *0* $\Lambda$*-gt-0*
**by** (*intro add-pos-nonneg mult-nonneg-nonneg*) *simp-all*
**qed**
**hence** $\gamma > 0$ **using** *assms*(*3*) **by** *auto*
**hence** *2*:$\gamma$∗*real l* > *0* **using** *l-gt-0* **by** *simp*

**let** *?w = pmf-of-multiset (walks (graph-of e) l)*

**have** *?L = measure ?w {y. γ∗real l≤ card {i ∈ {..<l}. T (pro-select S (y ! i mod pro-size S))}}*
**unfolding** *sample-pro-expander-walks* **by** *simp*
**also have** ... *= 0* **using** *pro-select-in-set 2* **by** *(subst 1) auto*
**also have** ... *≤ ?R* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expander-chernoff-bound-one-sided*:
**assumes** *AE x in sample-pro S. f x ∈ {0,1::real}*
**assumes** *(∫ x. f x ∂sample-pro S) ≤ μ l > 0 γ ≥ 0*
**shows** *measure (expander-pro l Λ S) {w. (∑ i<l. f (w i))/l−μ≥γ+Λ} ≤ exp (− 2 ∗ real l ∗ γ^2)*
(**is** *?L ≤ ?R*)
**proof** −
**let** *?w = sample-pro (expander-pro l Λ S)*
**define** *T* **where** *T x = (f x=1)* **for** *x*

**have** *1*: *indicator {w. T w} x = f x* **if** *x ∈ pro-set S* **for** *x*
**proof** −
**have** *f x ∈ {0,1}* **using** *assms(1) that* **unfolding** *AE-measure-pmf-iff* **by** *simp*
**thus** *?thesis* **unfolding** *T-def* **by** *auto*
**qed**

**have** *measure S {w. T w} = (∫ x. indicator {w. T w} x ∂S)* **by** *simp*
**also have** ... *= (∫ x. f x ∂S)* **using** *1* **by** *(intro integral-cong-AE AE-pmfI) auto*
**also have** ... *≤ μ* **using** *assms(2)* **by** *simp*
**finally have** *0*: *measure S {w. T w} ≤ μ* **by** *simp*

**hence** *μ-ge-0*: *μ ≥ 0* **using** *measure-nonneg order.trans* **by** *blast*

**have** *cases*: *(γ=0 ⟹ p) ⟹ (γ+Λ+μ > 1 ⟹ p) ⟹ (γ+Λ+μ ≤ 1 ∧ γ > 0 ⟹ p) ⟹ p* **for** *p*
**using** *assms(4)* **by** *argo*

**have** *?L = measure ?w {w. (γ+Λ+μ)∗l ≤ (∑ i<l. f (w i))}*
**using** *assms(3)* **by** *(intro measure-pmf-cong) (auto simp:field-simps)*
**also have** ... *= measure ?w {w. (γ+Λ+μ)∗l ≤ card {i ∈ {..<l}. T (w i)}}*
**proof** *(rule measure-pmf-cong)*
**fix** *ω*
**assume** *ω ∈ pro-set (expander-pro l Λ S)*
**hence** *ω x ∈ pro-set S* **for** *x* **using** *expander-pro-range set-sample-pro* **by** *(metis image-iff)*
**hence** *(∑ i<l. f (ω i)) = (∑ i<l. indicator {w. T w} (ω i))* **using** *1* **by** *(intro sum.cong) auto*
**also have** ... *= card {i ∈ {..<l}. T (ω i)}* **unfolding** *indicator-def* **by** *(auto simp:Int-def)*
**finally have** *(∑ i<l. f (ω i)) = (card {i ∈ {..<l}. T (ω i)})* **by** *simp*
**thus** *(ω ∈ {w. (γ+Λ+μ)∗l ≤ (∑ i<l. f (w i))})=(ω ∈ {w. (γ+Λ+μ)∗l≤card {i ∈ {..<l}. T (w i)}})*
**by** *simp*
**qed**
**also have** ... *≤ ?R* (**is** *?L1 ≤ -*)
**proof** *(rule cases)*
**assume** *γ = 0* **thus** *?thesis* **by** *simp*
**next**
**assume** *a:γ + Λ + μ ≤ 1 ∧ 0 < γ*
**hence** *μ-lt-1*: *μ < 1* **using** *assms(4) Λ-gt-0* **by** *simp*
**hence** *μ-le-1*: *μ ≤ 1* **by** *simp*

**have** $\mu + \Lambda * (1 - \mu) \le \mu + \Lambda * 1$ **using** *μ-ge-0 Λ-gt-0* **by** (*intro add-mono mult-left-mono*) *auto*
**also have** ... $< \gamma+\Lambda+\mu$ **using** *assms(4) a* **by** *simp*
**finally have** *b*:$\mu + \Lambda * (1 - \mu) < \gamma + \Lambda + \mu$ **by** *simp*
**hence** $\mu + \Lambda * (1 - \mu) < 1$ **using** *a* **by** *simp*
**moreover have** $\mu + \Lambda * (1 - \mu) > 0$ **using** *μ-lt-1*
**by** (*intro add-nonneg-pos μ-ge-0 mult-pos-pos Λ-gt-0*) *simp*
**ultimately have** *c*: $\mu + \Lambda * (1 - \mu) \in \{0<..<1\}$ **by** *simp*
**have** *d*: $\gamma + \Lambda + \mu \in \{0..1\}$ **using** *a b c* **by** *simp*
**have** $?L1 \le exp\ (- real\ l * KL\text{-}div\ (\gamma+\Lambda+\mu)\ (\mu + \Lambda*(1-\mu)))$
**using** *a b* **by** (*intro expander-kl-chernoff-bound μ-le-1 0*) *auto*
**also have** ... $\le exp\ (- real\ l * (2 * ((\gamma+\Lambda+\mu) - (\mu + \Lambda*(1-\mu)))\hat{}2))$
**by** (*intro iffD2[OF exp-le-cancel-iff] mult-left-mono-neg KL-div-lower-bound c d*) *simp*
**also have** ... $\le exp\ (- real\ l * (2 * (\gamma\hat{}2)))$
**using** *assms(4) μ-lt-1 Λ-gt-0 μ-ge-0*
**by** (*intro iffD2[OF exp-le-cancel-iff] mult-left-mono-neg*[**where** *c=−real l*] *mult-left-mono power-mono*) *simp-all*
**also have** ... $= ?R$ **by** *simp*
**finally show** $?L1 \le ?R$ **by** *simp*
**next**
**assume** *a*:$1 < \gamma + \Lambda + \mu$
**have** $(\gamma+\Lambda+\mu)* real\ l > real\ (card\ \{i \in \{..<l\}.\ (x\ i)\})$ **for** *x*
**proof** −
**have** $real\ (card\ \{i \in \{..<l\}.\ (x\ i)\}) \le card\ \{..<l\}$ **by** (*intro of-nat-mono card-mono*) *auto*
**also have** ... $= real\ l$ **by** *simp*
**also have** ... $< (\gamma+\Lambda+\mu)* real\ l$ **using** *assms(3) a* **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**
**hence** $?L1 = 0$ **unfolding** *not-le[symmetric]* **by** *auto*
**also have** ... $\le ?R$ **by** *simp*
**finally show** $?L1 \le ?R$ **by** *simp*
**qed**
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expander-chernoff-bound*:
**assumes** *AE x in sample-pro S. f x* $\in \{0,1::real\}$ $l > 0$ $\gamma \ge 0$
**defines** $\mu \equiv (\int x.\ f\ x\ \partial sample\text{-}pro\ S)$
**shows** $measure\ (expander\text{-}pro\ l\ \Lambda\ S)\ \{w.\ |(\sum i<l.\ f\ (w\ i))/l-\mu| \ge \gamma+\Lambda\} \le 2*exp\ (-\ 2 * real\ l * \gamma\hat{}2)$
(**is** $?L \le ?R$)
**proof** −
**let** $?w = sample\text{-}pro\ (expander\text{-}pro\ l\ \Lambda\ S)$
**have** $?L \le measure\ ?w\ \{w.\ (\sum i<l.\ f\ (w\ i))/l-\mu \ge \gamma+\Lambda\} + measure\ ?w\ \{w.\ (\sum i<l.\ f\ (w\ i))/l-\mu \le -(\gamma+\Lambda)\}$
**by** (*intro pmf-add*) *auto*
**also have** ... $\le exp\ (-2*real\ l*\gamma\hat{}2) + measure\ ?w\ \{w.\ -((\sum i<l.\ f\ (w\ i))/l-\mu) \ge (\gamma+\Lambda)\}$
**using** *assms* **by** (*intro add-mono expander-chernoff-bound-one-sided*) (*auto simp:algebra-simps*)
**also have** ... $\le exp\ (-2*real\ l*\gamma\hat{}2) + measure\ ?w\ \{w.\ ((\sum i<l.\ 1-f\ (w\ i))/l-(1-\mu)) \ge (\gamma+\Lambda)\}$
**using** *assms(2)* **by** (*auto simp: sum-subtractf field-simps*)
**also have** ... $\le exp\ (-2*real\ l*\gamma\hat{}2) + exp\ (-2*real\ l*\gamma\hat{}2)$
**using** *assms* **by** (*intro add-mono expander-chernoff-bound-one-sided*) *auto*
**also have** ... $= ?R$ **by** *simp*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *expander-pro-size*:
$pro\text{-}size\ (expander\text{-}pro\ l\ \Lambda\ S) = pro\text{-}size\ S * (16\ \hat{}\ ((l-1) * nat\ \lceil ln\ \Lambda\ /\ ln\ (19\ /\ 20)\rceil))$

(**is** *?L* = *?R*)
**proof** −
**have** *?L* = *see-size e* ∗ *see-degree e* ^ (*l* − *1*)
**unfolding** *expander-sample-size* **by** *simp*
**also have** ... = *pro-size S* ∗ (*16* ^ *nat* ⌈*ln* Λ / *ln* (*19* / *20*)⌉) ^ (*l* − *1*)
**using** *see-standard* **unfolding** *e-def* **by** *simp*
**also have** ... = *pro-size S* ∗ (*16* ^ ((*l*−*1*) ∗ *nat* ⌈*ln* Λ / *ln* (*19* / *20*)⌉))
**unfolding** *power-mult*[*symmetric*] **by** (*simp add*:*ac-simps*)
**finally show** *?thesis*
**by** *simp*
**qed**

**end**

**open-bundle** *expander-pseudorandom-object-syntax*
**begin**
**notation** *expander-pro* (‹ℰ›)
**end**

**unbundle** *no intro-cong-syntax*

**end**

# References


[1] J. Divasón, O. Kunar, R. Thiemann, and A. Yamada. Perron-frobenius theorem for spectral radius analysis. *Archive of Formal Proofs*, May 2016. https://isa-afp.org/entries/Perron_Frobenius.html, Formal proof development.

[2] M. Echenim. Simultaneous diagonalization of pairwise commuting hermitian matrices. *Archive of Formal Proofs*, July 2022. https://isa-afp.org/entries/Commuting_Hermitian.html, Formal proof development.

[3] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[4] S. Hoory and N. Linial. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[5] R. Impagliazzo and V. Kabanets. Constructive proofs of concentration bounds. In M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 617–631, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4), 1987.

[7] O. Kuncar and A. Popescu. From types to sets by local type definition in higher-order logic. *Journal of Automated Reasoning*, 62:237 – 260, 2016.

[8] G. A. Margulis. Explicit construction of a concentrator. *Probl. Peredachi Inf.*, 9(4):71–80, 1973.

[9] J. Murtagh, O. Reingold, A. Sidford, and S. Vadhan. Deterministic Approximation of Random Walks in Small Space. In D. Achlioptas and L. A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:22, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] L. Noschinski. Graph theory. *Archive of Formal Proofs*, April 2013. https://isa-afp.org/entries/Graph_Theory.html, Formal proof development.

[11] S. P. Vadhan. Pseudorandomness. *Foundations and Trends(R) in Theoretical Computer Science*, 7(13):1–336, 2012.