

# Expander Graphs

Emin Karayel

June 14, 2026

## 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

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Preliminary Results</b>	<b>2</b>
2.1	Constructive Chernoff Bound . . . . .	2
2.2	Congruence Method . . . . .	12
2.3	Multisets . . . . .	13
<b>3</b>	<b>Definitions</b>	<b>17</b>
<b>4</b>	<b>Setup for Types to Sets</b>	<b>35</b>
<b>5</b>	<b>Algebra-only Theorems</b>	<b>37</b>
<b>6</b>	<b>Spectral Theory</b>	<b>54</b>
<b>7</b>	<b>Cheeger Inequality</b>	<b>80</b>
<b>8</b>	<b>Margulis Gabber Galil Construction</b>	<b>92</b>
<b>9</b>	<b>Random Walks</b>	<b>114</b>
<b>10</b>	<b>Graph Powers</b>	<b>124</b>
<b>11</b>	<b>Strongly Explicit Expander Graphs</b>	<b>134</b>
<b>12</b>	<b>Expander Walks as Pseudorandom Objects</b>	<b>153</b>

# 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<sup>1</sup> with increasing sizes, but constant 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 construction 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 remains 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 algebraic 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-existing 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 Kuncăr 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*

*Negative-Association.Negative-Association-Util*

**begin**

**lemma** *powr-mono-rev*:

---

<sup>1</sup>A graph is regular if every node has the same degree.

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

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

**lemma** *integrable-pmf-iff-bounded*:  
**fixes**  $f :: 'a \Rightarrow \text{real}$   
**assumes**  $\bigwedge x. x \in \text{set-pmf } p \implies \text{abs } (f x) \leq C$   
**shows** *integrable* (*measure-pmf*  $p$ )  $f$

**proof** –  
**obtain**  $x$  **where**  $x \in \text{set-pmf } p$   
**using** *set-pmf-not-empty* **by** *fast*  
**hence**  $C \geq 0$  **using** *assms(1)* **by** *fastforce*  
**hence**  $(\int^+ x. \text{ennreal } (\text{abs } (f x)) \partial \text{measure-pmf } p) \leq (\int^+ x. C \partial \text{measure-pmf } p)$   
**using** *assms* *ennreal-le-iff*  
**by** (*intro nn-integral-mono-AE AE-pmfI*) *auto*  
**also have**  $\dots = C$   
**by** *simp*  
**also have**  $\dots < \text{Orderings.top}$   
**by** *simp*  
**finally have**  $(\int^+ x. \text{ennreal } (\text{abs } (f x)) \partial \text{measure-pmf } p) < \text{Orderings.top}$  **by** *simp*  
**thus** *?thesis*  
**by** (*intro iffD2[OF integrable-iff-bounded]*) *auto*  
**qed**

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

**proof** –  
**have**  $a: \text{integrable } (\text{measure-pmf } A) (\lambda x. \text{measure-pmf.prob } B \{b. (x, b) \in S\})$   
**by** (*intro integrable-pmf-iff-bounded[where C=1]*) *simp*  
  
**have**  $?L = (\int^+ x. \text{indicator } S x \partial (\text{measure-pmf } (\text{pair-pmf } A B)))$   
**by** (*simp add: measure-pmf.emmeasure-eq-measure*)  
**also have**  $\dots = (\int^+ x. (\int^+ y. \text{indicator } S (x,y) \partial B) \partial A)$   
**by** (*simp add: nn-integral-pair-pmf'*)  
**also have**  $\dots = (\int^+ x. (\int^+ y. \text{indicator } \{b. (x,b) \in S\} y \partial B) \partial A)$   
**by** (*simp add: indicator-def*)  
**also have**  $\dots = (\int^+ x. (\text{measure-pmf.prob } B \{b. (x,b) \in S\}) \partial A)$   
**by** (*simp add: measure-pmf.emmeasure-eq-measure*)  
**also have**  $\dots = ?R$   
**using**  $a$   
**by** (*subst nn-integral-eq-integral*) *auto*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *split-pair-pmf-2*:

$measure(pair\text{-}pmf\ A\ B)\ S = integral^L\ B\ (\lambda a. measure\text{-}pmf.\text{prob}\ A\ \{b. (b,a) \in S\})$   
(is ?L = ?R)

**proof** –

**have** ?L =  $measure\ (pair\text{-}pmf\ B\ A)\ \{\omega. (snd\ \omega, fst\ \omega) \in S\}$   
**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**

**theorem** *impagliazzo-kabanets-pmf*:

**fixes**  $Y :: nat \Rightarrow 'a \Rightarrow bool$

**fixes**  $p :: 'a\ pmf$

**assumes**  $n > 0$

**assumes**  $\bigwedge i. i \in \{..<n\} \implies \delta\ i \in \{0..1\}$

**assumes**  $\bigwedge S. S \subseteq \{..<n\} \implies measure\ p\ \{\omega. (\forall i \in S. Y\ i\ \omega)\} \leq (\prod i \in S. \delta\ i)$

**defines**  $\delta\text{-}avg \equiv (\sum i \in \{..<n\}. \delta\ i) / n$

**assumes**  $\gamma \in \{\delta\text{-}avg..1\}$

**assumes**  $\delta\text{-}avg > 0$

**shows**  $measure\ p\ \{\omega. real\ (card\ \{i \in \{..<n\}. Y\ i\ \omega\}) \geq \gamma * n\} \leq exp\ (-real\ n * KL\text{-}div\ \gamma\ \delta\text{-}avg)$

(is ?L ≤ ?R)

**proof** –

**let** ?n =  $real\ n$

**define**  $q :: real$  **where**  $q = (if\ \gamma = 1\ then\ 1\ else\ (\gamma - \delta\text{-}avg) / (\gamma * (1 - \delta\text{-}avg)))$

**define**  $g$  **where**  $g\ \omega = card\ \{i. i < n \wedge \neg Y\ i\ \omega\}$  **for**  $\omega$

**let** ?E =  $(\lambda\omega. real\ (card\ \{i. i < n \wedge Y\ i\ \omega\}) \geq \gamma * n)$

**let** ?Ξ =  $prod\text{-}pmf\ \{..<n\}\ (\lambda\text{-}. bernoulli\text{-}pmf\ q)$

**have**  $q\text{-}range: q \in \{0..1\}$

**proof** (*cases*  $\gamma < 1$ )

**case** *True*

**then show** ?thesis

**using** *assms(5,6)*

**unfolding**  $q\text{-}def$  **by** (*auto intro!: divide-nonneg-pos simp add:algebra-simps*)

**next**

**case** *False*

**hence**  $\gamma = 1$  **using** *assms(5)* **by** *simp*

**then show** ?thesis **unfolding**  $q\text{-}def$  **by** *simp*

**qed**

**have** *abs-pos-le-1I*:  $abs\ x \leq 1$  **if**  $x \geq 0$   $x \leq 1$  **for**  $x :: real$

**using** *that* **by** *auto*

**have**  $\gamma\text{-}n\text{-}nonneg: \gamma * ?n \geq 0$

**using** *assms(1,5,6)* **by** *simp*

**define**  $r$  **where**  $r = n - nat\ \lceil \gamma * n \rceil$

**have**  $2:(1-q) \wedge r \leq (1-q) \wedge g\ \omega$  **if** ?E  $\omega$  **for**  $\omega$

**proof** –

**have**  $g\ \omega = card\ (\{i. i < n\} - \{i. i < n \wedge Y\ i\ \omega\})$

**unfolding**  $g\text{-}def$  **by** (*intro arg-cong[where f=λx. card x]*) *auto*

**also have** ... =  $card\ \{i. i < n\} - card\ \{i. i < n \wedge Y\ i\ \omega\}$

**by** (*subst card-Diff-subset, auto*)

**also have** ... ≤  $card\ \{i. i < n\} - nat\ \lceil \gamma * n \rceil$

**using** *that*  $\gamma\text{-}n\text{-}nonneg$  **by** (*intro diff-le-mono2*) *simp*

**also have** ... =  $r$

**unfolding**  $r$ -def **by** *simp*  
**finally have**  $g \omega \leq r$  **by** *simp*  
**thus**  $(1-q) \wedge r \leq (1-q) \wedge (g \omega)$   
**using**  $q$ -range **by** (*intro power-decreasing*) *auto*  
**qed**

**have**  $\gamma$ -gt-0:  $\gamma > 0$   
**using** *assms(5,6)* **by** *simp*

**have**  $q$ -lt-1:  $q < 1$  **if**  $\gamma < 1$   
**proof** –  
**have**  $\delta$ -avg  $< 1$  **using** *assms(5)* **that** **by** *simp*  
**hence**  $(\gamma - \delta\text{-avg}) / (\gamma * (1 - \delta\text{-avg})) < 1$   
**using**  $\gamma$ -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:  $(\delta\text{-avg} * q + (1-q)) / (1-q) \text{ powr } (1-\gamma) = \exp(-KL\text{-div } \gamma \delta\text{-avg})$  (**is** ?L1 = ?R1)  
**if**  $\gamma < 1$

**proof** –  
**have**  $\delta$ -avg-range:  $\delta\text{-avg} \in \{0 < .. < 1\}$   
**using** *that assms(5,6)* **by** *simp*

**have** ?L1 =  $(1 - (1-\delta\text{-avg}) * q) / (1-q) \text{ powr } (1-\gamma)$   
**by** (*simp add:algebra-simps*)  
**also have** ... =  $(1 - (\gamma - \delta\text{-avg}) / \gamma) / (1-q) \text{ powr } (1-\gamma)$   
**unfolding**  $q$ -def **using** *that*  $\gamma$ -gt-0  $\delta$ -avg-range **by** *simp*  
**also have** ... =  $(\delta\text{-avg} / \gamma) / (1-q) \text{ powr } (1-\gamma)$   
**using**  $\gamma$ -gt-0 **by** (*simp add:divide-simps*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) * (1/(1-q)) \text{ powr } (1-\gamma)$   
**using**  $q$ -lt-1 [*OF that*] **by** (*subst powr-divide, simp-all*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) * (1/((\gamma * (1-\delta\text{-avg}) - (\gamma - \delta\text{-avg})) / (\gamma * (1-\delta\text{-avg})))) \text{ powr } (1-\gamma)$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **unfolding**  $q$ -def **by** (*simp add:divide-simps*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) * ((\gamma / \delta\text{-avg}) * ((1-\delta\text{-avg}) / (1-\gamma))) \text{ powr } (1-\gamma)$   
**by** (*simp add:algebra-simps*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) * (\gamma / \delta\text{-avg}) \text{ powr } (1-\gamma) * ((1-\delta\text{-avg}) / (1-\gamma)) \text{ powr } (1-\gamma)$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **by** (*subst powr-mult, auto*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) \text{ powr } 1 * (\delta\text{-avg} / \gamma) \text{ powr } -(1-\gamma) * ((1-\delta\text{-avg}) / (1-\gamma)) \text{ powr } (1-\gamma)$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **unfolding** *powr-minus-divide* **by** (*simp add:powr-divide*)  
**also have** ... =  $(\delta\text{-avg} / \gamma) \text{ powr } \gamma * ((1-\delta\text{-avg}) / (1-\gamma)) \text{ powr } (1-\gamma)$   
**by** (*subst powr-add[symmetric] simp*)  
**also have** ... =  $\exp(\ln((\delta\text{-avg} / \gamma) \text{ powr } \gamma * ((1-\delta\text{-avg}) / (1-\gamma)) \text{ powr } (1-\gamma)))$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **by** (*intro exp-ln[symmetric] mult-pos-pos*) *auto*  
**also have** ... =  $\exp((\ln((\delta\text{-avg} / \gamma) \text{ powr } \gamma) + \ln(((1 - \delta\text{-avg}) / (1 - \gamma)) \text{ powr } (1-\gamma))))$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **by** (*subst ln-mult*) *auto*  
**also have** ... =  $\exp((\gamma * \ln(\delta\text{-avg} / \gamma) + (1 - \gamma) * \ln(((1 - \delta\text{-avg}) / (1 - \gamma))))$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **by** (*simp add:algebra-simps*)  
**also have** ... =  $\exp(-(\gamma * \ln(\gamma / \delta\text{-avg}) + (1 - \gamma) * \ln(((1 - \gamma) / (1 - \delta\text{-avg}))))$   
**using**  $\gamma$ -gt-0  $\delta$ -avg-range **that** **by** (*simp add:ln-div algebra-simps*)  
**also have** ... = ?R1 **using** *assms(5,6)* **by** (*subst KL-div-eq'*) *force*  
**finally show** ?thesis **by** *simp*  
**qed**

**have** 3:  $(\delta\text{-avg} * q + (1-q)) \wedge n / (1-q) \wedge r \leq \exp(-?n * KL\text{-div } \gamma \delta\text{-avg})$  (**is** ?L1  $\leq$  ?R1)  
**proof** (*cases*  $\gamma < 1$ )

```

case True
have  $\gamma * \text{real } n \leq 1 * \text{real } n$ 
  using True by (intro mult-right-mono) auto
hence  $r = \text{real } n - \text{real } (\text{nat } \lceil \gamma * \text{real } n \rceil)$ 
  unfolding r-def by (subst of-nat-diff) auto
also have  $\dots = \text{real } n - \lceil \gamma * \text{real } n \rceil$ 
  using  $\gamma$ -n-nonneg by (subst of-nat-nat, auto)
also have  $\dots \leq ?n - \gamma * ?n$ 
  by (intro diff-mono) auto
also have  $\dots = (1-\gamma) * ?n$  by (simp add: algebra-simps)
finally have r-bound:  $r \leq (1-\gamma) * n$  by simp

have ?L1 =  $(\delta\text{-avg} * q + (1-q)) ^ n / (1-q) \text{ pow } r$ 
  using q-lt-1[OF True] assms(1) by (simp add: powr-realpow)
also have  $\dots = (\delta\text{-avg} * q + (1-q)) \text{ pow } n / (1-q) \text{ pow } r$ 
  using q-lt-1[OF True] assms(6) q-range
  by (subst powr-realpow[symmetric], auto intro!: add-nonneg-pos)
also have  $\dots \leq (\delta\text{-avg} * q + (1-q)) \text{ pow } n / (1-q) \text{ pow } ((1-\gamma) * n)$ 
  using q-range q-lt-1[OF True] by (intro divide-left-mono powr-mono-rev r-bound) auto
also have  $\dots = (\delta\text{-avg} * q + (1-q)) \text{ pow } n / ((1-q) \text{ pow } (1-\gamma)) \text{ pow } n$ 
  unfolding powr-powr by simp
also have  $\dots = ((\delta\text{-avg} * q + (1-q)) / (1-q) \text{ pow } (1-\gamma)) \text{ pow } n$ 
  using assms(6) q-range by (subst powr-divide) auto
also have  $\dots = \exp(-KL\text{-div } \gamma \delta\text{-avg}) \text{ pow } \text{real } n$ 
  unfolding 5[OF True] by simp
also have  $\dots = ?R1$ 
  unfolding exp-powr by simp
finally show ?thesis by simp
next
case False
hence  $\gamma\text{-eq-1}: \gamma=1$  using assms(5) by simp
have ?L1 =  $\delta\text{-avg} ^ n$ 
  using  $\gamma\text{-eq-1}$  r-def q-def by simp
also have  $\dots = \exp(-KL\text{-div } 1 \delta\text{-avg}) ^ n$ 
  using assms(5,6) by (subst KL-div-eq') (simp-all add: ln-div)
also have  $\dots = ?R1$ 
  using  $\gamma\text{-eq-1}$  by (simp add: powr-realpow[symmetric] exp-powr)
finally show ?thesis by simp
qed

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

have  $(1-q) ^ r * ?L = (\int \omega. \text{indicator } \{\omega. ?E \omega\} \omega * (1-q) ^ r \partial p)$ 
  by simp
also have  $\dots \leq (\int \omega. \text{indicator } \{\omega. ?E \omega\} \omega * (1-q) ^ g \omega \partial p)$ 
  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  $\dots = (\int \omega. \text{indicator } \{\omega. ?E \omega\} \omega * (\prod i \in \{i. i < n \wedge \neg Y i \omega\}. (1-q)) \partial p)$ 
  unfolding g-def using q-range

```

by (intro integral-cong-AE AE-pmfI, simp-all add:powr-realpow)  
 also have ... = (∫ ω. indicator {ω. ?E ω} ω \* measure ?Ξ ({j. j < n ∧ ¬Y j ω} → {False})  
 ∂p)  
 using q-range by (subst prob-prod-pmf') (auto simp add:measure-pmf-single)  
 also have ... = (∫ ω. measure ?Ξ {ξ. ?E ω ∧ (∀ i ∈ {j. j < n ∧ ¬Y j ω}. ¬ξ i)} ∂p)  
 by (intro integral-cong-AE AE-pmfI, simp-all add:Pi-def split:split-indicator)  
 also have ... = (∫ ω. measure ?Ξ {ξ. ?E ω ∧ (∀ i ∈ {..<n}. ξ i → Y i ω)} ∂p)  
 by (intro integral-cong-AE AE-pmfI measure-eq-AE) auto  
 also have ... = measure (pair-pmf p ?Ξ) {φ. ?E (fst φ) ∧ (∀ i ∈ {..<n}. snd φ i → Y i (fst φ))}  
 unfolding split-pair-pmf by simp  
 also have ... ≤ measure (pair-pmf p ?Ξ) {φ. (∀ i ∈ {j. j < n ∧ snd φ j}. Y i (fst φ))}  
 by (intro pmf-mono, auto)  
 also have ... = (∫ ξ. measure p {ω. ∀ i ∈ {j. j < n ∧ ξ j}. Y i ω} ∂ ?Ξ)  
 unfolding split-pair-pmf-2 by simp  
 also have ... ≤ (∫ a. (∏ i ∈ {j. j < n ∧ a j}. δ i) ∂ ?Ξ)  
 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 ... = (∫ a. (∏ i ∈ {..<n}. δ i ^ of-bool(a i)) ∂ ?Ξ)  
 unfolding of-bool-def by (intro integral-cong-AE AE-pmfI)  
 (auto simp add:if-distrib prod.If-cases Int-def)  
 also have ... = (∏ i < n. (∫ a. (δ i ^ of-bool a) ∂(bernoulli-pmf q)))  
 using assms(2) by (intro expectation-prod-Pi-pmf integrable-pmf-iff-bounded[where C=1])  
 auto  
 also have ... = (∏ i < n. δ i \* q + (1-q))  
 using q-range by simp  
 also have ... = (root (card {..<n}) (∏ i < n. δ i \* q + (1-q))) ^ (card {..<n})  
 using assms(1,2) q-range by (intro real-root-pow-pos2[symmetric] prod-nonneg) auto  
 also have ... ≤ ((∑ i < n. δ i \* q + (1-q)) / card {..<n}) ^ (card {..<n})  
 using assms(1,2) q-range by (intro power-mono arithmetic-geometric-mean)  
 (auto intro: prod-nonneg)  
 also have ... = ((∑ i < n. δ i \* q) / n + (1-q)) ^ n  
 using assms(1) by (simp add:sum.distrib divide-simps mult.commute)  
 also have ... = (δ-avg \* q + (1-q)) ^ n  
 unfolding δ-avg-def by (simp add: sum-distrib-right[symmetric])  
 finally have (1-q) ^ r \* ?L ≤ (δ-avg \* q + (1-q)) ^ n by simp  
 hence ?L ≤ (δ-avg \* q + (1-q)) ^ n / (1-q) ^ r  
 using 4 by (subst pos-le-divide-eq) (auto simp add:algebra-simps)  
 also have ... ≤ ?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 ⇒ 'b  
 assumes rv: random-variable discrete f  
 assumes countable (f ' space M)  
 shows distr M discrete f ∈ {M. prob-space M ∧ sets M = UNIV ∧ (AE x in M. measure M {x} ≠ 0)}

**proof** –

**define** N where N = {x ∈ space M. ¬ prob (f - ' {f x} ∩ space M) ≠ 0}

**define** I where I = {z ∈ (f ' space M). prob (f - ' {z} ∩ space M) = 0}

**have** countable-I: countable I

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

**have** *disj*: disjoint-family-on  $(\lambda y. f - \{y\} \cap \text{space } M) I$   
**unfolding** *disjoint-family-on-def* **by** *auto*

**have** *N-alt-def*:  $N = (\bigcup y \in I. f - \{y\} \cap \text{space } M)$   
**unfolding** *N-def I-def* **by** (*auto simp add:set-eq-iff*)

**have** *emeasure M N* =  $\int^+ y. \text{emeasure } M (f - \{y\} \cap \text{space } M) \partial \text{count-space } I$   
**using** *rv countable-I* **unfolding** *N-alt-def*  
**by** (*subst emeasure-UN-countable*) (*auto simp add:disjoint-family-on-def*)

**also have**  $\dots = \int^+ y. 0 \partial \text{count-space } I$   
**unfolding** *I-def* **using** *emeasure-eq-measure ennreal-0*  
**by** (*intro nn-integral-cong*) *auto*

**also have**  $\dots = 0$  **by** *simp*

**finally have**  $0:\text{emeasure } M N = 0$  **by** *simp*

**have**  $1:N \in \text{events}$   
**unfolding** *N-alt-def* **using** *rv*  
**by** (*intro sets.countable-UN'' countable-I*) *simp*

**have**  $AE x \text{ in } M. \text{prob } (f - \{f x\} \cap \text{space } M) \neq 0$   
**using**  $0 1$  **by** (*subst AE-iff-measurable[OF - N-def[symmetric]]*)

**hence**  $AE x \text{ in } M. \text{measure } (\text{distr } M \text{ discrete } f) \{f x\} \neq 0$   
**by** (*subst measure-distr[OF rv], auto*)

**hence**  $AE x \text{ in } \text{distr } M \text{ discrete } f. \text{measure } (\text{distr } M \text{ discrete } f) \{x\} \neq 0$   
**by** (*subst AE-distr-iff[OF rv], auto*)

**thus** *?thesis*  
**using** *prob-space-distr rv* **by** *auto*

**qed**

**lemma** *singletons-image-eq*:  
 $(\lambda x. \{x\}) ' T \subseteq \text{Pow } T$   
**by** *auto*

**theorem** (**in** *prob-space*) *impagliazzo-kabanets*:  
**fixes**  $Y :: \text{nat} \Rightarrow 'a \Rightarrow \text{bool}$   
**assumes**  $n > 0$   
**assumes**  $\bigwedge i. i \in \{..<n\} \implies \text{random-variable discrete } (Y i)$   
**assumes**  $\bigwedge i. i \in \{..<n\} \implies \delta i \in \{0..1\}$   
**assumes**  $\bigwedge S. S \subseteq \{..<n\} \implies \mathcal{P}(\omega \text{ in } M. (\forall i \in S. Y i \omega)) \leq (\prod i \in S. \delta i)$   
**defines**  $\delta\text{-avg} \equiv (\sum i \in \{..<n\}. \delta i) / n$   
**assumes**  $\gamma \in \{\delta\text{-avg}..1\} \delta\text{-avg} > 0$   
**shows**  $\mathcal{P}(\omega \text{ in } M. \text{real } (\text{card } \{i \in \{..<n\}. Y i \omega\}) \geq \gamma * n) \leq \exp (-\text{real } n * \text{KL-div } \gamma \delta\text{-avg})$   
**(is ?L ≤ ?R)**

**proof** –  
**define** *f* **where**  $f = (\lambda \omega i. \text{if } i < n \text{ then } Y i \omega \text{ else } \text{False})$   
**define** *g* **where**  $g = (\lambda \omega i. \text{if } i < n \text{ then } \omega i \text{ else } \text{False})$   
**define** *T* **where**  $T = \{\omega. (\forall i. \omega i \longrightarrow i < n)\}$

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

**have** *f-range*:  $f \in \text{space } M \rightarrow T$   
**unfolding** *T-def f-def* **by** *simp*

**have**  $T = \text{PiE-dflt } \{..<n\} \text{False } (\lambda -. \text{UNIV})$   
**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**  $f \text{ ' space } M \subseteq T$   
**using**  $f\text{-range}$  **by**  $auto$   
**ultimately have**  $countable\text{-}f: countable (f \text{ ' space } M)$   
**using**  $countable\text{-}subset$  **by**  $auto$

**have**  $f \text{ - ' } y \cap \text{ space } M \in \text{ events}$  **if**  $t: y \in (\lambda x. \{x\}) \text{ ' } T$  **for**  $y$   
**proof** –  
**obtain**  $t$  **where**  $y = \{t\}$  **and**  $t\text{-range}: t \in T$  **using**  $t$  **by**  $auto$   
**hence**  $f \text{ - ' } y \cap \text{ space } M = \{\omega \in \text{ space } M. f \ \omega = t\}$   
**by**  $(auto \text{ simp } add: \text{vimage-def})$   
**also have**  $\dots = \{\omega \in \text{ space } M. (\forall i < n. Y \ i \ \omega = t \ i)\}$   
**using**  $t\text{-range}$  **unfolding**  $f\text{-def}$   $T\text{-def}$  **by**  $auto$   
**also have**  $\dots = (\bigcap i \in \{..<n\}. \{\omega \in \text{ space } M. Y \ i \ \omega = t \ i\})$   
**using**  $assms(1)$  **by**  $auto$   
**also have**  $\dots \in \text{ events}$   
**using**  $assms(1,2)$   
**by**  $(intro \ sets.countable\text{-}INT)$   $auto$   
**finally show**  $?thesis$  **by**  $simp$   
**qed**

**hence**  $random\text{-}variable (count\text{-}space \ T) \ f$   
**using**  $sigma\text{-sets}\text{-}singletons[OF \ countable\text{-}T]$   $singletons\text{-}image\text{-}eq \ f\text{-range}$   
**by**  $(intro \ measurable\text{-}sigma\text{-sets[where \ \Omega=T \ \text{and} \ A= (\lambda x. \{x\}) \text{ ' } T])$   $simp\text{-}all$   
**moreover have**  $g \in \text{ measurable } discrete (count\text{-}space \ T)$   
**unfolding**  $g\text{-def}$   $T\text{-def}$  **by**  $simp$   
**ultimately have**  $random\text{-}variable \ discrete (g \circ f)$   
**by**  $simp$   
**hence**  $rv: random\text{-}variable \ discrete \ f$   
**unfolding**  $g\text{-idem}$  **by**  $simp$

**define**  $M' :: (nat \Rightarrow bool) \ measure$   
**where**  $M' = \text{distr } M \ \text{discrete } f$

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

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

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

**have**  $?L = \text{measure } M' \ \{\omega. \text{real } (\text{card } \{i. i < n \wedge \omega \ i\}) \geq \gamma * n\}$   
**unfolding**  $M'\text{-def}$  **by**  $(subst \ \text{measure-distr}[OF \ rv])$   
 $(auto \ \text{simp } add: f\text{-def} \ algebra\text{-sims} \ Int\text{-commute} \ Int\text{-def} \ b)$

**also have** ... = *measure-pmf.prob*  $\Omega \{ \omega. \text{real } (\text{card } \{i \in \{..<n\}. \omega \ i\}) \geq \gamma * n \}$   
**unfolding**  $\Omega$ -def a **by** *simp*  
**also have** ...  $\leq ?R$   
**using** *assms(1,3,6,7) c* **unfolding**  $\delta$ -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**  $0 \leq p \leq q \leq q' < 1$   
**shows**  $KL\text{-div } p \ q - 2*(p-q)^2 \leq KL\text{-div } p \ q' - 2*(p-q')^2$

**proof** (*cases p = 0*)

**case** *True*

**define**  $f' :: \text{real} \Rightarrow \text{real}$  **where**  $f' = (\lambda x. 1/(1-x) - 4 * x)$

**have** *deriv*:  $((\lambda q. \ln (1/(1-q)) - 2*q^2)$  *has-real-derivative*  $(f' x)$ ) (at  $x$ )  
**if**  $x \in \{q..q'\}$  **for**  $x$

**proof** –

**have**  $x \in \{0..<1\}$  **using** *assms that* **by** *auto*

**thus** *?thesis* **unfolding**  $f'$ -def **by** (*auto intro!*: *derivative-eq-intros*)

**qed**

**have** *deriv-nonneg*:  $f' x \geq 0$  **if**  $x \in \{q..q'\}$  **for**  $x$

**proof** –

**have**  $0 : x \in \{0..<1\}$  **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** ...  $\leq 1$  **by** *simp*

**finally have**  $4*x*(1-x) \leq 1$  **by** *simp*

**hence**  $1/(1-x) \geq 4*x$  **using**  $0$  **by** (*simp add: pos-le-divide-eq*)

**thus** *?thesis* **unfolding**  $f'$ -def **by** *auto*

**qed**

**have**  $\ln (1 / (1 - q)) - 2 * q^2 \leq \ln (1 / (1 - q')) - 2 * q'^2$

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

**thus** *?thesis* **using** *True assms* **by** (*subst (1 2) KL-div-eq'*) *simp-all*

**next**

**case** *False*

**hence** *p-gt-0*:  $p > 0$  **using** *assms* **by** *auto*

**define**  $f' :: \text{real} \Rightarrow \text{real}$  **where**  $f' = (\lambda x. (1-p)/(1-x) - p/x + 4 * (p-x))$

**let** *?KL-div* =  $(\lambda p \ q. p * \ln (p/q) + (1-p) * \ln ((1-p)/(1-q)))$

**have** *deriv*:  $((\lambda q. ?KL\text{-div } p \ q - 2*(p-q)^2)$  *has-real-derivative*  $(f' x)$ ) (at  $x$ ) **if**  $x \in \{q..q'\}$   
**for**  $x$

**proof** –

**have**  $0 < p / x \ 0 < (1 - p) / (1 - x)$  **using** *that assms p-gt-0* **by** *auto*

**thus** *?thesis* **unfolding**  $f'$ -def **by** (*auto intro!*: *derivative-eq-intros*)

**qed**

**have** *f'-part-nonneg*:  $(1/(x*(1-x)) - 4) \geq 0$  **if**  $x \in \{0<..<1\}$  **for**  $x :: \text{real}$

**proof** –

**have**  $4 * x * (1-x) = 1 - 4 * (x-1/2)^2$  **by** (*simp add:power2-eq-square algebra-simps*)

**also have** ...  $\leq 1$  **by** *simp*

**finally have**  $4 * x * (1-x) \leq 1$  **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  $\dots = (x-p)*(1/(x*(1-x)) - 4)$  **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  $\dots \geq 0$  **using** that *f'*-part-nonneg *assms* *p-gt-0* **by** (*intro mult-nonneg-nonneg*) *auto*  
 finally show *?thesis* **by** *simp*  
**qed**

have  $KL\text{-div } p \ q - 2*(p-q)^2 = ?KL\text{-div } p \ q - 2*(p-q)^2$   
**using** *assms* **by** (*subst KL-div-eq'*) *auto*  
 also have  $\dots \leq ?KL\text{-div } p \ q' - 2*(p-q')^2$  **using** *deriv deriv-nonneg*  
**by** (*intro DERIV-nonneg-imp-nondecreasing[OF assms(3)]*) *auto*  
 also have  $\dots = KL\text{-div } p \ q' - 2*(p-q')^2$   
**using** *assms* **by** (*subst KL-div-eq'*) *auto*  
 finally show *?thesis* **by** *simp*  
**qed**

**lemma** *KL-div-mono-right-aux-2*:  
**assumes**  $0 < q' \ q' \leq q \ q \leq p \ p \leq 1$   
**shows**  $KL\text{-div } p \ q - 2*(p-q)^2 \leq KL\text{-div } p \ q' - 2*(p-q')^2$   
**proof** –  
 have  $KL\text{-div } (1-p) \ (1-q) - 2*((1-p)-(1-q))^2 \leq KL\text{-div } (1-p) \ (1-q') - 2*((1-p)-(1-q'))^2$   
**using** *assms* **by** (*intro KL-div-mono-right-aux-1*) *auto*  
**thus** *?thesis* **using** *assms* **by** (*subst (1 2) KL-div-swap-gen*) (*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**  $KL\text{-div } p \ q - 2*(p-q)^2 \leq KL\text{-div } p \ q' - 2*(p-q')^2$   
**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**  $KL\text{-div } p \ q \leq KL\text{-div } p \ q'$  (**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**  $?L = (KL\text{-div } p \ q - 2*(p-q)^2) + 2 * (p-q)^2$  **by** *simp*  
**also have**  $\dots \leq (KL\text{-div } p \ q' - 2*(p-q')^2) + 2 * (p-q')^2$   
**by** (*intro add-mono KL-div-mono-right-aux* *assms mult-left-mono 0*) *auto*  
**also have**  $\dots = ?R$  **by** *simp*  
**finally show** *?thesis* **by** *simp*

qed

lemma *KL-div-lower-bound*:

assumes  $p \in \{0..1\}$   $q \in \{0 < .. < 1\}$

shows  $2 * (p - q)^2 \leq \text{KL-div } p \ q$

proof -

have  $0 \leq \text{KL-div } p \ p - 2 * (p - p)^2$  using *assms* by (*subst KL-div-eq'*) *auto*

also have  $\dots \leq \text{KL-div } p \ q - 2 * (p - q)^2$  using *assms* by (*intro KL-div-mono-right-ax*) *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* ::  $('a \Rightarrow 'b) \Rightarrow \text{cong-tag-type}$

where *cong-tag-1*  $x = \text{CongTag}$

definition *cong-tag-2* ::  $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow \text{cong-tag-type}$

where *cong-tag-2*  $x = \text{CongTag}$

definition *cong-tag-3* ::  $('a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd) \Rightarrow \text{cong-tag-type}$

where *cong-tag-3*  $x = \text{CongTag}$

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  $A :: \text{cong-tag-type list}$  uses *more* =

(*match*  $A$ ) in

*cong-tag-1*  $f \# h$  (*multi*) for  $f :: 'a \Rightarrow 'b$  and  $h$

$\Rightarrow \langle \text{intro-cong } h \ \text{more:more } \text{arg-cong}[\text{where } f=f] \rangle$

| *cong-tag-2*  $f \# h$  (*multi*) for  $f :: 'a \Rightarrow 'b \Rightarrow 'c$  and  $h$

$\Rightarrow \langle \text{intro-cong } h \ \text{more:more } \text{arg-cong2}[\text{where } f=f] \rangle$

| *cong-tag-3*  $f \# h$  (*multi*) for  $f :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd$  and  $h$

$\Rightarrow \langle \text{intro-cong } h \ \text{more:more } \text{arg-cong3}[\text{where } f=f] \rangle$

| -  $\Rightarrow \langle \text{intro more refl} \rangle$ )

bundle *intro-cong-syntax*

begin

notation *cong-tag-1*  $\langle \sigma_1 \rangle$

notation *cong-tag-2*  $\langle \sigma_2 \rangle$

notation *cong-tag-3*  $\langle \sigma_3 \rangle$

end

lemma *restr-Collect-cong*:

assumes  $\bigwedge x. x \in A \implies P \ x = Q \ x$

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**  $\bigwedge n M x. P M \implies \neg(x \in \# M) \implies n > 0 \implies P (M + \text{replicate-mset } n x)$

**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**  $x\text{-def}: x \in \# M$  **using** *multiset-nonemptyE* **by** *auto*

**define**  $M1$  **where**  $M1 = M - \text{replicate-mset } (\text{count } M x) x$

**then have**  $M\text{-def}: M = M1 + \text{replicate-mset } (\text{count } M x) x$

**by** (*metis count-le-replicate-mset-subset-eq dual-order.refl subset-mset.diff-add*)

**have**  $\text{size } M1 < \text{size } M$

**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**  $f :: 'a \Rightarrow 'b::\{\text{semiring-1}\}$

**shows**  $\text{sum-mset } (\text{image-mset } f A) = \text{sum } (\lambda x. \text{of-nat } (\text{count } A x) * f x) (\text{set-mset } A)$

**proof** (*induction A rule: disj-induct-mset*)

**case** *1*

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

**next**

**case** ( $2 n M x$ )

**moreover have**  $\text{count } M x = 0$  **using** *2* **by** (*simp add: count-eq-zero-iff*)

**moreover have**  $\bigwedge y. y \in \text{set-mset } M \implies y \neq x$  **using** *2* **by** *blast*

**ultimately show** *?case* **by** (*simp add: algebra-simps*)

**qed**

**lemma** *sum-mset-conv-2*:

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

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

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

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

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

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

**lemma**  $\text{sum-count-2}$ :  
**assumes**  $\text{finite } F$   
**shows**  $\text{sum } (\text{count } M) F = \text{size } (\text{filter-mset } (\lambda x. x \in F) M)$   
**using**  $\text{assms}$   
**proof** ( $\text{induction } F \text{ rule:finite-induct}$ )  
**case**  $\text{empty}$   
**then show**  $?case$  **by**  $\text{simp}$   
**next**  
**case** ( $\text{insert } x F$ )  
**have**  $\text{sum } (\text{count } M) (\text{insert } x F) = \text{size } (\{\#y \in \# M. y = x\# \} + \{\#x \in \# M. x \in F\#\})$   
**using**  $\text{insert}(1,2,3)$  **by** ( $\text{simp add:count-mset-exp}$ )

**also have** ... = size ( $\{\#y \in \# M. y = x \vee y \in F\#\}$ )  
**using** insert(2)  
**by** (intro arg-cong[where f=size] filter-mset-ex-predicates) simp  
**also have** ... = size (filter-mset ( $\lambda y. y \in \text{insert } x F$ ) M)  
**by** simp  
**finally show** ?case **by** simp  
**qed**

**definition** concat-mset :: ('a multiset) multiset  $\Rightarrow$  'a multiset  
**where** concat-mset xss = fold-mset ( $\lambda xs ys. xs + ys$ )  $\{\#\}$  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 ( $\lambda x. f x + g x$ ) 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 ( $\lambda x. \text{count } x xs$ ) xss)  
**unfolding** concat-mset-def **by** (induction xss, auto)

**lemma** set-mset-concat-mset:  
set-mset (concat-mset xss) =  $\bigcup$  (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**  $\bigwedge i. i \in I \Rightarrow \text{finite } (A i)$   
**assumes**  $\bigwedge i j. i \in I \Rightarrow j \in I \Rightarrow i \neq j \Rightarrow A i \cap A j = \{\}$   
**shows** mset-set ( $\bigcup (A ' I)$ ) = concat-mset (image-mset (mset-set  $\circ A$ ) (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 ( $\bigcup (A ' \text{insert } x F)$ ) = mset-set (A x  $\cup$  ( $\bigcup (A ' F)$ ))

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

**lemma** *size-filter-mset-conv*:

$size (filter\text{-}mset\ f\ A) = sum\text{-}mset (image\text{-}mset (\lambda x. of\text{-}bool (f\ x) :: nat) A)$   
 by (*induction A, auto*)

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

**lemma** *repeat-image-concat-mset*:

$repeat\text{-}mset\ n (image\text{-}mset\ f\ A) = concat\text{-}mset (image\text{-}mset (\lambda x. replicate\text{-}mset\ n (f\ x)) A)$   
**unfolding** *concat-mset-def* by (*induction A, auto*)

**lemma** *mset-prod-eq*:

**assumes** *finite A finite B*

**shows**

$mset\text{-}set (A \times B) = concat\text{-}mset \{ \# \{ \# (x,y). y \in \# mset\text{-}set\ B \# \} . x \in \# mset\text{-}set\ A \# \}$   
 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**  $mset\text{-}set (insert\ x\ F \times B) = mset\text{-}set (F \times B \cup (\lambda y. (x,y))\ 'B)$

by (*intro arg-cong*[**where**  $f=mset\text{-}set$ ]) *auto*

**also have** ... =  $mset\text{-}set (F \times B) + mset\text{-}set ((\lambda y. (x,y))\ 'B)$

using *insert(1,2)* *assms(2)* by (*intro mset-set-Union finite-cartesian-product*) *auto*

**also have** ... =  $mset\text{-}set (F \times B) + \{ \# (x,y). y \in \# mset\text{-}set\ B \# \}$

by (*intro arg-cong2*[**where**  $f=(+)$ ] *image-mset-mset-set[symmetric]* *inj-onI*) *auto*

**also have** ... =  $concat\text{-}mset \{ \# image\text{-}mset (Pair\ x) (mset\text{-}set\ B). x \in \# \{ \#x\# \} + (mset\text{-}set\ F) \# \}$

**unfolding** *insert image-mset-union concat-add-mset-2* by (*simp add:concat-mset-single*)

**also have** ... =  $concat\text{-}mset \{ \# image\text{-}mset (Pair\ x) (mset\text{-}set\ B). x \in \# mset\text{-}set (insert\ x\ F) \# \}$

using *insert(1,2)* by (*intro-cong* [ $\sigma_1\ concat\text{-}mset, \sigma_2\ image\text{-}mset$ ]) *auto*

**finally show** ?*case* by *simp*

**qed**

**lemma** *sum-mset-repeat*:

**fixes**  $f :: 'a \Rightarrow 'b :: \{ comm\text{-}monoid\text{-}add, semiring\text{-}1 \}$

**shows**  $sum\text{-}mset (image\text{-}mset\ f (repeat\text{-}mset\ n\ A)) = of\text{-}nat\ n * sum\text{-}mset (image\text{-}mset\ f\ A)$

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

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  $v \ w \in \text{verts } G$ .

**definition** *symmetric-multi-graph* **where**  $\text{symmetric-multi-graph } G =$

$(\text{fin-digraph } G \wedge (\forall v \ w. \{v, w\} \subseteq \text{verts } G \longrightarrow \text{card } (\text{arcs-betw } G \ w \ v) = \text{card } (\text{arcs-betw } G \ v \ w)))$

**lemma** *symmetric-multi-graphI*:

**assumes** *fin-digraph*  $G$

**assumes** *bij-betw*  $f$  (*arcs*  $G$ ) (*arcs*  $G$ )

**assumes**  $\bigwedge e. e \in \text{arcs } G \implies \text{head } G \ (f \ e) = \text{tail } G \ e \wedge \text{tail } G \ (f \ e) = \text{head } G \ e$

**shows** *symmetric-multi-graph*  $G$

**proof** –

**have**  $\text{card } (\text{arcs-betw } G \ w \ v) = \text{card } (\text{arcs-betw } G \ v \ w)$

(**is**  $?L = ?R$ ) **if**  $v \in \text{verts } G \ w \in \text{verts } G$  **for**  $v \ w$

**proof** –

**have**  $a: f \ x \in \text{arcs } G$  **if**  $x \in \text{arcs } G$  **for**  $x$

**using** *assms(2)* **that** **unfolding** *bij-betw-def* **by** *auto*

**have**  $b: \exists y. y \in \text{arcs } G \wedge f \ y = x$  **if**  $x \in \text{arcs } G$  **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*  $f$   $\{e \in \text{arcs } G. \text{head } G \ e = v \wedge \text{tail } G \ e = w\}$

**by** (*rule inj-on-subset, auto*)

**hence**  $?L = \text{card } (f \ \{e \in \text{arcs } G. \text{head } G \ e = v \wedge \text{tail } G \ e = w\})$

**unfolding** *arcs-betw-def*

**by** (*intro card-image[symmetric]*)

**also have**  $\dots = ?R$

**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**  $\text{card } \{e \in \text{arcs } G. \text{head } G \ e=v \wedge \text{tail } G \ e=w\} = \text{card } \{e \in \text{arcs } G. \text{head } G \ e=w \wedge \text{tail } G \ e=v\}$

(**is**  $\text{card } ?L = \text{card } ?R$ )

**proof** (*cases*  $v \in \text{verts } G \wedge w \in \text{verts } G$ )

**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**

$\text{card } \{e \in \text{arcs } G. \text{tail } G \ e=v \wedge \text{head } G \ e=w\} = \text{card } \{e \in \text{arcs } G. \text{tail } G \ e=w \wedge \text{head } G \ e=v\}$

**using** *symmetric-multi-graphD[OF assms]* **by** (*simp add:conj.commute*)

**lemma** *symmetric-multi-graphD4*:

**assumes** *symmetric-multi-graph*  $G$

**shows**  $\text{card } (\text{arcs-betw } G \ v \ w) = \text{card } (\text{arcs-betw } G \ w \ v)$

**using** *symmetric-multi-graphD[OF assms]* **unfolding** *arcs-betw-def* **by** *simp*

**lemma** *symmetric-degree-eq*:

**assumes** *symmetric-multi-graph*  $G$

**assumes**  $v \in \text{verts } G$

**shows**  $\text{out-degree } G \ v = \text{in-degree } G \ v$  (**is**  $?L = ?R$ )

**proof** –

**interpret** *fin-digraph*  $G$

**using** *assms(1) symmetric-multi-graph-def* **by** *auto*

**have**  $?L = \text{card } \{e \in \text{arcs } G. \text{tail } G \ e = v\}$

**unfolding** *out-degree-def out-arcs-def* **by** *simp*

**also have**  $\dots = \text{card } (\bigcup w \in \text{verts } G. \{e \in \text{arcs } G. \text{head } G \ e = w \wedge \text{tail } G \ e = v\})$

**by** (*intro arg-cong[where f=card]*) (*auto simp add:set-eq-iff*)

**also have**  $\dots = (\sum w \in \text{verts } G. \text{card } \{e \in \text{arcs } G. \text{head } G \ e = w \wedge \text{tail } G \ e = v\})$

**by** (*intro card-UN-disjoint*) *auto*

**also have**  $\dots = (\sum w \in \text{verts } G. \text{card } \{e \in \text{arcs } G. \text{head } G \ e = v \wedge \text{tail } G \ e = w\})$

**by** (*intro sum.cong refl symmetric-multi-graphD assms*)

**also have**  $\dots = \text{card } (\bigcup w \in \text{verts } G. \{e \in \text{arcs } G. \text{head } G \ e = v \wedge \text{tail } G \ e = w\})$

**by** (*intro card-UN-disjoint[symmetric]*) *auto*

**also have**  $\dots = \text{card } (\text{in-arcs } G \ v)$

**by** (*intro arg-cong[where f=card]*) (*auto simp add:set-eq-iff*)

**also have**  $\dots = ?R$

**unfolding** *in-degree-def* **by** *simp*

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

**qed**

**definition** *edges* **where**  $\text{edges } G = \text{image-mset } (\text{arc-to-ends } G) (\text{mset-set } (\text{arcs } G))$

**lemma** (in *fin-digraph*) *count-edges*:  
 $count (edges G) (u,v) = card (arcs-betw G u v) \text{ (is } ?L = ?R)$   
**proof** –  
**have**  $?L = card \{x \in arcs G. arc-to-ends G x = (u, v)\}$   
**unfolding** *edges-def count-mset-exp image-mset-filter-mset-swap[symmetric]* **by** *simp*  
**also have**  $\dots = ?R$   
**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**  $\{\# (y,x). (x,y) \in \# (edges G) \#\} = edges G$   
**proof** –  
**have**  $count \{\#(y, x). (x, y) \in \# edges G \#\} x = count (edges G) x \text{ (is } ?L = ?R) \text{ for } x$   
**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**  $\dots = count (edges G) (fst x, snd x)$   
**unfolding** *count-edges-sym[OF assms]* **by** *simp*  
**also have**  $\dots = count (edges G) x$  **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

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

**definition** *vertices-from G v* =  $\{\# snd e \mid e \in \# edges G. fst e = v \#\}$   
**definition** *vertices-to G v* =  $\{\# fst e \mid e \in \# edges G. snd e = v \#\}$

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

**lemma** *edge-set*:  
**assumes**  $x \in \# edges G$   
**shows**  $fst x \in verts G \wedge snd x \in verts G$   
**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**  $\{\#x \in \# mset-set (arcs G). tail G x = v \#\} = mset-set \{a \in arcs G. tail G a = v\}$   
**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*:

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

**lemma**  $set\text{-}mset\text{-}vertices\text{-}from$ :  
 $set\text{-}mset\ (vertices\text{-}from\ G\ x) \subseteq verts\ G$   
**unfolding**  $vertices\text{-}from\text{-}def$  **using**  $edge\text{-}set$  **by**  $auto$

**lemma**  $set\text{-}mset\text{-}vertices\text{-}to$ :  
 $set\text{-}mset\ (vertices\text{-}to\ G\ x) \subseteq verts\ G$   
**unfolding**  $vertices\text{-}to\text{-}def$  **using**  $edge\text{-}set$  **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**  $regular\text{-}graph = fin\text{-}digraph +$   
**assumes**  $sym$ :  $symmetric\text{-}multi\text{-}graph\ G$   
**assumes**  $verts\text{-}non\text{-}empty$ :  $verts\ G \neq \{\}$   
**assumes**  $arcs\text{-}non\text{-}empty$ :  $arcs\ G \neq \{\}$   
**assumes**  $reg'$ :  $\bigwedge v\ w. v \in verts\ G \implies w \in verts\ G \implies out\text{-}degree\ G\ v = out\text{-}degree\ G\ w$   
**begin**

**definition**  $d$  **where**  $d = out\text{-}degree\ G\ (SOME\ v. v \in verts\ G)$

**lemmas**  $count\text{-}sym = count\text{-}edges\text{-}sym[OF\ sym]$

**lemma**  $reg$ :  
**assumes**  $v \in verts\ G$   
**shows**  $out\text{-}degree\ G\ v = d\ in\text{-}degree\ G\ v = d$   
**proof** –  
**define**  $w$  **where**  $w = (SOME\ v. v \in verts\ G)$   
**have**  $w \in verts\ G$   
**unfolding**  $w\text{-}def$  **using**  $assms(1)$  **by**  $(rule\ someI)$   
**hence**  $out\text{-}degree\ G\ v = out\text{-}degree\ G\ w$   
**by**  $(intro\ reg'\ assms(1))$   
**also have**  $\dots = d$   
**unfolding**  $d\text{-}def\ w\text{-}def$  **by**  $simp$   
**finally show**  $a:out\text{-}degree\ G\ v = d$  **by**  $simp$   
  
**show**  $in\text{-}degree\ G\ v = d$   
**using**  $a\ symmetric\text{-}degree\text{-}eq[OF\ sym\ assms(1)]$  **by**  $simp$   
**qed**

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

**lemma**  $n\text{-}gt\text{-}0$ :  $n > 0$   
**unfolding**  $n\text{-}def$  **using**  $verts\text{-}non\text{-}empty$  **by**  $auto$

**lemma**  $d\text{-}gt\text{-}0$ :  $d > 0$   
**proof** –  
**obtain**  $a$  **where**  $a:a \in arcs\ G$   
**using**  $arcs\text{-}non\text{-}empty$  **by**  $auto$

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

**definition**  $g\text{-inner} :: ('a \Rightarrow ('c :: \text{conjugatable-field})) \Rightarrow ('a \Rightarrow 'c) \Rightarrow 'c$   
**where**  $g\text{-inner } f\ g = (\sum x \in \text{verts } G. (f\ x) * \text{conjugate } (g\ x))$

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

**lemma** *g-inner-simps*:  
 $g\text{-inner } (\lambda x. 0) g = 0$   
 $g\text{-inner } f (\lambda x. 0) = 0$   
 $g\text{-inner } (\lambda x. c * f\ x) g = c * g\text{-inner } f\ g$   
 $g\text{-inner } f (\lambda x. c * g\ x) = \text{conjugate } c * g\text{-inner } f\ g$   
 $g\text{-inner } (\lambda x. f\ x - g\ x) h = g\text{-inner } f\ h - g\text{-inner } g\ h$   
 $g\text{-inner } (\lambda x. f\ x + g\ x) h = g\text{-inner } f\ h + g\text{-inner } g\ h$   
 $g\text{-inner } f (\lambda x. g\ x + h\ x) = g\text{-inner } f\ g + g\text{-inner } f\ h$   
 $g\text{-inner } f (\lambda x. g\ x / c) = g\text{-inner } f\ g / \text{conjugate } c$   
 $g\text{-inner } (\lambda x. f\ x / c) g = g\text{-inner } f\ g / c$   
**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**  $g\text{-norm } f = \text{sqrt } (g\text{-inner } f\ f)$

**lemma** *g-norm-eq*:  $g\text{-norm } f = L2\text{-set } f$  (*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**  $f\ g :: 'a \Rightarrow \text{real}$   
**shows**  $|g\text{-inner } f\ g| \leq g\text{-norm } f * g\text{-norm } g$   
**proof** –  
**have**  $|g\text{-inner } f\ g| \leq (\sum v \in \text{verts } G. |f\ v * g\ v|)$   
**unfolding** *g-inner-def conjugate-real-def* **by** (*intro sum-abs*)  
**also have**  $\dots \leq g\text{-norm } f * g\text{-norm } g$   
**unfolding** *g-norm-eq abs-mult* **by** (*intro L2-set-mult-ineq*)  
**finally show** *?thesis* **by** *simp*  
**qed**

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

**assumes**  $\bigwedge x. x \in \text{verts } G \implies f1\ x = f2\ x$   
**assumes**  $\bigwedge x. x \in \text{verts } G \implies g1\ x = g2\ x$   
**shows**  $g\text{-inner } f1\ g1 = g\text{-inner } f2\ g2$   
**unfolding**  $g\text{-inner-def}$  **using**  $assms$   
**by**  $(\text{intro sum.cong refl})\ \text{auto}$

**lemma**  $g\text{-norm-cong}$ :  
**assumes**  $\bigwedge x. x \in \text{verts } G \implies f\ x = g\ x$   
**shows**  $g\text{-norm } f = g\text{-norm } g$   
**unfolding**  $g\text{-norm-def}$   
**by**  $(\text{intro arg-cong}[\text{where } f=\text{sqrt}]\ g\text{-inner-cong } assms)$

**lemma**  $g\text{-norm-nonneg}$ :  $g\text{-norm } f \geq 0$   
**unfolding**  $g\text{-norm-def } g\text{-inner-def}$   
**by**  $(\text{intro real-sqrt-ge-zero sum-nonneg})\ \text{auto}$

**lemma**  $g\text{-norm-sq}$ :  
 $g\text{-norm } f^2 = g\text{-inner } f\ f$   
**using**  $g\text{-norm-nonneg } g\text{-norm-def}$  **by**  $\text{simp}$

**definition**  $g\text{-step}$  ::  $(a \Rightarrow \text{real}) \Rightarrow (a \Rightarrow \text{real})$   
**where**  $g\text{-step } f\ v = (\sum x \in \text{in-arcs } G\ v. f\ (\text{tail } G\ x) / \text{real } d)$

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

**lemma**  $g\text{-inner-step-eq}$ :  
 $g\text{-inner } f\ (g\text{-step } f) = (\sum a \in \text{arcs } G. f\ (\text{head } G\ a) * f\ (\text{tail } G\ a)) / d$  (**is**  $?L = ?R$ )

**proof** –

**have**  $?L = (\sum v \in \text{verts } G. f\ v * (\sum a \in \text{in-arcs } G\ v. f\ (\text{tail } G\ a) / d))$   
**unfolding**  $g\text{-inner-def } g\text{-step-def}$  **by**  $\text{simp}$   
**also have**  $\dots = (\sum v \in \text{verts } G. (\sum a \in \text{in-arcs } G\ v. f\ v * f\ (\text{tail } G\ a) / d))$   
**by**  $(\text{subst sum-distrib-left})\ \text{simp}$   
**also have**  $\dots = (\sum v \in \text{verts } G. (\sum a \in \text{in-arcs } G\ v. f\ (\text{head } G\ a) * f\ (\text{tail } G\ a) / d))$   
**unfolding**  $\text{in-arcs-def}$  **by**  $(\text{intro sum.cong refl})\ \text{simp}$   
**also have**  $\dots = (\sum a \in (\bigcup (\text{in-arcs } G\ \cdot\ \text{verts } G)). f\ (\text{head } G\ a) * f\ (\text{tail } G\ a) / d)$   
**using**  $\text{finite-verts}$  **by**  $(\text{intro sum.UNION-disjoint[symmetric] ballI})$   
 $(\text{auto simp add:in-arcs-def})$   
**also have**  $\dots = (\sum a \in \text{arcs } G. f\ (\text{head } G\ a) * f\ (\text{tail } G\ a) / d)$   
**unfolding**  $\text{in-arcs-def}$  **using**  $\text{wellformed}$  **by**  $(\text{intro sum.cong})\ \text{auto}$   
**also have**  $\dots = ?R$   
**by**  $(\text{intro sum-divide-distrib[symmetric]})$   
**finally show**  $?thesis$  **by**  $\text{simp}$

**qed**

**definition**  $\Lambda\text{-test}$   
**where**  $\Lambda\text{-test} = \{f. g\text{-norm } f^2 \neq 0 \wedge g\text{-inner } f\ (\lambda \cdot. 1) = 0\}$

**lemma**  $\Lambda\text{-test-ne}$ :  
**assumes**  $n > 1$   
**shows**  $\Lambda\text{-test} \neq \{\}$

**proof** –

**obtain**  $v$  **where**  $v\text{-def}$ :  $v \in \text{verts } G$  **using**  $\text{verts-non-empty}$  **by**  $\text{auto}$   
**have**  $\text{False}$  **if**  $\bigwedge w. w \in \text{verts } G \implies w = v$   
**proof** –

**have**  $\text{verts } G = \{v\}$  **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**  $w\text{-def}: w \in \text{verts } G \ v \neq w$   
**by** *auto*  
**define**  $f$  **where**  $f\ x = (\text{if } x = v \text{ then } 1 \text{ else } (\text{if } x = w \text{ then } (-1) \text{ else } (0::\text{real})))$  **for**  $x$   
  
**have**  $g\text{-norm } f^{\wedge}2 = (\sum_{x \in \text{verts } G}. (\text{if } x = v \text{ then } 1 \text{ else } (\text{if } x = w \text{ then } -1 \text{ else } 0))^2)$   
**unfolding** *g-norm-sq g-inner-def conjugate-real-def power2-eq-square[symmetric]*  
**by** (*simp add:f-def*)  
**also have**  $\dots = (\sum_{x \in \{v,w\}}. (\text{if } x = v \text{ then } 1 \text{ else } (\text{if } x = w \text{ then } -1 \text{ else } 0))^2)$   
**using** *v-def(1) w-def(1)* **by** (*intro sum.mono-neutral-cong refl*) *auto*  
**also have**  $\dots = (\sum_{x \in \{v,w\}}. (\text{if } x = v \text{ then } 1 \text{ else } -1)^2)$   
**by** (*intro sum.cong*) *auto*  
**also have**  $\dots = 2$   
**using** *w-def(2)* **by** (*simp add:if-distrib if-distribR sum.If-cases*)  
**finally have**  $g\text{-norm } f^{\wedge}2 = 2$  **by** *simp*  
**hence**  $g\text{-norm } f \neq 0$  **by** *auto*  
  
**moreover have**  $g\text{-inner } f (\lambda.1) = 0$   
**unfolding** *g-inner-def f-def* **using** *v-def w-def* **by** (*simp add:sum.If-cases*)  
**ultimately have**  $f \in \Lambda\text{-test}$   
**unfolding**  *$\Lambda$ -test-def* **by** *simp*  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma**  *$\Lambda$ -test-empty:*

**assumes**  $n = 1$   
**shows**  $\Lambda\text{-test} = \{\}$

**proof** –

**obtain**  $v$  **where**  $v\text{-def}: \text{verts } G = \{v\}$   
**using** *assms card-1-singletonE* **unfolding** *n-def* **by** *auto*

**have** *False* **if**  $f \in \Lambda\text{-test}$  **for**  $f$

**proof** –

**have**  $0 = (g\text{-inner } f (\lambda.1))^{\wedge}2$   
**using** *that  $\Lambda$ -test-def* **by** *simp*

**also have**  $\dots = (f\ v)^{\wedge}2$   
**unfolding** *g-inner-def v-def* **by** *simp*

**also have**  $\dots = g\text{-norm } f^{\wedge}2$   
**unfolding** *g-norm-sq g-inner-def v-def*

**by** (*simp add:power2-eq-square*)

**also have**  $\dots \neq 0$

**using** *that  $\Lambda$ -test-def* **by** *simp*

**finally show** *False* **by** *simp*

**qed**

**thus** *?thesis* **by** *auto*

**qed**

The following are variational definitions for the maximum 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**  $\Lambda_2 :: \text{real}$

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

**definition**  $\Lambda_a :: \text{real}$

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

**lemma** *sum-arcs-tail*:

fixes  $f :: 'a \Rightarrow ('c :: \text{semiring-1})$

shows  $(\sum a \in \text{arcs } G. f \text{ (tail } G \ a)) = \text{of-nat } d * (\sum v \in \text{verts } G. f \ v)$  (is ?L = ?R)

**proof** –

have ?L =  $(\sum a \in (\bigcup (\text{out-arcs } G \ ' \ \text{verts } G)). f \ \text{(tail } G \ a))$

by (intro sum.cong) auto

also have ... =  $(\sum v \in \text{verts } G. (\sum a \in \text{out-arcs } G \ v. f \ \text{(tail } G \ a)))$

by (intro sum.UNION-disjoint) auto

also have ... =  $(\sum v \in \text{verts } G. \text{of-nat } (\text{out-degree } G \ v) * f \ v)$

unfolding out-degree-def by simp

also have ... =  $(\sum v \in \text{verts } G. \text{of-nat } d * f \ v)$

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  $f :: 'a \Rightarrow ('c :: \text{semiring-1})$

shows  $(\sum a \in \text{arcs } G. f \ \text{(head } G \ a)) = \text{of-nat } d * (\sum v \in \text{verts } G. f \ v)$  (is ?L = ?R)

**proof** –

have ?L =  $(\sum a \in (\bigcup (\text{in-arcs } G \ ' \ \text{verts } G)). f \ \text{(head } G \ a))$

by (intro sum.cong) auto

also have ... =  $(\sum v \in \text{verts } G. (\sum a \in \text{in-arcs } G \ v. f \ \text{(head } G \ a)))$

by (intro sum.UNION-disjoint) auto

also have ... =  $(\sum v \in \text{verts } G. \text{of-nat } (\text{in-degree } G \ v) * f \ v)$

unfolding in-degree-def by simp

also have ... =  $(\sum v \in \text{verts } G. \text{of-nat } d * f \ v)$

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*:

$|\sum a \in \text{arcs } G. f \ \text{(head } G \ a) * f \ \text{(tail } G \ a)| \leq d * g\text{-norm } f^{\wedge 2}$  (is ?L ≤ ?R)

**proof** –

have  $(\sum a \in \text{arcs } G. f \ \text{(head } G \ a)^{\wedge 2}) = d * g\text{-norm } f^{\wedge 2}$

unfolding sum-arcs-head[where f= $\lambda x. f \ x^{\wedge 2}$ ] g-norm-sq g-inner-def

by (simp add:power2-eq-square)

hence 0:L2-set ( $\lambda a. f \ \text{(head } G \ a)$ ) (arcs G)  $\leq \text{sqrt } (d * g\text{-norm } f^{\wedge 2})$

using g-norm-nonneg unfolding L2-set-def by simp

have  $(\sum a \in \text{arcs } G. f \ \text{(tail } G \ a)^{\wedge 2}) = d * g\text{-norm } f^{\wedge 2}$

unfolding sum-arcs-tail[where f= $\lambda x. f \ x^{\wedge 2}$ ] sum-distrib-left[symmetric] g-norm-sq g-inner-def

by (simp add:power2-eq-square)

hence 1:L2-set ( $\lambda a. f \ \text{(tail } G \ a)$ ) (arcs G)  $\leq \text{sqrt } (d * g\text{-norm } f^{\wedge 2})$

unfolding L2-set-def by simp

have ?L  $\leq (\sum a \in \text{arcs } G. |f \ \text{(head } G \ a)| * |f \ \text{(tail } G \ a)|)$

unfolding abs-mult[symmetric] by (intro divide-right-mono sum-abs)

**also have** ...  $\leq (L2\text{-set } (\lambda a. f (\text{head } G a)) (\text{arcs } G) * L2\text{-set } (\lambda a. f (\text{tail } G a)) (\text{arcs } G))$   
**by** *(intro L2-set-mult-ineq)*  
**also have** ...  $\leq (\text{sqrt } (d * g\text{-norm } f^2) * \text{sqrt } (d * g\text{-norm } f^2))$   
**by** *(intro mult-mono 0 1) auto*  
**also have** ...  $= d * g\text{-norm } f^2$   
**using** *d-gt-0 g-norm-nonneg* **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *bdd-above-aux-2:*

**assumes**  $f \in \Lambda\text{-test}$   
**shows**  $|g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f \leq 1$

**proof** –

**have**  $0 < g\text{-inner } f f > 0$   
**using** *assms unfolding*  $\Lambda\text{-test-def } g\text{-norm-sq[symmetric]$  **by** *auto*

**have**  $|g\text{-inner } f (g\text{-step } f)| = |\sum_{a \in \text{arcs } G}. f (\text{head } G a) * f (\text{tail } G a)| / \text{real } d$   
**unfolding** *g-inner-step-eq* **by** *simp*

**also have** ...  $\leq d * g\text{-norm } f^2 / d$   
**by** *(intro divide-right-mono bdd-above-aux assms) auto*

**also have** ...  $= g\text{-inner } f f$   
**using** *d-gt-0 unfolding*  $g\text{-norm-sq}$  **by** *simp*

**finally have**  $|g\text{-inner } f (g\text{-step } f)| \leq g\text{-inner } f f$   
**by** *simp*

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

**qed**

**lemma** *bdd-above-aux-3:*

**assumes**  $f \in \Lambda\text{-test}$   
**shows**  $g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f \leq 1$  (**is**  $?L \leq ?R$ )

**proof** –

**have**  $?L \leq |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f$   
**unfolding**  $g\text{-norm-sq[symmetric]}$

**by** *(intro divide-right-mono) auto*

**also have** ...  $\leq 1$   
**using** *bdd-above-aux-2[OF assms]* **by** *simp*

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

**qed**

**lemma** *bdd-above- $\Lambda$ :*  $bdd\text{-above } ((\lambda f. |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f) \text{ ‘ } \Lambda\text{-test})$

**using** *bdd-above-aux-2*  
**by** *(intro bdd-aboveI[where M=1]) auto*

**lemma** *bdd-above- $\Lambda_2$ :*  $bdd\text{-above } ((\lambda f. g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f) \text{ ‘ } \Lambda\text{-test})$

**using** *bdd-above-aux-3*  
**by** *(intro bdd-aboveI[where M=1]) auto*

**lemma**  *$\Lambda\text{-le-1}$ :*  $\Lambda_a \leq 1$

**proof** (*cases*  $n > 1$ )

**case** *True*

**have**  $(\text{SUP } f \in \Lambda\text{-test}. |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f) \leq 1$

**using** *bdd-above-aux-2*  $\Lambda\text{-test-ne[OF True]}$  **by** *(intro cSup-least) auto*

**thus**  $\Lambda_a \leq 1$

**unfolding**  $\Lambda_a\text{-def}$  **using** *True* **by** *simp*

**next**

**case** *False*

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

lemma  $\Lambda_2$ -le-1:  $\Lambda_2 \leq 1$

proof (cases  $n > 1$ )

case True

have  $(\text{SUP } f \in \Lambda\text{-test. } g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f) \leq 1$

using bdd-above-aux-3  $\Lambda$ -test-ne[OF True] by (intro cSup-least) auto

thus  $\Lambda_2 \leq 1$

unfolding  $\Lambda_2$ -def using True by simp

next

case False

thus ?thesis unfolding  $\Lambda_2$ -def by simp

qed

lemma  $\Lambda$ -ge-0:  $\Lambda_a \geq 0$

proof (cases  $n > 1$ )

case True

obtain  $f$  where  $f$ -def:  $f \in \Lambda$ -test

using  $\Lambda$ -test-ne[OF True] by auto

have  $0 \leq |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f$

unfolding  $g$ -norm-sq[symmetric] by (intro divide-nonneg-nonneg) auto

also have  $\dots \leq (\text{SUP } f \in \Lambda\text{-test. } |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f)$

using  $f$ -def by (intro cSup-upper bdd-above- $\Lambda$ ) auto

finally have  $(\text{SUP } f \in \Lambda\text{-test. } |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f) \geq 0$

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  $\bigwedge f. g\text{-inner } f (\lambda-. 1) = 0 \implies g\text{-inner } f (g\text{-step } f) \leq C * g\text{-norm } f^2$

shows  $\Lambda_2 \leq C$

proof -

have  $g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f \leq C$  if  $f \in \Lambda$ -test for  $f$

proof -

have  $g\text{-inner } f (g\text{-step } f) \leq C * g\text{-inner } f f$

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  $(\text{SUP } f \in \Lambda\text{-test. } g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f) \leq C$

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  $g\text{-inner } f (\lambda-. 1) = 0$

shows  $g\text{-inner } f (g\text{-step } f) \leq \Lambda_2 * g\text{-norm } f^2$  (is ?L  $\leq$  ?R)

proof (cases  $g\text{-norm } f \neq 0$ )

case True

**have**  $0:f \in \Lambda\text{-test}$   
**unfolding**  $\Lambda\text{-test-def}$  **using** *assms True* **by** *auto*  
  
**hence**  $1:n > 1$   
**using**  $\Lambda\text{-test-empty } n\text{-gt-0}$  **by** *fastforce*  
  
**have**  $g\text{-inner } f (g\text{-step } f) / g\text{-norm } f^2 = g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f$   
**unfolding**  $g\text{-norm-sq}$  **by** *simp*  
**also have**  $\dots \leq (SUP f \in \Lambda\text{-test. } g\text{-inner } f (g\text{-step } f) / g\text{-inner } f f)$   
**by** (*intro cSup-upper bdd-above- $\Lambda_2$  imageI 0*)  
**also have**  $\dots = \Lambda_2$   
**using**  $1$  **unfolding**  $\Lambda_2\text{-def}$  **by** *simp*  
**finally have**  $g\text{-inner } f (g\text{-step } f) / g\text{-norm } f^2 \leq \Lambda_2$  **by** *simp*  
**thus** *?thesis*  
**using** *True* **by** (*simp add:divide-simps*)

**next**

**case** *False*  
**hence**  $g\text{-inner } f f = 0$   
**unfolding**  $g\text{-norm-sq[symmetric]}$  **by** *simp*  
**hence**  $0:\bigwedge v. v \in \text{verts } G \implies f v = 0$   
**unfolding**  $g\text{-inner-def}$  **by** (*subst (asm) sum-nonneg-eq-0-iff*) *auto*  
**hence**  $?L = 0$   
**unfolding**  $g\text{-step-def } g\text{-inner-def}$  **by** *simp*  
**also have**  $\dots \leq \Lambda_2 * g\text{-norm } f^2$   
**using** *False* **by** *simp*  
**finally show** *?thesis* **by** *simp*

**qed**

**lemma** *expander-intro-1:*

**assumes**  $C \geq 0$   
**assumes**  $\bigwedge f. g\text{-inner } f (\lambda-. 1)=0 \implies |g\text{-inner } f (g\text{-step } f)| \leq C * g\text{-norm } f^2$   
**shows**  $\Lambda_a \leq C$

**proof** (*cases n > 1*)

**case** *True*  
**have**  $|g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f \leq C$  **if**  $f \in \Lambda\text{-test}$  **for**  $f$   
**proof** –  
**have**  $|g\text{-inner } f (g\text{-step } f)| \leq C * g\text{-inner } f f$   
**using** *that*  $\Lambda\text{-test-def assms(2)}$  **unfolding**  $g\text{-norm-sq}$  **by** *auto*  
**moreover have**  $g\text{-inner } f f > 0$   
**using** *that* **unfolding**  $\Lambda\text{-test-def } g\text{-norm-sq[symmetric]}$  **by** *auto*  
**ultimately show** *?thesis*  
**by** (*simp add:divide-simps*)

**qed**

**hence**  $(SUP f \in \Lambda\text{-test. } |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f) \leq C$   
**using**  $\Lambda\text{-test-ne[OF True]}$  **by** (*intro cSup-least*) *auto*  
**thus** *?thesis* **using** *True* **unfolding**  $\Lambda_a\text{-def}$  **by** *auto*

**next**

**case** *False*  
**then show** *?thesis* **using** *assms* **unfolding**  $\Lambda_a\text{-def}$  **by** *simp*

**qed**

**lemma** *expander-intro:*

**assumes**  $C \geq 0$   
**assumes**  $\bigwedge f. g\text{-inner } f (\lambda-. 1)=0 \implies |\sum a \in \text{arcs } G. f(\text{head } G a) * f(\text{tail } G a)| \leq C * g\text{-norm } f^2$   
**shows**  $\Lambda_a \leq C/d$

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

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

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

**have**  $|g\text{-inner } f (g\text{-step } f)| / g\text{-norm } f^2 = |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f$   
**unfolding**  $g\text{-norm-sq}$  **by**  $\text{simp}$   
**also have**  $\dots \leq (SUP f \in \Lambda\text{-test}. |g\text{-inner } f (g\text{-step } f)| / g\text{-inner } f f)$   
**by** (*intro cSup-upper bdd-above- $\Lambda$  imageI 0*)  
**also have**  $\dots = \Lambda_a$   
**using**  $1$  **unfolding**  $\Lambda_a\text{-def}$  **by**  $\text{simp}$   
**finally have**  $|g\text{-inner } f (g\text{-step } f)| / g\text{-norm } f^2 \leq \Lambda_a$  **by**  $\text{simp}$   
**thus**  $?thesis$   
**using**  $\text{True}$  **by** (*simp add:divide-simps*)

**next**  
**case** *False*  
**hence**  $g\text{-inner } f f = 0$   
**unfolding**  $g\text{-norm-sq[symmetric]}$  **by**  $\text{simp}$   
**hence**  $0 : \bigwedge v. v \in \text{verts } G \implies f v = 0$   
**unfolding**  $g\text{-inner-def}$  **by** (*subst (asm) sum-nonneg-eq-0-iff*) **auto**  
**hence**  $?L = 0$   
**unfolding**  $g\text{-step-def } g\text{-inner-def}$  **by**  $\text{simp}$   
**also have**  $\dots \leq \Lambda_a * g\text{-norm } f^2$   
**using**  $\text{False}$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$   
**qed**

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

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

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

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

**lemma edge-expansionD:**

assumes  $S \subseteq \text{verts } G \wedge 2 * \text{card } S \leq n$   
shows  $\Lambda_e * \text{card } S \leq \text{real } (\text{card } (\text{edges-betw } S \ (-S)))$

**proof** (cases  $S \neq \{\}$ )

case True

moreover have finite S

using finite-subset[OF assms(1)] by simp

ultimately have  $\text{card } S > 0$  by auto

hence 1:  $\text{real } (\text{card } S) > 0$  by simp

hence 2:  $n > 1$  using assms(2) by simp

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

have 0: finite ?St

by (rule finite-subset[where B=Pow (verts G)]) auto

have  $\Lambda_e = (\text{MIN } S \in ?St. \text{real } (\text{card } (\text{edges-betw } S \ (-S))) / \text{card } S)$

using 2 unfolding  $\Lambda_e$ -def by simp

also have  $\dots \leq \text{real } (\text{card } (\text{edges-betw } S \ (-S))) / \text{card } S$

using assms True by (intro Min-le finite-imageI imageI) auto

finally have  $\Lambda_e \leq \text{real } (\text{card } (\text{edges-betw } S \ (-S))) / \text{card } S$  by simp

thus ?thesis using 1 by (simp add:divide-simps)

next

case False

hence  $\text{card } S = 0$  by simp

thus ?thesis by simp

qed

**lemma edge-expansionI:**

fixes  $\alpha :: \text{real}$

assumes  $n > 1$

assumes  $\bigwedge S. S \subseteq \text{verts } G \implies 2 * \text{card } S \leq n \implies S \neq \{\} \implies \text{card } (\text{edges-betw } S \ (-S)) \geq \alpha * \text{card } S$

shows  $\Lambda_e \geq \alpha$

**proof** –

define St where  $St = \{S. S \subseteq \text{verts } G \wedge 2 * \text{card } S \leq n \wedge S \neq \{\}\}$

have 0: finite St

unfolding St-def

by (rule finite-subset[where B=Pow (verts G)]) auto

obtain v where v-def:  $v \in \text{verts } G$  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:  $\alpha \leq \text{real}(\text{card}(\text{edges-betw } S (- S))) / \text{real}(\text{card } S)$  if  $S \in \text{St}$  for  $S$   
**proof** –

  have  $\text{real}(\text{card}(\text{edges-betw } S (- S))) \geq \alpha * \text{card } S$   
   **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**  $\text{card } S > 0$   
   **using** that **unfolding** *St-def* **by** *auto*  
  **ultimately show** *?thesis*  
   **by** (*simp add:divide-simps*)

**qed**

have  $\alpha \leq (\text{MIN } S \in \text{St}. \text{real}(\text{card}(\text{edges-betw } S (- S))) / \text{real}(\text{card } S))$   
  **using** 0 1 2  
  **by** (*intro Min.boundedI finite-imageI*) *auto*

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

**end**

**lemma** *regular-graphI*:

**assumes** *symmetric-multi-graph G*  
  **assumes**  $\text{verts } G \neq \{\}$   $d > 0$   
  **assumes**  $\bigwedge v. v \in \text{verts } G \implies \text{out-degree } G v = d$   
  **shows** *regular-graph G*

**proof** –

**obtain**  $v$  **where** *v-def: v ∈ verts G*  
   **using** *assms(2)* **by** *auto*  
  **have**  $\text{arcs } G \neq \{\}$   
  **proof** (*rule ccontr*)  
   **assume**  $\neg \text{arcs } G \neq \{\}$   
   **hence**  $\text{arcs } G = \{\}$  **by** *simp*  
   **hence**  $\text{out-degree } G v = 0$   
   **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**  $v \in \text{verts } G$   $w \in \text{verts } G$  **for**  $v$   $w$

**proof** –

**have**  $?L$  = { $a \in \text{iso-arcs } h$  ‘ *arcs*  $G$ . *iso-head*  $h$   $a$ =*iso-verts*  $h$   $w$   $\wedge$  *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** ... = { $a$ . ( $\exists b \in \text{arcs } G$ .  $a = \text{iso-arcs } h$   $b$   $\wedge$  *iso-verts*  $h$  (*head*  $G$   $b$ ) = *iso-verts*  $h$   $w$   $\wedge$  *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** ... = { $a$ . ( $\exists b \in \text{arcs } G$ .  $a = \text{iso-arcs } h$   $b$   $\wedge$  *head*  $G$   $b$  =  $w$   $\wedge$  *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*  $H$   $v$   $w$ ) = *card* (*arcs-betw*  $H$   $v$   $w$ ) (*is*  $?L$  =  $?R$ )  
**if**  $v$ -*range*:  $v \in \text{verts } H$  **and**  $w$ -*range*:  $w \in \text{verts } H$  **for**  $v$   $w$

**proof** –

**obtain**  $v'$  **where**  $v'$ :  $v = \text{iso-verts } h$   $v'$   $v' \in \text{verts } G$

**using** *that v-range verts-app-iso* **unfolding**  $H$ -*alt* **by** *auto*

**obtain**  $w'$  **where**  $w'$ :  $w = \text{iso-verts } h$   $w'$   $w' \in \text{verts } G$

**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*:  $H = \text{app-iso } h \ G$   
**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*:  $h\text{-wit} \in \text{verts } H$   
**by** *auto*

**have** *3:out-degree H v = d* **if** *v-range*:  $v \in \text{verts } H$  **for** *v*  
**proof** –

**obtain** *v'* **where** *v'*:  $v = \text{iso-verts } h \ v' \ v' \in \text{verts } G$   
**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**  $\dots = d$   
**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**  $H.n = \text{card } (\text{iso-verts } h \ \text{verts } G)$   
**unfolding** *H.n-def* **unfolding** *H-alt* *verts-app-iso* **by** *simp*  
**also have**  $\dots = \text{card } (\text{verts } G)$   
**by** (*intro card-image digraph-isomorphism-inj-on-verts hom-iso*)  
**also have**  $\dots = n$   
**unfolding** *n-def* **by** *simp*  
**finally show** *n-eq*:  $H.n = n$  **by** *simp*

**have**  $H.d = \text{out-degree } H \ h\text{-wit}$   
**by** (*intro H.reg*[*symmetric*] *h-wit*)  
**also have**  $\dots = d$   
**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*:  
 $H.g\text{-inner } f \ g = g\text{-inner } (\lambda x. f \ (\text{iso-verts } h \ x)) \ (\lambda x. g \ (\text{iso-verts } h \ x))$   
**for**  $f \ g :: 'c \Rightarrow \text{real}$   
**unfolding** *g-inner-def* *H.g-inner-def* **by** (*intro sum.reindex-bij-betw*[*symmetric*])

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

**for**  $f :: 'c \Rightarrow \text{real}$  **and**  $x$   
**proof** –  
**have**  $\text{inj-on } (\text{iso-arcs } h) (\text{in-arcs } G \ x)$   
**using**  $\text{inj-on-subset}[OF \ \text{digraph-isomorphism-inj-on-arcs}[OF \ \text{hom-iso}]]$   
**by**  $(\text{simp add:in-arcs-def})$   
**moreover have**  $\text{in-arcs } H (\text{iso-verts } h \ x) = \text{iso-arcs } h \ ' \ \text{in-arcs } G \ x$   
**unfolding**  $H\text{-alt}$  **by**  $(\text{intro in-arcs-app-iso-eq}[OF \ \text{hom-iso}] \ \text{that})$   
**moreover have**  $\text{tail } H (\text{iso-arcs } h \ a) = \text{iso-verts } h \ (\text{tail } G \ a)$  **if**  $a \in \text{in-arcs } G \ x$  **for**  $a$   
**unfolding**  $H\text{-alt}$  **using**  $\text{that}$  **by**  $(\text{simp add: hom-iso iso-verts-tail})$   
**ultimately show**  $?thesis$   
**unfolding**  $g\text{-step-def } H.g\text{-step-def}$   
**by**  $(\text{intro-cong } [\sigma_2(/), \sigma_1 \ f, \sigma_1 \ \text{of-nat}] \ \text{more: } 4 \ \text{sum.reindex-cong}[\mathbf{where} \ l=\text{iso-arcs } h])$   
**qed**

**show**  $H.\Lambda_a \leq \Lambda_a$   
**using**  $\text{expansionD1}$  **by**  $(\text{intro } H.\text{expander-intro-1 } \Lambda\text{-ge-0})$   
 $(\text{simp add:g-inner-conv } g\text{-step-conv } H.g\text{-norm-sq } g\text{-norm-sq } \text{cong:g-inner-conv})$

**show**  $H.\Lambda_2 \leq \Lambda_2$   
**proof**  $(\text{cases } n > 1)$   
**case**  $\text{True}$   
**hence**  $H.n > 1$   
**by**  $(\text{simp add:n-eq})$   
**thus**  $?thesis$   
**using**  $\text{os-expanderD}$  **by**  $(\text{intro } H.\text{os-expanderI})$   
 $(\text{simp-all add:g-inner-conv } g\text{-step-conv } H.g\text{-norm-sq } g\text{-norm-sq } \text{cong:g-inner-conv})$

**next**  
**case**  $\text{False}$   
**thus**  $?thesis$   
**unfolding**  $H.\Lambda_2\text{-def } \Lambda_2\text{-def}$  **by**  $(\text{simp add:n-eq})$   
**qed**

**show**  $H.\Lambda_e \geq \Lambda_e$   
**proof**  $(\text{cases } n > 1)$   
**case**  $\text{True}$   
**hence**  $n\text{-gt-1: } H.n > 1$   
**by**  $(\text{simp add:n-eq})$   
**have**  $\Lambda_e * \text{real } (\text{card } S) \leq \text{real } (\text{card } (H.\text{edges-betw } S \ (-S)))$   
**if**  $S \subseteq \text{verts } H \ 2 * \text{card } S \leq H.n \ S \neq \{\}$  **for**  $S$   
**proof** –  
**define**  $T$  **where**  $T = \text{iso-verts } h \ -' \ S \cap \text{verts } G$   
**have**  $4:\text{card } T = \text{card } S$   
**using**  $\text{that}(1)$  **unfolding**  $T\text{-def } H\text{-alt } \text{verts-app-iso}$   
**by**  $(\text{intro card-vimage-inj-on digraph-isomorphism-inj-on-verts}[OF \ \text{hom-iso}]) \ \text{auto}$

**have**  $\text{card } (H.\text{edges-betw } S \ (-S)) = \text{card } \{a \in \text{iso-arcs } h \ ' \ \text{arcs } G. \ \text{iso-tail } h \ a \in S \wedge \text{iso-head } h \ a \in -S\}$   
**unfolding**  $H.\text{edges-betw-def}$  **unfolding**  $H\text{-alt } \text{tail-app-iso } \text{head-app-iso } \text{arcs-app-iso}$   
**by**  $\text{simp}$   
**also have**  $\dots =$   
 $\text{card}(\text{iso-arcs } h \ ' \ \{a \in \text{arcs } G. \ \text{iso-tail } h \ (\text{iso-arcs } h \ a) \in S \wedge \text{iso-head } h \ (\text{iso-arcs } h \ a) \in -S\})$   
**by**  $(\text{intro arg-cong}[\mathbf{where} \ f=\text{card}]) \ \text{auto}$   
**also have**  $\dots = \text{card } \{a \in \text{arcs } G. \ \text{iso-tail } h \ (\text{iso-arcs } h \ a) \in S \wedge \text{iso-head } h \ (\text{iso-arcs } h \ a) \in -S\}$   
**by**  $(\text{intro card-image inj-on-subset}[OF \ \text{digraph-isomorphism-inj-on-arcs}[OF \ \text{hom-iso}]] \ \text{auto})$   
**also have**  $\dots = \text{card } \{a \in \text{arcs } G. \ \text{iso-verts } h \ (\text{tail } G \ a) \in S \wedge \text{iso-verts } h \ (\text{head } G \ a) \in -S\}$   
**by**  $(\text{intro restr-Collect-cong arg-cong}[\mathbf{where} \ f=\text{card}])$   
 $(\text{simp add: iso-verts-tail}[OF \ \text{hom-iso}] \ \text{iso-verts-head}[OF \ \text{hom-iso}])$   
**also have**  $\dots = \text{card } \{a \in \text{arcs } G. \ \text{tail } G \ a \in T \wedge \text{head } G \ a \in -T \}$

**unfolding**  $T$ -def by (intro-cong [ $\sigma_1(\text{card}), \sigma_2(\wedge)$ ] 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:  $T \subseteq \text{verts } G$  **unfolding**  $T$ -def by simp

**have**  $\Lambda_e * \text{real}(\text{card } S) = \Lambda_e * \text{real}(\text{card } T)$   
**unfolding** 4 by simp  
**also have** ...  $\leq \text{real}(\text{card}(\text{edges-betw } T(-T)))$   
**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:  $\exists (f :: ('n \Rightarrow 'a)) g. \text{type-definition } f \ g \ (\text{verts } G)$ 
begin

```

```

definition td-components :: ('n  $\Rightarrow$  'a)  $\times$  ('a  $\Rightarrow$  '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:  $\exists q. \text{type-definition } ((\text{fst } q)::('n \Rightarrow 'a)) \ (\text{snd } q) \ (\text{verts } G)$ 
    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)  $\hat{\sim}$  n  $\hat{\sim}$  n where

```

```

  A = ( $\chi \ i \ j. \text{of-nat } (\text{count } (\text{edges } G) (\text{enum-verts } j, \text{enum-verts } i)) / \text{of-nat } d$ )

```

```

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:
   $(\chi \ i. \ g\text{-step } f \ (\text{enum-verts } i)) = A * v \ (\chi \ i. \ f \ (\text{enum-verts } i))$ 
proof -
  have g-step f (enum-verts i) =  $(\sum j \in UNIV. A \ \$ \ i \ \$ \ j * f \ (\text{enum-verts } j))$  (is ?L = ?R) for i
  proof -
    have ?L =  $(\sum x \in \text{in-arcs } G \ (\text{enum-verts } i). f \ (\text{tail } G \ x) / d)$ 
      unfolding g-step-def by simp
    also have ... =  $(\sum x \in \#\text{vertices-to } G \ (\text{enum-verts } i). f \ x / d)$ 
      unfolding verts-to-alt sum-unfold-sum-mset by (simp add:image-mset.compositionality
    comp-def)
    also have ... =  $(\sum j \in \text{verts } G. (\text{count } (\text{vertices-to } G \ (\text{enum-verts } i)) \ j) * (f \ j / \text{real } d))$ 
      by (intro sum-mset-conv-2 set-mset-vertices-to) auto
    also have ... =  $(\sum j \in \text{verts } G. (\text{count } (\text{edges } G) \ (j, \text{enum-verts } i)) * (f \ j / \text{real } d))$ 
      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 ... =  $(\sum j \in UNIV. (\text{count } (\text{edges } G) \ (\text{enum-verts } j, \text{enum-verts } i)) * (f \ (\text{enum-verts } j) / \text{real } d))$ 
      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:
   $g\text{-inner } f \ g = (\chi \ i. \ f \ (\text{enum-verts } i)) \cdot (\chi \ i. \ g \ (\text{enum-verts } i))$ 
  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:
   $g\text{-norm } f = \text{norm } (\chi \ i. \ f \ (\text{enum-verts } i))$ 
proof -
  have g-norm f2 =  $\text{norm } (\chi \ i. \ f \ (\text{enum-verts } i))^2$ 
    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  $\exists (f :: 'n :: \text{finite}) \Rightarrow 'a$  g. type-definition f g (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  $\exists (Rep :: 'n \Rightarrow 'a) Abs.$  type-definition Rep Abs (verts G)
  shows class.finite TYPE('n)
proof –
  obtain  $Rep :: 'n \Rightarrow 'a$  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
    by (auto simp: class.finite-def simp flip: Abs-image)
qed

lemma remove-finite-premise:
  (class.finite TYPE('n)  $\implies \exists (Rep :: 'n \Rightarrow 'a) Abs.$  type-definition Rep Abs (verts G)  $\implies PROP Q$ )
 $\equiv (\exists (Rep :: 'n \Rightarrow 'a) Abs.$  type-definition Rep Abs (verts G)  $\implies PROP Q$ )
  (is ?L  $\equiv$  ?R)
proof (intro Pure.equal-intr-rule)
  assume  $e:\exists (Rep :: 'n \Rightarrow 'a) Abs.$  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  $\exists (Rep :: 'n \Rightarrow 'a) Abs.$  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**  $v \cdot w = 0$

**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*  $:: ('a :: zero)^n \Rightarrow 'a^{n \times n}$

**where**  $diag v = (\chi i j. \text{if } i = j \text{ then } (v \$ i) \text{ else } 0)$

**definition** *ind-vec*  $:: 'n \text{ set} \Rightarrow real^n$

**where**  $ind-vec S = (\chi i. \text{of-bool}(i \in S))$

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

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

**definition** *matrix-norm-bound* ::  $\text{real}^n \text{ } ^m \Rightarrow \text{real} \Rightarrow \text{bool}$   
**where** *matrix-norm-bound*  $A \ l = (\forall x. \text{norm } (A *v x) \leq l * \text{norm } x)$

**lemma** *matrix-norm-boundI*:  
**assumes**  $\bigwedge x. \text{norm } (A *v x) \leq l * \text{norm } x$   
**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**  $\text{norm } (A *v x) \leq l * \text{norm } x$   
**using** *assms unfolding matrix-norm-bound-def* **by** *simp*

**lemma** *matrix-norm-bound-nonneg*:  
**fixes**  $A :: \text{real}^n \text{ } ^m$   
**assumes** *matrix-norm-bound*  $A \ l$   
**shows**  $l \geq 0$

**proof** –  
**have**  $0 \leq \text{norm } (A *v 1)$  **by** *simp*  
**also have**  $\dots \leq l * \text{norm } (1 :: \text{real}^n)$   
**using** *assms(1) unfolding matrix-norm-bound-def* **by** *simp*  
**finally have**  $0 \leq l * \text{norm } (1 :: \text{real}^n)$   
**by** *simp*  
**moreover have**  $\text{norm } (1 :: \text{real}^n) > 0$   
**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**  $A = (0 :: \text{real}^n \text{ } ^m)$

**proof** (*intro iffD2[OF matrix-eq] allI*)  
**fix**  $x :: \text{real}^n$   
**have**  $\text{norm } (A *v x) = 0$   
**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**  $x :: \text{real}^n$   
**assumes**  $\bigwedge i. |x \ \$ \ i| \leq l$   
**shows** *matrix-norm-bound* (*diag*  $x$ )  $l$

**proof** (*rule matrix-norm-boundI*)  
**fix**  $y :: \text{real}^n$

**have** *l-ge-0*:  $l \geq 0$  **using** *assms* **by** *fastforce*

**have**  $a: |x \$ i * v| \leq |l * v|$  **for**  $v i$   
**using**  $l\text{-ge-}0$  **assms** **by** ( $\text{simp add:abs-mult mult-right-mono}$ )

**have**  $\text{norm } (\text{diag } x * v y) = \text{sqrt } (\sum i \in \text{UNIV}. (x \$ i * y \$ i)^2)$   
**unfolding**  $\text{matrix-vector-mult-def diag-def norm-vec-def L2-set-def}$   
**by** ( $\text{auto simp add:if-distrib if-distribR sum.If-cases}$ )

**also have**  $\dots \leq \text{sqrt } (\sum i \in \text{UNIV}. (l * y \$ i)^2)$   
**by** ( $\text{intro real-sqrt-le-mono sum-mono iffD1[OF abs-le-square-iff] a}$ )

**also have**  $\dots = l * \text{norm } y$   
**using**  $l\text{-ge-}0$  **by** ( $\text{simp add:norm-vec-def L2-set-def algebra-simps sum-distrib-left[symmetric] real-sqrt-mult}$ )

**finally show**  $\text{norm } (\text{diag } x * v y) \leq l * \text{norm } y$  **by**  $\text{simp}$   
**qed**

**lemma**  $\text{vector-scaleR-matrix-ac-2}: b *_R (A::\text{real}^n{}^m) * v x = b *_R (A * v x)$   
**unfolding**  $\text{vector-transpose-matrix[symmetric] transpose-scalar}$   
**by** ( $\text{intro vector-scaleR-matrix-ac}$ )

**lemma**  $\text{matrix-norm-bound-scale}$ :  
**assumes**  $\text{matrix-norm-bound } A l$   
**shows**  $\text{matrix-norm-bound } (b *_R A) (|b| * l)$   
**proof** ( $\text{intro matrix-norm-boundI}$ )  
**fix**  $x$   
**have**  $\text{norm } (b *_R A * v x) = \text{norm } (b *_R (A * v x))$   
**by** ( $\text{metis transpose-scalar vector-scaleR-matrix-ac vector-transpose-matrix}$ )  
**also have**  $\dots = |b| * \text{norm } (A * v x)$   
**by**  $\text{simp}$   
**also have**  $\dots \leq |b| * (l * \text{norm } x)$   
**using**  $\text{assms matrix-norm-bound-def}$  **by** ( $\text{intro mult-left-mono}$ )  $\text{auto}$   
**also have**  $\dots \leq (|b| * l) * \text{norm } x$  **by**  $\text{simp}$   
**finally show**  $\text{norm } (b *_R A * v x) \leq (|b| * l) * \text{norm } x$  **by**  $\text{simp}$   
**qed**

**definition**  $\text{nonneg-mat} :: \text{real}^n{}^m \Rightarrow \text{bool}$   
**where**  $\text{nonneg-mat } A = (\forall i j. A \$ i \$ j \geq 0)$

**lemma**  $\text{nonneg-mat-1}$ :  
**shows**  $\text{nonneg-mat } (\text{mat } 1)$   
**unfolding**  $\text{nonneg-mat-def mat-def}$  **by**  $\text{auto}$

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

**lemma**  $\text{nonneg-mat-transpose}$ :  
 $\text{nonneg-mat } (\text{transpose } A) = \text{nonneg-mat } A$   
**unfolding**  $\text{nonneg-mat-def transpose-def}$   
**by**  $\text{auto}$

**definition**  $\text{spec-bound} :: \text{real}^n{}^n \Rightarrow \text{real} \Rightarrow \text{bool}$   
**where**  $\text{spec-bound } M l = (l \geq 0 \wedge (\forall v. v \cdot 1 = 0 \longrightarrow \text{norm } (M * v v) \leq l * \text{norm } v))$

**lemma**  $\text{spec-boundD1}$ :  
**assumes**  $\text{spec-bound } M l$   
**shows**  $0 \leq l$

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

**lemma** *spec-boundD2*:

**assumes** *spec-bound*  $M$   $l$

**assumes**  $v \cdot 1 = 0$

**shows**  $\text{norm } (M *v v) \leq l * \text{norm } v$

**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**  $\text{norm } (M *v v) \leq \beta * \text{norm } v$  **if** *inner*  $v$   $1 = 0$  **for**  $v$

**proof** –

**have**  $\text{norm } (M *v v) \leq \alpha * \text{norm } v$

**by** (*intro spec-boundD2[OF assms(1)] that*)

**also have**  $\dots \leq \beta * \text{norm } v$

**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* ::  $\text{real}^n \Rightarrow \text{bool}$

**where** *markov*  $M = (\text{nonneg-mat } M \wedge M *v 1 = 1 \wedge 1 v* M = 1)$

**lemma** *markov-symI*:

**assumes** *nonneg-mat*  $A$  *transpose*  $A = A$   $A *v 1 = 1$

**shows** *markov*  $A$

**proof** –

**have**  $1 v* A = \text{transpose } A *v 1$

**unfolding** *vector-transpose-matrix[symmetric]* **by** *simp*

**also have**  $\dots = 1$  **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*  $A = \text{markov } (\text{transpose } A)$

**unfolding** *markov-def nonneg-mat-transpose* **by** *auto*

**fun** *matrix-pow* **where**

*matrix-pow*  $M$   $0 = \text{mat } 1$  |

*matrix-pow*  $M$  (*Suc*  $n$ ) =  $M ** (\text{matrix-pow } M$   $n)$

**lemma** *markov-orth-inv*:

**assumes** *markov*  $A$

**shows** *inner*  $(A *v x)$   $1 = \text{inner } x$   $1$

**proof** –

**have** *inner*  $(A *v x)$   $1 = \text{inner } x$   $(1 v* A)$

**using** *dot-lmul-matrix inner-commute* **by** *metis*  
**also have**  $\dots = \text{inner } x \ 1$   
**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**  $\text{norm } ((A ** B) * v \ x) \leq (la * lb) * \text{norm } x$  **if** *inner x 1 = 0* **for**  $x$

**proof** –  
**have**  $\text{norm } ((A ** B) * v \ x) = \text{norm } (A * v \ (B * v \ x))$   
**by** (*simp add:matrix-vector-mul-assoc*)  
**also have**  $\dots \leq la * \text{norm } (B * v \ x)$   
**by** (*intro spec-boundD2[OF assms(3)] (simp add:markov-orth-inv that assms(2))*)  
**also have**  $\dots \leq la * (lb * \text{norm } x)$   
**by** (*intro spec-boundD2[OF assms(4)] mult-left-mono that la-ge-0*)  
**finally show** *?thesis* **by** *simp*

**qed**  
**moreover have**  $la * lb \geq 0$   
**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) ( $l^{\wedge}k$ )
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) ( $l * l^{\wedge}k$ )
    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  $\neq$  []
  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  $\neq$  []
  shows foldl f a ((intersperse x xs)@[x]) = foldl ( $\lambda y z. f (f y z) x$ ) 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 ( $\lambda x z. f (f x y) z$ ) (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 ( $\lambda x. f (f x y)$ ) (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* ::  $\text{real}^n$   
**where** *stat* =  $(1 / \text{real } \text{CARD}(n)) *_{\mathbb{R}} 1$

**definition** *J* ::  $(c :: \text{field})^n$   
**where** *J* =  $(\chi \text{ } i \text{ } j. \text{of-nat } 1 / \text{of-nat } \text{CARD}(n))$

**lemma** *inner-1-1*:  $1 \cdot (1 :: \text{real}^n) = \text{CARD}(n)$   
**unfolding** *inner-vec-def* **by** *simp*

**definition** *proj-unit* ::  $\text{real}^n \Rightarrow \text{real}^n$   
**where** *proj-unit* *v* =  $(1 \cdot v) *_{\mathbb{R}} \text{stat}$

**definition** *proj-rem* ::  $\text{real}^n \Rightarrow \text{real}^n$   
**where** *proj-rem* *v* =  $v - \text{proj-unit } v$

**lemma** *proj-rem-orth*:  $1 \cdot (\text{proj-rem } v) = 0$   
**unfolding** *proj-rem-def* *proj-unit-def* *inner-diff-right* *stat-def*  
**by** (*simp add:inner-1-1*)

**lemma** *split-vec*:  $v = \text{proj-unit } v + \text{proj-rem } v$   
**unfolding** *proj-rem-def* **by** *simp*

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

**lemma** *spec-bound-J*:  $\text{spec-bound } (J :: \text{real}^n) 0$   
**proof** –  
**have**  $\text{norm } (J * v v) = 0$  **if**  $\text{inner } v 1 = 0$  **for**  $v :: \text{real}^n$   
**proof** –  
**have**  $\text{inner } (\text{proj-unit } v + \text{proj-rem } v) 1 = 0$   
**using** *that* **by** (*subst (asm) split-vec[of v], simp*)  
**hence**  $\text{inner } (\text{proj-unit } v) 1 = 0$   
**using** *proj-rem-orth* *inner-commute* **unfolding** *inner-add-left*  
**by** (*metis add-cancel-left-right*)  
**hence**  $\text{proj-unit } v = 0$   
**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* ::  $\text{real}^n$   
**assumes** *markov* *A*  
**shows**  $\text{spec-bound } A \text{ } l \iff \text{matrix-norm-bound } (A - (1-l) *_{\mathbb{R}} J) \text{ } l$  (**is** *?L*  $\iff$  *?R*)  
**proof**

**assume**  $a: ?L$   
**hence**  $l\text{-ge-}0: l \geq 0$  **using** *spec-boundD1* **by** *auto*  
**show**  $?R$   
**proof** (*rule matrix-norm-boundI*)  
**fix**  $x :: \text{real}^n$   
**have**  $(A - (1-l) *_R J) *v x = A *v x - (1-l) *_R (\text{proj-unit } x)$   
**by** (*simp add: algebra-simps vector-scaleR-matrix-ac-2 apply-J*)  
**also have**  $\dots = A *v \text{proj-unit } x + A *v \text{proj-rem } x - (1-l) *_R (\text{proj-unit } x)$   
**by** (*subst split-vec[of x], simp add: algebra-simps*)  
**also have**  $\dots = \text{proj-unit } x + A *v \text{proj-rem } x - (1-l) *_R (\text{proj-unit } x)$   
**using** *markov-apply[OF assms(1)]*  
**unfolding** *proj-unit-def stat-def* **by** (*simp add: algebra-simps*)  
**also have**  $\dots = A *v \text{proj-rem } x + l *_R \text{proj-unit } x$  (**is - =**  $?R1$ )  
**by** (*simp add: algebra-simps*)  
**finally have**  $d: (A - (1-l) *_R J) *v x = ?R1$  **by** *simp*  
  
**have inner**  $(l *_R \text{proj-unit } x) (A *v \text{proj-rem } x) =$   
 $\text{inner} ((l * \text{inner } 1 x / \text{real CARD}(n)) *_R 1 v * A) (\text{proj-rem } x)$   
**by** (*subst dot-lmul-matrix[symmetric]*) (*simp add: proj-unit-def stat-def*)  
**also have**  $\dots = (l * \text{inner } 1 x / \text{real CARD}(n)) * \text{inner } 1 (\text{proj-rem } x)$   
**unfolding** *scaleR-vector-matrix-assoc markov-apply[OF assms]* **by** *simp*  
**also have**  $\dots = 0$   
**unfolding** *proj-rem-orth* **by** *simp*  
**finally have**  $b: \text{inner} (l *_R \text{proj-unit } x) (A *v \text{proj-rem } x) = 0$  **by** *simp*  
  
**have**  $c: \text{inner} (\text{proj-rem } x) (\text{proj-unit } x) = 0$   
**using** *proj-rem-orth[of x]*  
**unfolding** *proj-unit-def stat-def* **by** (*simp add: inner-commute*)  
  
**have norm**  $(?R1)^2 = \text{norm} (A *v \text{proj-rem } x)^2 + \text{norm} (l *_R \text{proj-unit } x)^2$   
**using**  $b$  **by** (*intro pythagoras*) (*simp add: inner-commute*)  
**also have**  $\dots \leq (l * \text{norm} (\text{proj-rem } x))^2 + \text{norm} (l *_R \text{proj-unit } x)^2$   
**using** *proj-rem-orth[of x]*  
**by** (*intro add-mono power-mono spec-boundD2 a*) (*auto simp add: inner-commute*)  
**also have**  $\dots = l^2 * (\text{norm} (\text{proj-rem } x)^2 + \text{norm} (\text{proj-unit } x)^2)$   
**by** (*simp add: algebra-simps*)  
**also have**  $\dots = l^2 * (\text{norm} (\text{proj-rem } x + \text{proj-unit } x)^2)$   
**using**  $c$  **by** (*subst pythagoras*) *auto*  
**also have**  $\dots = l^2 * \text{norm } x^2$   
**by** (*subst (3) split-vec[of x]*) (*simp add: algebra-simps*)  
**also have**  $\dots = (l * \text{norm } x)^2$   
**by** (*simp add: algebra-simps*)  
**finally have**  $\text{norm} (?R1)^2 \leq (l * \text{norm } x)^2$  **by** *simp*  
**hence**  $\text{norm} (?R1) \leq l * \text{norm } x$   
**using**  $l\text{-ge-}0$  **by** (*subst (asm) power-mono-iff*) *auto*  
  
**thus**  $\text{norm} ((A - (1-l) *_R J) *v x) \leq l * \text{norm } x$   
**unfolding**  $d$  **by** *simp*  
**qed**  
**next**  
**assume**  $a: ?R$   
**have**  $\text{norm} (A *v x) \leq l * \text{norm } x$  **if**  $\text{inner } x 1 = 0$  **for**  $x$   
**proof** -  
**have**  $(1 - l) *_R J *v x = (1 - l) *_R (\text{proj-unit } x)$   
**by** (*simp add: vector-scaleR-matrix-ac-2 apply-J*)  
**also have**  $\dots = 0$   
**unfolding** *proj-unit-def* **using** *that* **by** (*simp add: inner-commute*)  
**finally have**  $b: (1 - l) *_R J *v x = 0$  **by** *simp*

**have**  $\text{norm } (A *v x) = \text{norm } ((A - (1-l) *R J) *v x + ((1-l) *R J) *v x)$   
**by** *(simp add:algebra-simps)*  
**also have**  $\dots \leq \text{norm } ((A - (1-l) *R J) *v x) + \text{norm } (((1-l) *R J) *v x)$   
**by** *(intro norm-triangle-ineq)*  
**also have**  $\dots \leq l * \text{norm } x + 0$   
**using** *a b unfolding matrix-norm-bound-def* **by** *(intro add-mono, auto)*  
**also have**  $\dots = l * \text{norm } x$   
**by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**moreover have**  $l \geq 0$   
**using** *a matrix-norm-bound-nonneg* **by** *blast*

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

**lemma** *matrix-decomposition-lemma:*

**fixes**  $A :: \text{real}^n \times \text{real}^n$

**assumes** *markov A*

**shows**  $\text{spec-bound } A \ l \longleftrightarrow (\exists E. A = (1-l) *R J + l *R E \wedge \text{matrix-norm-bound } E \ 1 \wedge l \geq 0)$

**(is** *?L*  $\longleftrightarrow$  *?R*)

**proof** –

**have**  $?L \longleftrightarrow \text{matrix-norm-bound } (A - (1-l) *R J) \ l$   
**using** *matrix-decomposition-lemma-aux[OF assms]* **by** *simp*

**also have**  $\dots \longleftrightarrow ?R$

**proof**

**assume** *a:matrix-norm-bound (A - (1 - l) \*R J) l*

**hence**  $l \geq 0$  **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**  $\text{matrix-norm-bound } (A - J) \ 0$

**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**  $\text{matrix-norm-bound } E \ 1$

**proof** (*cases l = 0*)

**case** *True*

**hence**  $E = 0$  **if**  $l = 0$

**unfolding** *E-def* **by** *simp*

**thus**  $\text{matrix-norm-bound } E \ 1$  **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**  $\text{matrix-norm-bound } E \ (|1 / l| * l)$

**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**  $E\text{-def}: A = (1 - l) *_R J + l *_R E$  *matrix-norm-bound*  $E$   $l \geq 0$   
**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**  $E\text{-def}$  **by** *simp*  
**moreover have**  $l *_R E = (A - (1 - l) *_R J)$   
**using**  $E\text{-def}(1)$  **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**  $S :: ('n :: \text{finite}) \text{ set}$   
**assumes** *l-range*:  $l \in \{0..1\}$   
**defines**  $P \equiv \text{diag } (\text{ind-vec } S)$   
**defines**  $\mu \equiv \text{card } S / \text{CARD}('n)$   
**assumes**  $\bigwedge M. M \in \text{set } Ms \implies \text{spec-bound } M \ l \ \wedge \ \text{markov } M$   
**shows** *foldl*  $(\lambda x M. P *v (M *v x)) (P *v \text{stat}) Ms \cdot 1 \leq (\mu + l * (1 - \mu))^{\wedge} (\text{length } Ms + 1)$

**proof** –

**define**  $t :: \text{real}^{\wedge} n$  **where**  $t = (\chi \ i. \ \text{of-bool } (i \in S))$   
**define**  $r$  **where**  $r = \text{foldl } (\lambda x M. P *v (M *v x)) (P *v \text{stat}) Ms$   
**have** *P-proj*:  $P ** P = P$   
**unfolding**  $P\text{-def}$  *diag-mult-eq ind-vec-def* **by** (*intro arg-cong[where f=diag]*) (*vector*)

**have** *P-1-left*:  $1 *v P = t$   
**unfolding**  $P\text{-def}$  *diag-def ind-vec-def vector-matrix-mult-def t-def* **by** *simp*

**have** *P-1-right*:  $P *v 1 = t$   
**unfolding**  $P\text{-def}$  *diag-def ind-vec-def matrix-vector-mult-def t-def* **by** *simp*

**have** *P-norm* : *matrix-norm-bound*  $P$   $1$   
**unfolding**  $P\text{-def}$  *ind-vec-def* **by** (*intro matrix-norm-bound-diag*) *simp*

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

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

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

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

**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 ∧ markov M*  
**using** *snoc(2)* **by** *simp*  
**then obtain** *E* **where** *E-def: M = (1-l) \*<sub>R</sub> J + l \*<sub>R</sub> E matrix-norm-bound E 1*  
**using** *iffD1[OF matrix-decomposition-lemma]* **by** *auto*  
  
**define** *y* **where** *y = foldl (λx 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)\*<sub>R</sub> J \*v y) + P \*v (l \*<sub>R</sub> E \*v y))*  
**by** (*simp add:E-def algebra-simps*)  
**also have** *... ≤ norm (P \*v ((1-l)\*<sub>R</sub> J \*v y)) + norm (P \*v (l \*<sub>R</sub> 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 · (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 · 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** *... ≤ (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) \* μ \* norm y + l \* norm (E \*v y)*  
**unfolding** *μ-def norm-t* **by** *simp*  
**also have** *... ≤ (1-l) \* μ \* norm y + l \* (1 \* norm y)*  
**using** *μ-range l-range*  
**by** (*intro add-mono matrix-norm-boundD mult-left-mono E-def*) *auto*  
**also have** *... = (μ + l \* (1-μ)) \* norm y*  
**by** (*simp add:algebra-simps*)  
**also have** *... ≤ (μ + l \* (1-μ)) \* ((μ + l \* (1-μ))<sup>length xs</sup> \* sqrt (card S)/CARD('n))*  
**using** *b μ-range l-range unfolding condition-def*  
**by** (*intro mult-left-mono*) *auto*  
**also have** *... = (μ + l \* (1-μ))<sup>length xs + 1</sup> \* sqrt (card S)/CARD('n)*  
**by** *simp*  
**finally have** *norm (P \*v (M \*v y)) ≤ (μ + l \* (1-μ))<sup>length xs + 1</sup> \* 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**  $\dots = inner\ (1\ v * P)\ r$   
**unfolding**  $dot-lmul-matrix$  **by**  $(simp\ add:inner-commute)$   
**also have**  $\dots = inner\ t\ r$   
**unfolding**  $P-1-left$  **by**  $simp$   
**also have**  $\dots \leq norm\ t * norm\ r$   
**by**  $(intro\ norm-cauchy-schwarz)$   
**also have**  $\dots \leq sqrt\ (card\ S) * ((\mu + l * (1-\mu))^{\wedge}(length\ Ms) * sqrt(card\ S) / CARD('n))$   
**using**  $a\ unfolding\ condition-def\ norm-t$   
**by**  $(intro\ mult-mono)\ auto$   
**also have**  $\dots = (\mu + 0) * ((\mu + l * (1-\mu))^{\wedge}(length\ Ms))$   
**by**  $(simp\ add:\mu-def)$   
**also have**  $\dots \leq (\mu + l * (1-\mu)) * (\mu + l * (1-\mu))^{\wedge}(length\ Ms)$   
**using**  $\mu-range\ l-range$   
**by**  $(intro\ mult-right-mono\ zero-le-power\ add-mono)\ auto$   
**also have**  $\dots = (\mu + l * (1-\mu))^{\wedge}(length\ Ms+1)$  **by**  $simp$   
**finally show**  $?thesis$   
**unfolding**  $r-def$  **by**  $simp$   
**qed**

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

**definition**  $bool-list-split :: bool\ list \Rightarrow (nat\ list \times nat)$   
**where**  $bool-list-split\ xs = foldl\ (\lambda(ys,z)\ x.\ (if\ x\ then\ (ys@[z],0)\ else\ (ys,z+1)))\ ([],0)\ xs$

**lemma**  $bool-list-split$ :  
**assumes**  $bool-list-split\ xs = (ys,z)$   
**shows**  $xs = concat\ (map\ (\lambda k.\ replicate\ k\ False@[True])\ ys)@replicate\ z\ False$   
**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**  $xs@[x] = concat\ (map\ (\lambda k.\ replicate\ k\ False@[True])\ u)@replicate\ v\ False@[True]$   
**using**  $snoc(1)[OF\ uv-def]\ True$  **by**  $simp$   
**also have**  $\dots = concat\ (map\ (\lambda k.\ replicate\ k\ False@[True])\ (u@[v]))@replicate\ 0\ False$   
**by**  $simp$   
**also have**  $\dots = concat\ (map\ (\lambda k.\ replicate\ k\ False@[True])\ (ys))@replicate\ z\ False$   
**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**  $xs@[x] = concat\ (map\ (\lambda k.\ replicate\ k\ False@[True])\ u)@replicate\ (v+1)\ False$

```

    using snoc(1)[OF uv-def] False unfolding replicate-add by simp
  also have ... = concat (map ( $\lambda k. \text{replicate } k \text{ False}@[\text{True}]$ ) (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 ( $\lambda y \text{ xs}. \text{foldl } f \ y \ \text{xs}$ ) 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  $\alpha$ -range:  $\alpha \in \{0..1\}$ 
assumes  $I \subseteq \{.. $l\}$ 
defines P i  $\equiv$  (if  $i \in I$  then diag (ind-vec S) else mat 1)
defines  $\mu \equiv \text{real } (\text{card } S) / \text{real } (\text{CARD } ('n))$ 
assumes spec-bound M  $\alpha$  markov M
shows
  foldl ( $\lambda x \ M. \ M * v \ x$ ) stat (intersperse M (map P [0.. $l$ ]))  $\cdot 1 \leq (\mu + \alpha * (1 - \mu))^{\text{card } I}$ 
  (is ?L  $\leq$  ?R)$ 
```

**proof** (cases  $I \neq \{\}$ )

```

case True
define xs where xs = map ( $\lambda i. i \in I$ ) [0.. $l$ ]
define Q where Q = diag (ind-vec S)
define P' where P' = ( $\lambda x. \text{if } x \text{ then } Q \text{ else mat } 1$ )

```

```

let ?rep = ( $\lambda x. \text{replicate } x \ (\text{mat } 1)$ )

```

```

have P-eq: P i = P' (i  $\in$  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  $\neq$  []
  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  $\cap$  {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  $a$ :  $\text{foldl } (\lambda x N. M *v (N *v x)) x (?rep z) \cdot 1 = x \cdot 1$  **for**  $x$   
**proof** (*induction z*)  
  case 0  
  then show ?case **by simp**  
next  
  case (*Suc z*)  
  have  $\text{foldl } (\lambda x N. M *v (N *v x)) x (?rep (z+1)) \cdot 1 = x \cdot 1$   
  **unfolding replicate-add using Suc**  
  **by (simp add: markov-orth-inv[OF assms(6)])**  
  then show ?case **by simp**  
**qed**

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

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

have  $d$ : *spec-bound*  $N \alpha \wedge$  *markov*  $N$  **if**  $t1$ :  $N \in \text{set } (\text{map } (\lambda x. \text{matrix-pow } M (x + 1)) yt)$  **for**  $N$   
**proof** –  
  **obtain**  $y$  **where**  $N\text{-def}$ :  $N = \text{matrix-pow } M (y+1)$   
  **using**  $t1$  **by auto**  
  **hence**  $d1$ : *spec-bound*  $N (\alpha \hat{\sim}(y+1))$   
  **unfolding N-def using spec-bound-pow assms(5,6) by blast**  
  **have** *spec-bound*  $N (\alpha \hat{\sim}1)$   
  **using**  $\alpha$ -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  $?L = \text{foldl } (\lambda x M. M *v x) \text{stat} (\text{intersperse } M (\text{map } P' xs)) \cdot 1$   
**unfolding P-eq xs-def map-map by (simp add: comp-def)**  
**also have**  $\dots = \text{foldl } (\lambda x M. M *v x) \text{stat} (\text{intersperse } M (\text{map } P' xs) @ [M]) \cdot 1$   
**by (simp add: markov-orth-inv[OF assms(6)])**  
**also have**  $\dots = \text{foldl } (\lambda x N. M *v (N *v x)) \text{stat} (\text{map } P' xs) \cdot 1$   
**using xs-ne by (subst foldl-intersperse) auto**  
**also have**  $\dots = \text{foldl } (\lambda x N. M *v (N *v x)) \text{stat} ((ys \gg (\lambda x. ?rep x @ [Q])) @ ?rep z) \cdot 1$   
**unfolding bool-list-split[OF ys-z] P'-def List.bind-def by (simp add: comp-def map-concat)**  
**also have**  $\dots = \text{foldl } (\lambda x N. M *v (N *v x)) \text{stat} (ys \gg (\lambda x. ?rep x @ [Q])) \cdot 1$   
**by (simp add: a)**  
**also have**  $\dots = \text{foldl } (\lambda x N. M *v (N *v x)) \text{stat} (?rep yh @ [Q] @ (yt \gg (\lambda x. ?rep x @ [Q]))) \cdot 1$

**unfolding** *ys-split* **by** *simp*  
**also have** ... = *foldl* ( $\lambda x N. M *v (N *v x)$ ) *stat* ( $[Q]@ (yt \gg (\lambda x. ?rep x @ [Q]))$ )  $\cdot 1$   
**by** (*simp add:b*)  
**also have** ... = *foldl* ( $\lambda x N. N *v x$ ) *stat* (*intersperse*  $M (Q\#(yt \gg (\lambda x. ?rep x @ [Q])))@ [M]$ )  $\cdot 1$   
**by** (*subst foldl-intersperse, auto*)  
**also have** ... = *foldl* ( $\lambda x N. N *v x$ ) *stat* (*intersperse*  $M (Q\#(yt \gg (\lambda x. ?rep x @ [Q])))$ )  $\cdot 1$   
**by** (*simp add:markov-orth-inv[OF assms(6)]*)  
**also have** ... = *foldl* ( $\lambda x N. N *v (M *v x)$ ) ( $Q *v \text{stat}$ ) ( $yt \gg (\lambda x. ?rep x @ [Q])$ )  $\cdot 1$   
**by** (*subst foldl-intersperse-2, simp*)  
**also have** ... = *foldl* ( $\lambda a x. \text{foldl} (\lambda x N. N *v (M *v x)) a (?rep x @ [Q]) (Q *v \text{stat}) yt$ )  $\cdot 1$   
**unfolding** *List.bind-def foldl-concat foldl-map* **by** *simp*  
**also have** ... = *foldl* ( $\lambda a x. Q *v (\text{matrix-pow } M (x+1) *v a)$ ) ( $Q *v \text{stat}$ )  $yt$   $\cdot 1$   
**unfolding** *c* **by** *simp*  
**also have** ... = *foldl* ( $\lambda a N. Q *v (N *v a)$ ) ( $Q *v \text{stat}$ ) (*map* ( $\lambda x. \text{matrix-pow } M (x+1) yt$ ))  $\cdot 1$   
**by** (*simp add:foldl-map*)  
**also have** ...  $\leq (\mu + \alpha * (1 - \mu))^{\wedge (\text{length} (\text{map} (\lambda x. \text{matrix-pow } M (x+1) yt) + 1))}$   
**unfolding**  $\mu$ -*def*  $Q$ -*def* **by** (*intro hitting-property-alg  $\alpha$ -range d*) *simp*  
**also have** ... =  $(\mu + \alpha * (1 - \mu))^{\wedge (\text{length } ys)}$   
**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**  $?L = \text{stat} \cdot (1 :: \text{real}^{\wedge n})$   
**proof** (*cases l > 0*)  
**case** *True*  
**have**  $?L = \text{foldl} (\lambda x M. M *v x) \text{stat} ((\text{intersperse } M (\text{map } P [0..<l]))@ [M]) \cdot 1$   
**by** (*simp add:markov-orth-inv[OF assms(6)]*)  
**also have** ... = *foldl* ( $\lambda x N. M *v (N *v x)$ ) *stat* (*map*  $P [0..<l]$ )  $\cdot 1$   
**using** *True* **by** (*subst foldl-intersperse, auto*)  
**also have** ... = *foldl* ( $\lambda x N. M *v (N *v x)$ ) *stat* (*map* ( $\lambda -. \text{mat } 1$ )  $[0..<l]$ )  $\cdot 1$   
**unfolding** *P-def* **using** *I-empty* **by** *simp*  
**also have** ... = *foldl* ( $\lambda x -. M *v x$ ) *stat*  $[0..<l]$   $\cdot 1$   
**unfolding** *foldl-map* **by** *simp*  
**also have** ... =  $\text{stat} \cdot (1 :: \text{real}^{\wedge n})$   
**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** ...  $\leq ?R$  **unfolding** *I-empty* **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**  
  
**lemma** *uniform-property-alg*:  
**fixes**  $x :: ('n :: \text{finite})$  **and**  $l :: \text{nat}$   
**assumes**  $i < l$   
**defines**  $P j \equiv (\text{if } j = i \text{ then } \text{diag} (\text{ind-vec } \{x\}) \text{ else } \text{mat } 1)$   
**assumes** *markov*  $M$   
**shows** *foldl* ( $\lambda x M. M *v x$ ) *stat* (*intersperse*  $M (\text{map } P [0..<l])$ )  $\cdot 1 = 1 / \text{CARD}('n)$   
**(is**  $?L = ?R$ **)**  
**proof** –

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

**have**  $0$ : *foldl*  $(\lambda x N. M *v (N *v x)) y (xs) \cdot 1 = y \cdot 1$  **if** *set*  $xs \subseteq \{mat\ 1\}$  **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*  $(\lambda x N. M *v (N *v x)) y (xs @ [x]) \cdot 1 = foldl (\lambda x N. M *v (N *v x)) y xs \cdot 1$   
**by** (*simp add:markov-orth-inv[OF assms(3)]*)

**also have**  $\dots = y \cdot 1$

**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*  $(\lambda x N. M *v (N *v x)) stat xs$ ) = *stat* **if** *set*  $xs \subseteq \{mat\ 1\}$  **for**  $xs$   
**using** *that* **by** (*induction xs, auto*)

**have**  $?L = foldl (\lambda x M. M *v x) stat ((intersperse M (map P [0..<l]))@[M]) \cdot 1$   
**by** (*simp add:markov-orth-inv[OF assms(3)]*)

**also have**  $\dots = foldl (\lambda x N. M *v (N *v x)) stat (map P [0..<l]) \cdot 1$   
**using** *a* **by** (*subst foldl-intersperse*) *auto*

**also have**  $\dots = foldl (\lambda x N. M *v (N *v x)) stat (map P ([0..<i+1]@[i+1..<l])) \cdot 1$   
**using** *assms(1)* **by** (*subst upto-append*) *auto*

**also have**  $\dots = foldl (\lambda x N. M *v (N *v x)) stat (map P [0..<i + 1]) \cdot 1$   
**unfolding** *map-append foldl-append P-def* **by** (*subst 0*) *auto*

**also have**  $\dots = foldl (\lambda x N. M *v (N *v x)) stat (map P ([0..<i]@[i])) \cdot 1$   
**by** *simp*

**also have**  $\dots = (M *v (diag (ind-vec \{x\}) *v stat)) \cdot 1$   
**unfolding** *map-append foldl-append P-def* **by** (*subst 1*) *auto*

**also have**  $\dots = (diag (ind-vec \{x\}) *v stat) \cdot 1$   
**by** (*simp add:markov-orth-inv[OF assms(3)]*)

**also have**  $\dots = ((1/CARD('n)) *R ind-vec \{x\}) \cdot 1$   
**unfolding** *diag-def ind-vec-def stat-def matrix-vector-mult-def*  
**by** (*intro arg-cong2[where f=(\cdot)] refl*)  
*(vector of-bool-def sum.If-cases if-distrib if-distribR)*

**also have**  $\dots = (1/CARD('n)) * (ind-vec \{x\} \cdot 1)$   
**by** *simp*

**also have**  $\dots = (1/CARD('n)) * 1$   
**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**  $M_s :: (('r::\{semiring-1, comm-monoid-mult\}) \wedge a \wedge a)$  *list*

**shows** (*foldl*  $(\lambda x M. M *v x) a M_s$ )  $\$ k = (\sum x \mid length\ x = length\ M_{s+1} \wedge x!\ length\ M_s = k.$   
 $(\prod_{i < length\ M_s. (M_s ! i) \$ (x ! (i+1)) \$ (x ! i)) * a \$ (x ! 0))$

**proof** (*induction M\_s arbitrary: k rule:rev-induct*)

**case** *Nil*

**have**  $\text{length } x = \text{Suc } 0 \implies x = [x!0]$  **for**  $x :: 'a \text{ list}$   
**by** (*cases x, auto*)  
**hence**  $\{x. \text{length } x = \text{Suc } 0 \wedge x!0 = k\} = \{[k]\}$   
**by** *auto*  
**thus** *?case by auto*  
**next**  
**case** (*snoc M Ms*)  
**let**  $?l = \text{length } Ms$   
  
**have**  $0: \text{finite } \{w. \text{length } w = \text{Suc } (\text{length } Ms) \wedge w! \text{length } Ms = i\}$  **for**  $i :: 'a$   
**using** *finite-lists-length-eq* **where**  $A = \text{UNIV} :: 'a \text{ set}$  **and**  $n = ?l + 1$  **by** *simp*  
  
**have**  $\text{take } (?l+1) x @ [x! (?l+1)] = x$  **if**  $\text{length } x = ?l+2$  **for**  $x :: 'a \text{ list}$   
**proof** –  
**have**  $\text{take } (?l+1) x @ [x! (?l+1)] = \text{take } (\text{Suc } (?l+1)) x$   
**using** *that by (intro take-Suc-conv-app-nth[symmetric], simp)*  
**also have**  $\dots = x$   
**using** *that by simp*  
**finally show** *?thesis by simp*  
**qed**  
**hence**  $1: \text{bij-betw } (\text{take } (?l+1)) \{w. \text{length } w = ?l+2 \wedge w!(?l+1) = k\} \{w. \text{length } w = ?l+1\}$   
**by** (*intro bij-betwI[where g =  $\lambda x. x@[k]$ ]*) (*auto simp add: nth-append*)  
  
**have**  $\text{foldl } (\lambda x M. M * v x) a (Ms @ [M]) \$ k = (\sum j \in \text{UNIV}. M \$ k \$ j * (\text{foldl } (\lambda x M. M * v x) a Ms \$ j))$   
**by** (*simp add: matrix-vector-mult-def*)  
**also have**  $\dots =$   
 $(\sum j \in \text{UNIV}. M \$ k \$ j * (\sum w | \text{length } w = ?l+1 \wedge w! ?l = j. (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0))$   
**unfolding** *snoc by simp*  
**also have**  $\dots =$   
 $(\sum j \in \text{UNIV}. (\sum w | \text{length } w = ?l+1 \wedge w! ?l = j. M \$ k \$ w! ?l * (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0))$   
**by** (*intro sum.cong refl*) (*simp add: sum-distrib-left algebra-simps*)  
**also have**  $\dots = (\sum w \in (\bigcup j \in \text{UNIV}. \{w. \text{length } w = ?l+1 \wedge w! ?l = j\}). M \$ k \$ w! ?l * (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**using**  $0$  **by** (*subst sum.UNION-disjoint, simp, simp*) *auto*  
**also have**  $\dots = (\sum w | \text{length } w = ?l+1. M \$ k \$ (w! ?l) * (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**by** (*intro sum.cong arg-cong2[where f = (\*)] refl*) *auto*  
**also have**  $\dots = (\sum w \in \text{take } (?l+1) ' \{w. \text{length } w = ?l+2 \wedge w!(?l+1) = k\}. M \$ k \$ w! ?l * (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**using**  $1$  **unfolding** *bij-betw-def by (intro sum.cong refl, auto)*  
**also have**  $\dots = (\sum w | \text{length } w = ?l+2 \wedge w!(?l+1) = k. M \$ k \$ w! ?l * (\prod i < ?l. Ms!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**using**  $1$  **unfolding** *bij-betw-def by (subst sum.reindex, auto)*  
**also have**  $\dots = (\sum w | \text{length } w = ?l+2 \wedge w!(?l+1) = k. (Ms @ [M])! ?l \$ k \$ w! ?l * (\prod i < ?l. (Ms @ [M])!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**by** (*intro sum.cong arg-cong2[where f = (\*)] prod.cong refl*) (*auto simp add: nth-append*)  
**also have**  $\dots = (\sum w | \text{length } w = ?l+2 \wedge w!(?l+1) = k. (\prod i < (?l+1). (Ms @ [M])!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**by** (*intro sum.cong, auto simp add: algebra-simps*)  
**finally have**  $\text{foldl } (\lambda x M. M * v x) a (Ms @ [M]) \$ k =$   
 $(\sum w | \text{length } w = ?l+2 \wedge w!(?l+1) = k. (\prod i < (?l+1). (Ms @ [M])!i \$ w!(i+1) \$ w!i) * a \$ w!0)$   
**by** *simp*  
**then show** *?case by simp*  
**qed**

**lemma** *foldl-matrix-mult-expand-2*:  
**fixes**  $Ms :: (\text{real}^{\wedge} a^{\wedge} a)$  list  
**shows**  $(\text{foldl } (\lambda x M. M * v x) a Ms) \cdot 1 = (\sum x \mid \text{length } x = \text{length } Ms + 1. (\prod_{i < \text{length } Ms. (Ms ! i) \$ (x ! (i+1)) \$ (x ! i)) * a \$ (x ! 0))$   
**(is**  $?L = ?R$ )  
**proof** –  
**let**  $?l = \text{length } Ms$   
**have**  $?L = (\sum j \in UNIV. (\text{foldl } (\lambda x M. M * v x) a Ms) \$ j)$   
**by** (*simp add:inner-vec-def*)  
**also have**  $\dots = (\sum j \in UNIV. \sum x \mid \text{length } x = ?l + 1 \wedge x ! ?l = j. (\prod_{i < ?l. Ms ! i \$ x ! (i+1) \$ x ! i} * a \$ (x ! 0))$   
 $\$ x ! 0)$   
**unfolding** *foldl-matrix-mult-expand* **by** *simp*  
**also have**  $\dots = (\sum x \in (\bigcup j \in UNIV. \{w. \text{length } w = \text{length } Ms + 1 \wedge w ! \text{length } Ms = j\}). (\prod_{i < \text{length } Ms. (Ms ! i) \$ (x ! (i+1)) \$ (x ! i)} * a \$ (x ! 0))$   
**using** *finite-lists-length-eq* **where**  $A = UNIV :: 'a$  set **and**  $n = ?l + 1$   
**by** (*intro sum.UNION-disjoint[symmetric]*) *auto*  
**also have**  $\dots = ?R$   
**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 parameters 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**  $\langle \cdot \rangle$  70)  
**no-notation** *Ordered-Semiring.max* ( $\langle \text{Max} \rangle$ )

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

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

shows  $x \in\# xs$   
 using *assms* by (*induction xs, auto*)

lemma *poly-prod-inj-aux-1*:

fixes  $xs\ ys :: ('a :: idom)\ multiset$   
 assumes  $x \in\# xs$   
 assumes  $(\prod a \in\# xs. [-\ a, 1:]) = (\prod a \in\# ys. [-\ a, 1:])$   
 shows  $x \in\# ys$

proof –

have  $poly (\prod a \in\# ys. [-\ a, 1:])\ x = poly (\prod a \in\# xs. [-\ a, 1:])\ x$  using *assms(2)* by *simp*

also have  $\dots = poly (\prod a \in\# xs - \{\#x\} + \{\#x\}. [-\ a, 1:])\ x$

using *assms(1)* by *simp*

also have  $\dots = 0$

by *simp*

finally have  $poly (\prod a \in\# ys. [-\ a, 1:])\ x = 0$  by *simp*

thus  $x \in\# ys$  using *poly-prod-zero* by *blast*

qed

lemma *poly-prod-inj-aux-2*:

fixes  $xs\ ys :: ('a :: idom)\ multiset$   
 assumes  $x \in\# xs \cup\# ys$   
 assumes  $(\prod a \in\# xs. [-\ a, 1:]) = (\prod a \in\# ys. [-\ a, 1:])$   
 shows  $x \in\# xs \cap\# ys$

proof (*cases x \in\# xs*)

case *True*

then show *?thesis* using *poly-prod-inj-aux-1[OF True assms(2)]* by *simp*

next

case *False*

hence  $a:x \in\# ys$

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  $(\prod a \in\# xs. [-\ a, 1:]) = (\prod a \in\# ys. [-\ a, 1:])$   
 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 \cup\# ys = \{\#\}*)

case *True*

then show *?thesis* by *simp*

next

case *False*

then obtain  $x$  where  $x \in\# xs \cup\# ys$  by *auto*

hence  $a:x \in\# xs \cap\# ys$

by (*intro poly-prod-inj-aux-2[OF - 1(2)]*)

have  $b: [-\ x, 1:] \neq 0$

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  $[-\ x, 1:] * (\prod a \in\# xs - \{\#x\}. [-\ a, 1:]) = (\prod a \in\# xs. [-\ a, 1:])$

using  $a$  by (*subst prod-mset.insert[symmetric]*) *simp*

also have  $\dots = (\prod a \in\# ys. [-\ a, 1:])$  using  $1$  by *simp*

**also have** ... =  $[: - x, 1:] * (\prod a \in \#ys - \{\#x\}. [: - a, 1:])$   
**using** *a* by (*subst prod-mset.insert[symmetric]*) *simp*  
**finally have**  $[: - x, 1:] * (\prod a \in \#xs - \{\#x\}. [: - a, 1:]) = [: - x, 1:] * (\prod a \in \#ys - \{\#x\}. [: - a, 1:])$   
*1:]*  
**by** *simp*  
**hence**  $(\prod a \in \#xs - \{\#x\}. [: - a, 1:]) = (\prod a \in \#ys - \{\#x\}. [: - a, 1:])$   
**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* ::  $('a::comm-ring-1)^{\wedge}n^{\wedge}n \Rightarrow 'a$  multiset  
**where**

*eigenvalues* *A* =  $(SOME as. charpoly A = (\prod a \in \#as. [: - a, 1:]) \wedge size as = CARD ('n))$

**lemma** *char-poly-factorized-hma*:

**fixes** *A* ::  $complex^{\wedge}n^{\wedge}n$

**shows**  $\exists as. charpoly A = (\prod a \leftarrow as. [: - a, 1:]) \wedge length as = CARD ('n)$

**by** (*transfer-hma rule:char-poly-factorized*)

**lemma** *eigvals-poly-length*:

**fixes** *A* ::  $complex^{\wedge}n^{\wedge}n$

**shows**

$charpoly A = (\prod a \in \#eigenvalues A. [: - a, 1:])$  (**is** *?A*)

$size (eigenvalues A) = CARD ('n)$  (**is** *?B*)

**proof** –

**define** *f* **where**  $f as = (charpoly A = (\prod a \in \#as. [: - a, 1:]) \wedge size as = CARD ('n))$  **for** *as*

**obtain** *as* **where** *as-def*:  $charpoly A = (\prod a \leftarrow as. [: - a, 1:]) \wedge length as = CARD ('n)$

**using** *char-poly-factorized-hma* **by** *auto*

**have**  $charpoly A = (\prod a \leftarrow as. [: - a, 1:])$

**unfolding** *as-def* **by** *simp*

**also have** ... =  $(\prod a \in \#mset as. [: - a, 1:])$

**unfolding** *prod-mset-prod-list[symmetric]* *mset-map* **by** *simp*

**finally have**  $charpoly A = (\prod a \in \#mset as. [: - a, 1:])$  **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* ::  $complex^{\wedge}n^{\wedge}n$

**assumes** *similar-matrix A B*

**shows**  $eigenvalues A = eigenvalues B$

**proof** –

**have**  $(\prod a \in \# \text{eigenvalues } A. [- a, 1:]) = (\prod a \in \# \text{eigenvalues } B. [- a, 1:])$   
**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* ::  $'a :: \text{zero} \wedge 'n \wedge 'n \Rightarrow \text{bool}$   
**where** *upper-triangular-hma*  $A \equiv$   
 $\forall i. \forall j. (\text{to-nat } j < \text{Bij-Nat.to-nat } i \longrightarrow A \$h i \$h j = 0)$

**lemma** *for-all-reindex2*:  
**assumes**  $\text{range } f = A$   
**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**  $A :: ('a :: \text{zero}) \wedge 'n \wedge 'n$   
**shows**  $\text{upper-triangular } (\text{from-hma}_m A) = \text{upper-triangular-hma } A$  (**is**  $?L = ?R$ )

**proof** –

**have**  $?L \longleftrightarrow (\forall i \in \{0..< \text{CARD}('n)\}. \forall j \in \{0..< \text{CARD}('n)\}. j < i \longrightarrow A \$h \text{from-nat } i \$h \text{from-nat } j = 0)$   
**unfolding** *upper-triangular-def* *from-hma<sub>m</sub>-def* **by** *auto*  
**also have**  $\dots \longleftrightarrow (\forall (i :: 'n) (j :: 'n). \text{to-nat } j < \text{to-nat } i \longrightarrow A \$h \text{from-nat } (\text{to-nat } i) \$h \text{from-nat } (\text{to-nat } j) = 0)$   
**by** (*intro for-all-reindex2* *range-to-nat*[**where**  $'a = 'n$ ])  
**also have**  $\dots \longleftrightarrow ?R$   
**unfolding** *upper-triangular-hma-def* **by** *auto*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *from-hma-carrier*:  
**fixes**  $A :: 'a \wedge ('n :: \text{finite}) \wedge ('m :: \text{finite})$   
**shows**  $\text{from-hma}_m A \in \text{carrier-mat } (\text{CARD } ('m)) (\text{CARD } ('n))$   
**unfolding** *from-hma<sub>m</sub>-def* **by** *simp*

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

**lemma** *diag-mat-hma*:  
**fixes**  $A :: 'a \wedge 'n \wedge 'n$   
**shows**  $\text{mset } (\text{diag-mat } (\text{from-hma}_m A)) = \text{diag-mat-hma } A$  (**is**  $?L = ?R$ )

**proof** –

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

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

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

**lemma** *adjoint-hma*:  
**fixes**  $A :: \text{complex}^{\wedge('n::\text{finite})} \wedge('m::\text{finite})$   
**shows**  $\text{mat-adjoint } (\text{from-hma}_m \ A) = \text{from-hma}_m \ (\text{adjoint-hma } A)$   
**proof** –  
**have**  $\text{mat-adjoint } (\text{from-hma}_m \ A) \ \$\$ \ (i,j) = \text{from-hma}_m \ (\text{adjoint-hma } A) \ \$\$ \ (i,j)$   
**if**  $i < \text{CARD}'n \ j < \text{CARD}'m$  **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**  $\text{cinner } v \ w = \text{scalar-product } v \ (\text{map-vector } \text{cnj } w)$

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

**lemma** *cinner-hma*:  
**fixes**  $x \ y :: \text{complex}^{\wedge'n}$   
**shows**  $\text{cinner } x \ y = (\text{from-hma}_v \ x) \cdot c \ (\text{from-hma}_v \ y)$  (**is**  $?L = ?R$ )  
**proof** –  
**have**  $?L = (\sum_{i \in \text{UNIV}} x \ \$h \ i * \text{cnj } (y \ \$h \ i))$   
**unfolding** *cinner-def map-vector-def scalar-product-def* **by** *simp*  
**also have**  $\dots = (\sum_{i = 0..<\text{CARD}'n} x \ \$h \ \text{from-nat } i * \text{cnj } (y \ \$h \ \text{from-nat } i))$   
**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**  $\dots = ?R$   
**unfolding** *Matrix.scalar-prod-def from-hma\_v-def*  
**by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *cinner-hma-transfer[transfer-rule]*:  
 $(\text{HMA-V} \ ==\Rightarrow \ \text{HMA-V} \ ==\Rightarrow \ (=)) \ (\cdot c) \ \text{cinner}$   
**unfolding** *HMA-V-def cinner-hma*  
**by** (*auto simp:rel-fun-def*)

**lemma** *adjoint-hma-transfer[transfer-rule]*:  
 $(\text{HMA-M} \ ==\Rightarrow \ \text{HMA-M}) \ (\text{mat-adjoint}) \ \text{adjoint-hma}$   
**unfolding** *HMA-M-def rel-fun-def* **by** (*auto simp add:adjoint-hma*)

**end**

**lemma** *adjoint-adjoint-id[simp]*:  $\text{adjoint-hma } (\text{adjoint-hma } A) = A$   
**by** (*transfer*) (*simp add:adjoint-adjoint*)

**lemma** *adjoint-def-alter-hma*:  
 $\text{cinner } (A * v \ v) \ w = \text{cinner } v \ (\text{adjoint-hma } A * v \ w)$   
**by** (*transfer-hma rule:adjoint-def-alter*)

**lemma** *cinner-0*:  $\text{cinner } 0 \ 0 = 0$   
**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* ::  $complex^{n^2} \Rightarrow bool$   
**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* ::  $(a::zero)^{n^2} \Rightarrow bool$  **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*  $A \implies upper-triangular-hma A$   
**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* ::  $complex^{n^2} \Rightarrow bool$  **where**  
*hermitian-hma*  $A = (adjoint-hma A = A)$

**lemma** *from-hma-one*:  
*from-hma<sub>m</sub>* (*mat* 1 ::  $((a::\{one,zero\})^{n^2}) = 1_m CARD(n)$ )  
**unfolding** *Finite-Cartesian-Product.mat-def from-hma<sub>m</sub>-def* **using** *from-nat-inj*  
**by** (*intro eq-matI*) *auto*

**lemma** *from-hma-mult*:  
**fixes**  $A :: (a :: semiring-1)^{m^2}$   
**fixes**  $B :: (a^{k^2} m::finite)$   
**shows** *from-hma<sub>m</sub>*  $A * from-hma<sub>m</sub> B = from-hma<sub>m</sub> (A ** B)$   
**using** *HMA-M-mult unfolding rel-fun-def HMA-M-def* **by** *auto*

**lemma** *hermitian-hma*:  
*hermitian-hma*  $A = hermitian (from-hma<sub>m</sub> A)$   
**unfolding** *hermitian-def adjoint-hma hermitian-hma-def* **by** *auto*

**lemma** *unitary-hma*:  
**fixes**  $A :: \text{complex}^{\wedge n} \wedge n$   
**shows**  $\text{unitary-hma } A = \text{unitary } (\text{from-hma}_m A)$  (**is**  $?L = ?R$ )  
**proof** –  
**have**  $?R \longleftrightarrow \text{from-hma}_m A * \text{mat-adjoint } (\text{from-hma}_m A) = 1_m$  ( $\text{CARD}(n)$ )  
**using** *from-hma-carrier*  
**unfolding** *unitary-def inverts-mat-def* **by** *simp*  
**also have**  $\dots \longleftrightarrow \text{from-hma}_m (A ** \text{adjoint-hma } A) = \text{from-hma}_m (\text{mat } 1 :: \text{complex}^{\wedge n} \wedge n)$   
**unfolding** *adjoint-hma from-hma-mult from-hma-one* **by** *simp*  
**also have**  $\dots \longleftrightarrow A ** \text{adjoint-hma } A = \text{Finite-Cartesian-Product.mat } 1$   
**unfolding** *from-hma<sub>m</sub>-inj* **by** *simp*  
**also have**  $\dots \longleftrightarrow ?L$  **unfolding** *unitary-hma-def* **by** *simp*  
**finally show** *?thesis* **by** *simp*

**qed**

**lemma** *unitary-hmaD*:  
**fixes**  $A :: \text{complex}^{\wedge n} \wedge n$   
**assumes** *unitary-hma*  $A$   
**shows**  $\text{adjoint-hma } A ** A = \text{mat } 1$  (**is**  $?A$ )  $A ** \text{adjoint-hma } A = \text{mat } 1$  (**is**  $?B$ )  
**proof** –  
**have**  $\text{mat-adjoint } (\text{from-hma}_m A) * \text{from-hma}_m A = 1_m$   $\text{CARD}(n)$   
**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<sub>m</sub>-inj*  
**by** *simp*  
**show**  $?B$   
**using** *assms* **unfolding** *unitary-hma-def* **by** *simp*

**qed**

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

**lemma** *unitarily-equiv-hma*:  
**fixes**  $A :: \text{complex}^{\wedge n} \wedge n$   
**shows**  $\text{unitarily-equiv-hma } A B U =$   
 $\text{unitarily-equiv } (\text{from-hma}_m A) (\text{from-hma}_m B) (\text{from-hma}_m U)$   
**(is**  $?L = ?R$ )

**proof** –

**have**  $?R \longleftrightarrow (\text{unitary-hma } U \wedge \text{similar-mat-wit } (\text{from-hma}_m A) (\text{from-hma}_m B) (\text{from-hma}_m U))$   
 $(\text{from-hma}_m (\text{adjoint-hma } U))$   
**unfolding** *Spectral-Theory-Complements.unitarily-equiv-def unitary-hma[symmetric]* *adjoint-hma*  
**by** *simp*  
**also have**  $\dots \longleftrightarrow \text{unitary-hma } U \wedge \text{similar-matrix-wit } A B U (\text{adjoint-hma } U)$   
**using** *HMA-similar-mat-wit* **unfolding** *rel-fun-def HMA-M-def*  
**by** (*intro arg-cong2[where f=( $\wedge$ )] refl*) *force*  
**also have**  $\dots \longleftrightarrow ?L$   
**unfolding** *unitarily-equiv-hma-def* **by** *auto*  
**finally show** *?thesis* **by** *simp*

**qed**

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

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

**lemma** *diagonal-mat-hma*:

**fixes**  $A :: ('a :: \text{zero}) \wedge ('n :: \text{finite}) \wedge n$

**shows**  $\text{diagonal-mat } A = \text{Matrix.diagonal-mat } (\text{from-hma}_m A)$  **(is**  $?L = ?R)$

**proof**

**show**  $?L \implies ?R$

**unfolding** *diagonal-mat-def* *Matrix.diagonal-mat-def* *from-hma<sub>m</sub>-def*

**using** *from-nat-inj* **by** *auto*

**next**

**assume**  $a: ?R$

**have**  $A \$h\ i\ \$h\ j = 0$  **if**  $i \neq j$  **for**  $i\ j$

**proof**  $-$

**have**  $A \$h\ i\ \$h\ j = (\text{from-hma}_m A) \$\$ (to-nat\ i, to-nat\ j)$

**unfolding** *from-hma<sub>m</sub>-def* **using** *to-nat-less-card* **[where**  $'a = 'n$  **]** **by** *simp*

**also have**  $\dots = 0$

**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**  $A :: \text{complex} \wedge 'n \wedge n$

**shows**  $\text{unitary-diag } A\ d\ U =$

$\text{Spectral-Theory-Complements.unitary-diag } (\text{from-hma}_m A) (\text{from-hma}_m (\text{diag } d)) (\text{from-hma}_m U)$

**proof**  $-$

**have**  $\text{Matrix.diagonal-mat } (\text{from-hma}_m (\text{diag } d))$

**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**  $A :: \text{complex} \wedge 'n \wedge n$

**shows**  $\text{real-diag-decomp-hma } A\ d\ U =$

$\text{real-diag-decomp } (\text{from-hma}_m A) (\text{from-hma}_m (\text{diag } d)) (\text{from-hma}_m U)$

**proof**  $-$

**have**  $0: (\forall i. d \$h\ i \in \mathbf{R}) \longleftrightarrow (\forall i < \text{CARD}'n. \text{from-hma}_m (\text{diag } d) \$\$ (i, i) \in \mathbf{R})$

**unfolding** *from-hma<sub>m</sub>-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**  $\text{Matrix.diagonal-mat } A\ A \in \text{carrier-mat } \text{CARD}'n\ \text{CARD } ('n :: \text{finite})$

**shows**  $\text{from-hma}_m (\text{diag } (\chi\ (i::'n). A \$\$ (to-nat\ i, to-nat\ i))) = A$

**using** *assms* *from-nat-inj* **unfolding** *from-hma<sub>m</sub>-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**  $A\ f :: (\text{complex} \wedge 'n \wedge 'n)$  *set*

```

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

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

    have  $U \in \text{carrier-mat } CARD('n) CARD('n)$   $Bmap (\text{from-hma}_m A) \in \text{carrier-mat } CARD('n)$ 
       $CARD('n)$ 
      Matrix.diagonal-mat  $(Bmap (\text{from-hma}_m A))$ 
      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  $(\text{from-hma}_m (\text{diag } (Bmap' A))) = Bmap (\text{from-hma}_m A) (\text{from-hma}_m U') = U$ 
      unfolding  $Bmap'\text{-def } U'\text{-def}$  by (auto simp add:diagonal-mat-diag-ex-hma)
    hence real-diag-decomp  $(\text{from-hma}_m A) (\text{from-hma}_m (\text{diag } (Bmap' A))) (\text{from-hma}_m U')$ 
      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  $A :: \text{complex}^{\sim n} \sim n$ 
  assumes upper-triangular-hma  $A$ 
  shows charpoly  $A = (\prod a \in \# \text{diag-mat-hma } A. [:- a, 1:])$ 
proof –
  have charpoly  $A = \text{char-poly } (\text{from-hma}_m A)$ 
    using HMA-char-poly unfolding rel-fun-def HMA-M-def
    by (auto simp add:eq-commute)

```

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

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

**lemma** *cinner-self*:  
**fixes**  $v :: \text{complex}^{\wedge n}$   
**shows** *cinner*  $v v = \text{norm } v^2$   
**proof** –  
**have**  $0: x * \text{cnj } x = \text{complex-of-real } (x \cdot x)$  **for**  $x :: \text{complex}$   
**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*  $(U * v v) = \text{norm } v$   
**proof** –  
**have** *norm*  $(U * v v)^2 = \text{cinner } (U * v v) (U * v v)$   
**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*  $(\text{norm } (U * v v)^2) = \text{norm } v^2$  **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*  $(A * v v) = \text{map-matrix hom } A * v \text{map-vector hom } v$   
**using** *mult-mat-vec-hom* **by** *transfer auto*

**lemma** (*in semiring-hom*) *mat-hom-mult-hma*:  
*map-matrix hom*  $(A ** B) = \text{map-matrix hom } A ** \text{map-matrix hom } B$   
**using** *mat-hom-mult* **by** *transfer auto*

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

**lemma** *to-nat-less-n*:  $to\text{-}nat\ (x::'n) < n$   
**using** *to-nat-less-card card-n* **by** *metis*

**lemma** *to-nat-from-nat*:  $x < n \implies to\text{-}nat\ (from\text{-}nat\ x :: 'n) = x$   
**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**  $v \in \text{verts } G$   
**shows**  $g\text{-}step\ (\lambda\cdot. 1)\ v = 1$  (**is**  $?L = ?R$ )  
**proof** –  
**have**  $?L = in\text{-}degree\ G\ v / d$   
**unfolding** *g-step-def in-degree-def* **by** *simp*  
**also have**  $\dots = 1$   
**unfolding** *reg(2)[OF assms]* **using** *d-gt-0* **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *markov*: *markov (A :: real<sup>n</sup><sup>n</sup>)*  
**proof** –  
**have**  $A * v\ 1 = (1::real\ ^n)$  (**is**  $?L = ?R$ )  
**proof** –  
**have**  $A * v\ 1 = (\chi\ i.\ g\text{-}step\ (\lambda\cdot. 1)\ (enum\text{-}verts\ i))$   
**unfolding** *g-step-conv one-vec-def* **by** *simp*  
**also have**  $\dots = (\chi\ i.\ 1)$   
**using** *bij-betw-apply[OF enum-verts]* **by** (*subst g-step-1*) *auto*  
**also have**  $\dots = 1$  **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-eivals*: *eigenvalues J = {#1::complex#} + replicate-mset (n - 1) 0*  
**proof** –  
**define**  $\alpha :: nat \Rightarrow real$  **where**  $\alpha\ i = sqrt\ (i^2+i)$  **for**  $i :: nat$

**define**  $q :: nat \Rightarrow nat \Rightarrow real$   
**where**  $q\ i\ j =$   
*if*  $i = 0$  *then*  $(1/sqrt\ n)$  *else* (  
*if*  $j < i$  *then*  $((-1) / \alpha\ i)$  *else* (  
*if*  $j = i$  *then*  $(i / \alpha\ i)$  *else*  $0$ )) **for**  $i\ j$

**define**  $Q :: complex^{n \times n}$  **where**  $Q = (\chi\ i\ j.\ complex\text{-of}\text{-}real\ (q\ (to\text{-}nat\ i)\ (to\text{-}nat\ j)))$

define  $D :: \text{complex}^{\wedge n} \wedge n$  where

$$D = (\chi \ i \ j. \text{if to-nat } i = 0 \wedge \text{to-nat } j = 0 \text{ then } 1 \text{ else } 0)$$

have 2:  $[0..<n] = 0 \# [1..<n]$

using  $n\text{-gt-0}$   $\text{upt-conv-Cons}$  by  $\text{auto}$

have  $\text{aux0}: (\sum k = 0..<n. q \ j \ k * q \ i \ k) = \text{of-bool } (i = j)$  if  $1:i \leq j < n$  for  $i \ j$

proof –

consider (a)  $i = j \wedge j = 0$  | (b)  $i = 0 \wedge i < j$  | (c)  $0 < i \wedge i < j$  | (d)  $0 < i \wedge i = j$   
 using 1 by  $\text{linarith}$

thus  $?thesis$

proof (cases)

case a

then show  $?thesis$  using  $n\text{-gt-0}$  by ( $\text{simp add:q-def}$ )

next

case b

have  $(\sum k = 0..<n. q \ j \ k * q \ i \ k) = (\sum k \in \text{insert } j \ (\{0..<j\} \cup \{j+1..<n\}). q \ j \ k * q \ i \ k)$   
 using  $\text{that}(2)$  by ( $\text{intro sum.cong}$ )  $\text{auto}$

also have  $\dots = q \ j \ j * q \ i \ j + (\sum k = 0..<j. q \ j \ k * q \ i \ k) + (\sum k = j+1..<n. q \ j \ k * q \ i \ k)$   
 by ( $\text{subst sum.insert}$ ) ( $\text{auto simp add: sum.union-disjoint}$ )

also have  $\dots = 0$  using b unfolding  $q\text{-def}$  by  $\text{simp}$

finally show  $?thesis$  using b by  $\text{simp}$

next

case c

have  $(\sum k = 0..<n. q \ j \ k * q \ i \ k) = (\sum k \in \text{insert } i \ (\{0..<i\} \cup \{i+1..<n\}). q \ j \ k * q \ i \ k)$   
 using  $\text{that}(2)$  c by ( $\text{intro sum.cong}$ )  $\text{auto}$

also have  $\dots = q \ j \ i * q \ i \ i + (\sum k = 0..<i. q \ j \ k * q \ i \ k) + (\sum k = i+1..<n. q \ j \ k * q \ i \ k)$   
 by ( $\text{subst sum.insert}$ ) ( $\text{auto simp add: sum.union-disjoint}$ )

also have  $\dots = (-1) / \alpha \ j * i / \alpha \ i + i * ((-1) / \alpha \ j * (-1) / \alpha \ i)$   
 using c unfolding  $q\text{-def}$  by  $\text{simp}$

also have  $\dots = 0$

by ( $\text{simp add:algebra-simps}$ )

finally show  $?thesis$  using c by  $\text{simp}$

next

case d

have  $\text{real } i + \text{real } i^{\wedge 2} = \text{real } (i + i^{\wedge 2})$  by  $\text{simp}$

also have  $\dots \neq \text{real } 0$

unfolding  $\text{of-nat-eq-iff}$  using d by  $\text{simp}$

finally have  $d-1: \text{real } i + \text{real } i^{\wedge 2} \neq 0$  by  $\text{simp}$

have  $(\sum k = 0..<n. q \ j \ k * q \ i \ k) = (\sum k \in \text{insert } i \ (\{0..<i\} \cup \{i+1..<n\}). q \ j \ k * q \ i \ k)$   
 using  $\text{that}(2)$  d by ( $\text{intro sum.cong}$ )  $\text{auto}$

also have  $\dots = q \ j \ i * q \ i \ i + (\sum k = 0..<i. q \ j \ k * q \ i \ k) + (\sum k = i+1..<n. q \ j \ k * q \ i \ k)$   
 by ( $\text{subst sum.insert}$ ) ( $\text{auto simp add: sum.union-disjoint}$ )

also have  $\dots = i / \alpha \ i * i / \alpha \ i + i * ((-1) / \alpha \ i * (-1) / \alpha \ i)$   
 using d that unfolding  $q\text{-def}$  by  $\text{simp}$

also have  $\dots = (i^{\wedge 2} + i) / (\alpha \ i)^{\wedge 2}$

by ( $\text{simp add: power2-eq-square divide-simps}$ )

also have  $\dots = 1$

using  $d-1$  unfolding  $\alpha\text{-def}$  by ( $\text{simp add:algebra-simps}$ )

finally show  $?thesis$  using d by  $\text{simp}$

qed

qed

have 0:  $(\sum k = 0..<n. q \ j \ k * q \ i \ k) = \text{of-bool } (i = j)$  (is  $?L = ?R$ ) if  $i < n \ j < n$  for  $i \ j$

proof –

have  $?L = (\sum k = 0..<n. q \ (\text{max } i \ j) \ k * q \ (\text{min } i \ j) \ k)$

by (cases  $i \leq j$ ) ( $\text{simp-all add:ac-simps cong:sum.cong}$ )

also have  $\dots = \text{of-bool } (\text{min } i \ j = \text{max } i \ j)$

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

**have**  $(\sum k \in UNIV. Q \ \$h \ j \ \$h \ k * cnj (Q \ \$h \ i \ \$h \ k)) = of\text{-}bool \ (i=j) \ (is \ ?L = ?R) \ \mathbf{for} \ i \ j$   
**proof** –  
**have** ?L = *complex-of-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-of-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*  $(of\text{-}bool(to\text{-}nat \ i = to\text{-}nat \ j))$   
**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**  $(adjoint\text{-}hma \ Q ** D ** Q) \ \$h \ i \ \$h \ j = J \ \$h \ i \ \$h \ j \ (\mathbf{is} \ ?L1 = ?R1) \ \mathbf{for} \ i \ j$   
**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** ... =  $1/\sqrt{real \ n}^2$   
**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-eivals*)  
**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..
   unfolding range-to-nat card-n mset-map by simp  
 also have ... = mset (1 # map (λi. 0) [1..
   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<sup>n</sup>)  
   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<sup>n</sup>) = 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=(\*)] 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<sup>n</sup><sup>n</sup>)

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*:  $J ** A = A ** (J :: \text{complex}^{\wedge n} \wedge n)$  (**is** *?L = ?R*)

**proof** –

**have**  $J ** A = \text{map-matrix complex-of-real } (J ** A)$   
**unfolding** *of-real-hom.mat-hom-mult-hma J-def A-def*  
**by** (*auto simp add:map-matrix-def map-vector-def*)  
**also have**  $\dots = \text{map-matrix complex-of-real } (A ** J)$   
**unfolding** *J-A-comm-real* **by** *simp*  
**also have**  $\dots = \text{map-matrix complex-of-real } A ** \text{map-matrix complex-of-real } J$   
**unfolding** *of-real-hom.mat-hom-mult-hma* **by** *simp*  
**also have**  $\dots = ?R$   
**unfolding** *A-def J-def*  
**by** (*auto simp add:map-matrix-def map-vector-def*)  
**finally show** *?thesis* **by** *simp*

**qed**

**definition**  $\gamma_a :: 'n \text{ itself} \Rightarrow \text{real}$  **where**

$\gamma_a - = (\text{if } n > 1 \text{ then Max-mset } (\text{image-mset cmod } (\text{eigenvalues } A - \{\#1\})) \text{ else } 0)$

**definition**  $\gamma_2 :: 'n \text{ itself} \Rightarrow \text{real}$  **where**

$\gamma_2 - = (\text{if } n > 1 \text{ then Max-mset } \{\# \text{Re } x. x \in \# (\text{eigenvalues } A - \{\#1\})\} \text{ else } 0)$

**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*:  $\text{set-mset } (\text{eigenvalues } A :: \text{complex multiset}) \subseteq \mathbb{R}$  (**is** *?R1*)  
**and** *ev-1*:  $(1 :: \text{complex}) \in \# \text{eigenvalues } A$   
**and**  *$\gamma_a$ -ge-0*:  $\gamma_a \text{ TYPE } ('n) \geq 0$   
**and** *find-any-ev*:  
 $\forall \alpha \in \# \text{eigenvalues } A - \{\#1\}. \exists v. \text{cinner } v \ 1 = 0 \wedge v \neq 0 \wedge A * v \ v = \alpha * s \ v$   
**and**  *$\gamma_a$ -bound*:  $\forall v. \text{cinner } v \ 1 = 0 \longrightarrow \text{norm } (A * v \ v) \leq \gamma_a \text{ TYPE } ('n) * \text{norm } v$   
**and**  *$\gamma_2$ -bound*:  $\forall (v :: \text{real}^{\wedge n}). v \cdot 1 = 0 \longrightarrow v \cdot (A * v \ v) \leq \gamma_2 \text{ TYPE } ('n) * \text{norm } v^{\wedge 2}$

**proof** –

**have**  $\exists U. \forall A \in \{J, A\}. \exists B. \text{real-diag-decomp-hma } A \ B \ U$   
**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*  $(\text{diag } Jd) = \text{eigenvalues } (\text{diag } Jd)$   
**by** (*intro upper-tri-eigvals[symmetric] diag-imp-upper-tri J-sim*) *auto*  
**also have**  $\dots = \text{eigenvalues } J$   
**using** *J-sim* **by** (*intro similar-matrix-eigvals[symmetric]*) (*auto simp add:similar-matrix-def*)  
**also have**  $\dots = \{\#1 :: \text{complex}\} + \text{replicate-mset } (n - 1) \ 0$   
**unfolding** *J-eigvals* **by** *simp*

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

**define**  $u$  **where**  $u = \text{adjoint-hma } U *v 1$   
**define**  $\alpha$  **where**  $\alpha = u \$h i$

**have**  $U *v u = (U ** \text{adjoint-hma } U) *v 1$   
**unfolding**  $u\text{-def}$  **by**  $(\text{simp add:matrix-vector-mul-assoc})$   
**also have**  $\dots = 1$   
**unfolding**  $\text{unitary-hmaD}[OF \text{unit-U}]$  **by simp**  
**also have**  $\dots \neq 0$   
**by simp**  
**finally have**  $U *v u \neq 0$  **by simp**  
**hence**  $u\text{-nz}: u \neq 0$   
**by**  $(\text{cases } u = 0)$  **auto**

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

**have**  $A\text{-sim}: \text{similar-matrix-wit } A (\text{diag } Ad) U (\text{adjoint-hma } U)$  **and**  $Ad\text{-real}: \forall i. Ad \$h i \in \mathbb{R}$   
**using**  $A\text{-decomp}$  **unfolding**  $\text{real-diag-decomp-hma-def}$   $\text{unitary-diag-def}$   $\text{unitarily-equiv-hma-def}$   
**by auto**

**have**  $\text{diag-mat-hma}(\text{diag } Ad) = \text{eigenvalues}(\text{diag } Ad)$   
**by**  $(\text{intro upper-tri-eivals}[\text{symmetric}] \text{diag-imp-upper-tri } A\text{-sim})$  **auto**  
**also have**  $\dots = \text{eigenvalues } A$

**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)  $\in$  # 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  $\in$  # mset-set UNIV#} - {# Ad \$h i #}  
**unfolding** 5 diag-mat-hma-def diag-def **by** simp

**also have** ... = {#Ad \$h j. j  $\in$  # mset-set UNIV - mset-set {i}#}  
**by** (subst image-mset-Diff) auto

**also have** ... = {#Ad \$h j. j  $\in$  # 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  $\in$  # mset-set (UNIV - {i})#} **by** simp

**have** cmod (Ad \$h k)  $\leq \gamma_a$  TYPE ('n) **if**  $n > 1$   $k \neq i$  **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)  $\leq \gamma_a$  TYPE ('n)  $\vee k = i$  **for** k

**using** n-gt-0 **by** (cases  $n = 1$ , auto)

**have** Re (Ad \$h k)  $\leq \gamma_2$  TYPE ('n) **if**  $n > 1$   $k \neq i$  **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)  $\leq \gamma_2$  TYPE ('n)  $\vee k = i$  **for** k

**using** n-gt-0 **by** (cases  $n = 1$ , auto)

**show**  $\Lambda_e$ -ge-0:  $\gamma_a$  TYPE ('n)  $\geq 0$

**proof** (cases  $n > 1$ )

**case** True

**then obtain** k **where** k-def:  $k \neq i$

**by** (metis (full-types) card-n from-nat-inj n-gt-0 one-neq-zero)

**have**  $0 \leq$  cmod (Ad \$h k)

**by** simp

**also have** ...  $\leq \gamma_a$  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  $\exists v. \text{cinner } v \ 1 = 0 \wedge v \neq 0 \wedge A *v \ v = \beta *s \ v$  if  $\beta$ -ran:  $\beta \in \#$  eigenvalues  $A - \{\#1\#$   
 for  $\beta$

proof –

obtain  $j$  where  $j$ -def:  $\beta = \text{Ad } \$h \ j \ j \neq i$   
 using  $\beta$ -ran unfolding 4 by *auto*  
 define  $v$  where  $v = U *v \ \text{axis } j \ 1$

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

have  $\text{cinner } v \ 1 = \text{cinner } (\text{axis } j \ 1) (\text{adjoint-hma } U *v \ 1)$   
 unfolding  $v$ -def *adjoint-def-alter-hma* by *simp*  
 also have  $\dots = \text{cinner } (\text{axis } j \ 1) (\text{axis } i \ \alpha)$   
 unfolding  $u$ -def[symmetric] *u-alt* by *simp*  
 also have  $\dots = 0$   
 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:  $\text{cinner } v \ 1 = 0$   
 by *simp*

have  $\text{cinner } v \ v = \text{cinner } (\text{axis } j \ 1) (\text{adjoint-hma } U *v \ (U *v \ (\text{axis } j \ 1)))$   
 unfolding  $v$ -def *adjoint-def-alter-hma* by *simp*  
 also have  $\dots = \text{cinner } (\text{axis } j \ 1) (\text{axis } j \ 1)$   
 unfolding *matrix-vector-mul-assoc unitary-hmaD[OF unit-U]* by *simp*  
 also have  $\dots = 1$   
 unfolding *cinner-def axis-def scalar-product-def map-vector-def*  
 by (*auto simp:if-distrib if-distribR sum.If-cases*)  
 finally have  $\text{cinner } v \ v = 1$   
 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  $A - \{\#1\#$ .  $\exists v. \text{cinner } v \ 1 = 0 \wedge v \neq 0 \wedge A *v \ v = \alpha *s \ v$   
 by *simp*

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

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

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

**qed**

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

**have**  $v \cdot (A * v v) \leq \gamma_2 \text{ TYPE}('n) * \text{norm } v^{\wedge 2}$  **if**  $v \cdot 1 = 0$  **for**  $v :: \text{real}^{\wedge n}$   
**proof** –

**define**  $v'$  **where**  $v' = \text{map-vector complex-of-real } v$   
**define**  $w$  **where**  $w = \text{adjoint-hma } U * v v'$

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

**also have**  $\dots = \text{cinner } v' ((1 / \alpha) * s 1)$   
**unfolding** *u-def matrix-vector-mul-assoc unitary-hmaD[OF unit-U]* **by** *simp*  
**also have**  $\dots = \text{cnj } (1 / \alpha) * \text{cinner } v' 1$   
**unfolding** *cinner-scale-right* **by** *simp*  
**also have**  $\dots = \text{cnj } (1 / \alpha) * \text{complex-of-real } (v \cdot 1)$   
**unfolding** *cinner-def scalar-product-def map-vector-def inner-vec-def v'-def*  
**by** (*intro arg-cong2[where f=(\*) refl*) (*simp*)  
**also have**  $\dots = 0$   
**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**  $\dots = \text{cinner } v' v'$   
**unfolding** *v'-def cinner-def scalar-product-def inner-vec-def map-vector-def* **by** *simp*  
**also have**  $\dots = \text{norm } v'^2$   
**unfolding** *cinner-self* **by** *simp*  
**also have**  $\dots = \text{norm } w^2$   
**unfolding** *w-def unitary-iso[OF unitary-hma-adjoint[OF unit-U]]* **by** *simp*  
**also have**  $\dots = \text{cinner } w w$   
**unfolding** *cinner-self* **by** *simp*  
**also have**  $\dots = (\sum j \in \text{UNIV}. \text{complex-of-real } (\text{cmod } (w \$h j)^2))$   
**unfolding** *cinner-def scalar-product-def map-vector-def*  
*cmod-power2 complex-mult-cnj[symmetric]* **by** *simp*  
**also have**  $\dots = \text{complex-of-real } (\sum j \in \text{UNIV}. (\text{cmod } (w \$h j)^2))$   
**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**  $\dots = \text{cinner } v' (\text{map-matrix of-real } A * v v')$   
**unfolding** *v'-def of-real-hom.mult-mat-vec-hma* **by** *simp*  
**also have**  $\dots = \text{cinner } v' (A * v v')$   
**unfolding** *map-matrix-def map-vector-def A-def* **by** *auto*  
**also have**  $\dots = \text{cinner } v' (U ** \text{diag } Ad ** \text{adjoint-hma } U * v v')$   
**using** *A-sim* **unfolding** *similar-matrix-wit-def* **by** *simp*  
**also have**  $\dots = \text{cinner } (\text{adjoint-hma } U * v v') (\text{diag } Ad ** \text{adjoint-hma } U * v v')$   
**unfolding** *adjoint-def-alter-hma adjoint-adjoint adjoint-adjoint-id*  
**by** (*simp add:matrix-vector-mul-assoc matrix-mul-assoc*)  
**also have**  $\dots = \text{cinner } w (\text{diag } Ad * v w)$   
**unfolding** *w-def* **by** (*simp add:matrix-vector-mul-assoc*)  
**also have**  $\dots = \text{cinner } w (Ad * w)$   
**unfolding** *diag-vec-mult-eq* **by** *simp*  
**also have**  $\dots = (\sum j \in \text{UNIV}. \text{cnj } (Ad \$h j) * \text{cmod } (w \$h j)^2)$   
**unfolding** *cinner-def map-vector-def scalar-product-def cmod-power2 complex-mult-cnj[symmetric]*  
**by** (*simp add:algebra-simps*)  
**also have**  $\dots = (\sum j \in \text{UNIV}. Ad \$h j * \text{cmod } (w \$h j)^2)$   
**using** *Ad-real* **by** (*intro sum.cong refl arg-cong2[where f=(\*) iffD1[OF Reals-cnj-iff]]*) *auto*  
**also have**  $\dots = (\sum j \in \text{UNIV}. \text{complex-of-real } (\text{Re } (Ad \$h j) * \text{cmod } (w \$h j)^2))$   
**using** *Ad-real* **by** (*intro sum.cong refl*) *simp*  
**also have**  $\dots = \text{complex-of-real } (\sum j \in \text{UNIV}. \text{Re } (Ad \$h j) * \text{cmod } (w \$h j)^2)$   
**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  $v \cdot (A * v) = (\sum_{j \in UNIV}. \text{Re } (A \$ j) * \text{cmod } (w \$ j) \hat{2})$   
 using *of-real-hom.injectivity* by *blast*  
 also have  $\dots \leq (\sum_{j \in UNIV}. \gamma_2 \text{ TYPE } ('n) * \text{cmod } (w \$ j) \hat{2})$   
 using *w-orth Re-Ad* by (*intro sum-mono mult-right-mono'*) *auto*  
 also have  $\dots = \gamma_2 \text{ TYPE } ('n) * (\sum_{j \in UNIV}. \text{cmod } (w \$ j) \hat{2})$   
 by (*simp add:sum-distrib-left*)  
 also have  $\dots = \gamma_2 \text{ TYPE } ('n) * \text{norm } v \hat{2}$   
 unfolding *norm-v* by *simp*  
 finally show *?thesis* by *simp*  
 qed

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

lemma *find-any-real-ev*:  
 assumes *complex-of-real*  $\alpha \in \# \text{ eigenvalues } A - \{\#1\}$   
 shows  $\exists v. v \cdot 1 = 0 \wedge v \neq 0 \wedge A * v = \alpha * v$   
 proof –  
 obtain *w* where *w-def*:  $\text{cinner } w = 0 \wedge w \neq 0 \wedge A * w = \alpha * w$   
 using *find-any-ev assms* by *auto*  
  
 have  $w = 0$  if *map-vector Re*  $(1 * w) = 0$  *map-vector Re*  $(i * w) = 0$   
 using *that* by (*simp add:vec-eq-iff map-vector-def complex-eq-iff*)  
 then obtain *c* where *c-def*: *map-vector Re*  $(c * w) \neq 0$   
 using *w-def(2)* by *blast*  
  
 define *u* where  $u = c * w$   
  
 define *v* where  $v = \text{map-vector Re } u$   
  
 hence  $v \cdot 1 = \text{Re } (\text{cinner } u)$   
 unfolding *cinner-def inner-vec-def scalar-product-def map-vector-def* by *simp*  
 also have  $\dots = 0$   
 unfolding *u-def cinner-scale-left w-def(1)* by *simp*  
 finally have  $1 : v \cdot 1 = 0$  by *simp*  
  
 have  $A * v = (\chi \ i. \sum_{j \in UNIV}. A \$ i \$ j * \text{Re } (u \$ j))$   
 unfolding *matrix-vector-mult-def v-def map-vector-def* by *simp*  
 also have  $\dots = (\chi \ i. \sum_{j \in UNIV}. \text{Re } ( \text{of-real } (A \$ i \$ j) * u \$ j))$   
 by *simp*  
 also have  $\dots = (\chi \ i. \text{Re } (\sum_{j \in UNIV}. A \$ i \$ j * u \$ j))$   
 unfolding *A-def* by *simp*  
 also have  $\dots = \text{map-vector Re } (A * u)$   
 unfolding *map-vector-def matrix-vector-mult-def* by *simp*  
 also have  $\dots = \text{map-vector Re } (\text{of-real } \alpha * u)$   
 unfolding *u-def vector-scalar-commute w-def(3)*  
 by (*simp add:ac-simps*)  
 also have  $\dots = \alpha * v$   
 unfolding *v-def* by (*simp add:vec-eq-iff map-vector-def*)  
 finally have  $2 : A * v = \alpha * v$  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  $\gamma_a TYPE('n) \in \# image-mset cmod (eigenvalues A - \{\#1::complex\})$

proof –

have  $set-mset (eigenvalues A - \{\#(1::complex)\}) \neq \{\}$

using *assms size-evs* by auto

hence  $2: cmod ' set-mset (eigenvalues A - \{\#1\}) \neq \{\}$

by *simp*

have  $\gamma_a TYPE('n) \in set-mset (image-mset cmod (eigenvalues A - \{\#1\}))$

unfolding  *$\gamma_a$ -def* using *assms 2 Max-in* by auto

thus  $\gamma_a TYPE('n) \in \# image-mset cmod (eigenvalues A - \{\#1\})$

by *simp*

qed

lemma  *$\gamma_2$ -real-ev*:

assumes  $n > 1$

shows  $\exists v. (\exists \alpha. abs \alpha = \gamma_a TYPE('n) \wedge v \cdot 1 = 0 \wedge v \neq 0 \wedge A * v = \alpha * v)$

proof –

obtain  $\alpha$  where  *$\alpha$ -def*:  $cmod \alpha = \gamma_a TYPE('n) \alpha \in \# eigenvalues A - \{\#1\}$

using *find- $\gamma_2$ [OF assms]* by auto

have  $\alpha \in \mathbb{R}$

using *in-diffD[OF  $\alpha$ -def(2)] evs-real* by auto

then obtain  $\beta$  where  *$\beta$ -def*:  $\alpha = of-real \beta$

using *Reals-cases* by auto

have  $0: complex-of-real \beta \in \# eigenvalues A - \{\#1\}$

using  *$\alpha$ -def unfolding  $\beta$ -def* by auto

have  $1: |\beta| = \gamma_a TYPE('n)$

using  *$\alpha$ -def unfolding  $\beta$ -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  $v \cdot 1 = 0$

shows  $norm (A * v) \leq \gamma_a TYPE('n) * norm v$

proof –

define  $w$  where  $w = map-vector complex-of-real v$

have *cinner*  $w 1 = v \cdot 1$

unfolding *w-def cinner-def map-vector-def scalar-product-def inner-vec-def*

by *simp*

also have  $\dots = 0$  using *assms* by *simp*

finally have  $0: cinner w 1 = 0$  by *simp*

have  $norm (A * v) = norm (map-matrix complex-of-real A * v (map-vector complex-of-real v))$

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

**lemma**  $\Lambda_e\text{-eq-}\Lambda$ :  $\Lambda_a = \gamma_a \ \text{TYPE}('n)$

**proof** –

**have**  $|g\text{-inner } f \ (g\text{-step } f)| \leq \gamma_a \ \text{TYPE}('n) * (g\text{-norm } f)^2$   
**(is ?L ≤ ?R) if**  $g\text{-inner } f \ (\lambda\cdot. 1) = 0$  **for**  $f$

**proof** –

**define**  $v$  **where**  $v = (\chi \ i. f \ (\text{enum-verts } i))$

**have**  $0$ :  $v \cdot 1 = 0$

**using** *that* **unfolding** *g-inner-conv one-vec-def v-def* **by** *auto*

**have**  $?L = |v \cdot (A * v \ v)|$

**unfolding** *g-inner-conv g-step-conv v-def* **by** *simp*

**also have**  $\dots \leq (\text{norm } v * \text{norm } (A * v \ v))$

**by** *(intro Cauchy-Schwarz-ineq2)*

**also have**  $\dots \leq (\text{norm } v * (\gamma_a \ \text{TYPE}('n) * \text{norm } v))$

**by** *(intro mult-left-mono  $\gamma_a\text{-real-bound } 0$ ) auto*

**also have**  $\dots = ?R$

**unfolding** *g-norm-conv v-def* **by** *(simp add: algebra-simps power2-eq-square)*

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

**qed**

**hence**  $\Lambda_a \leq \gamma_a \ \text{TYPE}('n)$

**using**  $\gamma_a\text{-ge-}0$  **by** *(intro expander-intro-1) auto*

**moreover have**  $\Lambda_a \geq \gamma_a \ \text{TYPE}('n)$

**proof** *(cases  $n > 1$ )*

**case** *True*

**then obtain**  $v \ \alpha$  **where**  $v\text{-def: } \text{abs } \alpha = \gamma_a \ \text{TYPE}('n) \ A * v \ v = \alpha * s \ v \ v \neq 0 \ v \cdot 1 = 0$

**using**  $\gamma_2\text{-real-ev}$  **by** *auto*

**define**  $f$  **where**  $f \ x = v \ \$h \ \text{enum-verts-inv } x$  **for**  $x$

**have**  $v\text{-alt}$ :  $v = (\chi \ i. f \ (\text{enum-verts } i))$

**unfolding** *f-def Rep-inverse* **by** *simp*

**have**  $g\text{-inner } f \ (\lambda\cdot. 1) = v \cdot 1$

**unfolding** *g-inner-conv v-alt one-vec-def* **by** *simp*

**also have**  $\dots = 0$  **using** *v-def* **by** *simp*

**finally have**  $2:g\text{-inner } f \ (\lambda\cdot. 1) = 0$  **by** *simp*

**have**  $\gamma_a \ \text{TYPE}('n) * g\text{-norm } f^{\wedge 2} = \gamma_a \ \text{TYPE}('n) * \text{norm } v^{\wedge 2}$

**unfolding** *g-norm-conv v-alt* **by** *simp*

**also have**  $\dots = \gamma_a \ \text{TYPE}('n) * |v \cdot v|$

**by** *(simp add: power2-norm-eq-inner)*

**also have**  $\dots = |v \cdot (\alpha * s \ v)|$

**unfolding** *v-def(1)[symmetric] scalar-mult-eq-scaleR*

**by** *(simp add: abs-mult)*

**also have**  $\dots = |v \cdot (A * v \ v)|$

**unfolding** *v-def* **by** *simp*

**also have**  $\dots = |g\text{-inner } f \ (g\text{-step } f)|$

**unfolding** *g-inner-conv g-step-conv v-alt* **by** *simp*

**also have**  $\dots \leq \Lambda_a * g\text{-norm } f^{\wedge 2}$

**by** *(intro expansionD1 2)*

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

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

**lemma**  $\Lambda_2\text{-eq-}\gamma_2$ :  $\Lambda_2 = \gamma_2 \text{ TYPE}('n)$   
**proof**  $(\text{cases } n > 1)$   
**case**  $True$   
  
**obtain**  $v$  **where**  $v\text{-def}: v \cdot 1 = 0 \wedge v \neq 0 \wedge A * v v = \gamma_2 \text{ TYPE}('n) * s v$   
**using**  $\gamma_2\text{-ev}[OF True]$  **by auto**  
  
**define**  $f$  **where**  $f x = v \$h \text{ enum-verts-inv } x$  **for**  $x$   
**have**  $v\text{-alt}: v = (\chi i. f(\text{enum-verts } i))$   
**unfolding**  $f\text{-def Rep-inverse}$  **by simp**  
  
**have**  $g\text{-inner } f(\lambda-. 1) = v \cdot 1$   
**unfolding**  $g\text{-inner-conv } v\text{-alt one-vec-def}$  **by simp**  
**also have**  $\dots = 0$  **unfolding**  $v\text{-def}(1)$  **by simp**  
**finally have**  $f\text{-orth}: g\text{-inner } f(\lambda-. 1) = 0$  **by simp**

**have**  $\gamma_2 \text{ TYPE}('n) * \text{norm } v^{\wedge}2 = v \cdot (\gamma_2 \text{ TYPE}('n) * s v)$   
**unfolding** *power2-norm-eq-inner* **by** (*simp add: algebra-simps scalar-mult-eq-scaleR*)  
**also have**  $\dots = v \cdot (A * v v)$   
**unfolding** *v-def* **by** *simp*  
**also have**  $\dots = g\text{-inner } f \text{ (} g\text{-step } f \text{)}$   
**unfolding** *v-alt g-inner-conv g-step-conv* **by** *simp*  
**also have**  $\dots \leq \Lambda_2 * g\text{-norm } f^{\wedge}2$   
**by** (*intro os-expanderD f-orth*)  
**also have**  $\dots = \Lambda_2 * \text{norm } v^{\wedge}2$   
**unfolding** *v-alt g-norm-conv* **by** *simp*  
**finally have**  $\gamma_2 \text{ TYPE}('n) * \text{norm } v^{\wedge}2 \leq \Lambda_2 * \text{norm } v^{\wedge}2$  **by** *simp*  
**hence**  $\gamma_2 \text{ TYPE}('n) \leq \Lambda_2$   
**using** *v-def(2)* **by** *simp*  
**moreover have**  $\Lambda_2 \leq \gamma_2 \text{ TYPE}('n)$   
**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**  $g\text{-inner } f \text{ (}\lambda\text{-. } 1) = 0$   
**shows**  $g\text{-norm } (g\text{-step } f) \leq \Lambda_a * g\text{-norm } f$  (**is**  $?L \leq ?R$ )  
**proof** –  
**define** *v* **where**  $v = (\chi \text{ i. } f \text{ (enum-verts i)})$   
**have**  $v \cdot 1 = g\text{-inner } f \text{ (}\lambda\text{-. } 1)$   
**unfolding** *g-inner-conv v-def one-vec-def* **by** *simp*  
**also have**  $\dots = 0$  **using** *assms* **by** *simp*  
**finally have**  $0 : v \cdot 1 = 0$  **by** *simp*  
**have**  $g\text{-norm } (g\text{-step } f) = \text{norm } (A * v v)$   
**unfolding** *g-norm-conv g-step-conv v-def* **by** *auto*  
**also have**  $\dots \leq \Lambda_a * \text{norm } v$   
**unfolding**  *$\Lambda_e$ -eq- $\Lambda$*  **by** (*intro  $\gamma_a$ -real-bound 0*)  
**also have**  $\dots = \Lambda_a * g\text{-norm } f$   
**unfolding** *g-norm-conv v-def* **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *rayleigh-bound*:  
**fixes**  $v :: \text{real}^n$   
**shows**  $|v \cdot (A * v v)| \leq \text{norm } v^{\wedge}2$   
**proof** –  
**define** *f* **where**  $f x = v \$h \text{ enum-verts-inv } x$  **for**  $x$   
**have** *v-alt*:  $v = (\chi \text{ i. } f \text{ (enum-verts i)})$   
**unfolding** *f-def Rep-inverse* **by** *simp*  
  
**have**  $|v \cdot (A * v v)| = |g\text{-inner } f \text{ (} g\text{-step } f \text{)}|$   
**unfolding** *v-alt g-inner-conv g-step-conv* **by** *simp*  
**also have**  $\dots = |(\sum_{a \in \text{arcs } G} f \text{ (head } G \ a) * f \text{ (tail } G \ a))| / d$   
**unfolding** *g-inner-step-eq* **by** *simp*  
**also have**  $\dots \leq (d * (g\text{-norm } f)^2) / d$   
**by** (*intro divide-right-mono bdd-above-aux*) *auto*  
**also have**  $\dots = g\text{-norm } f^{\wedge}2$   
**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#})  $\neq$  {}
    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 ...  $\in$  Re ' set-mset (eigenvalues A - {#1::complex#})
    using Max-in 0 by simp
  finally have  $\gamma_2$  TYPE ('n)  $\in$  Re ' set-mset (eigenvalues A - {#1::complex#})
    by simp
  then obtain  $\alpha$  where  $\alpha$ -def:  $\alpha \in$  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 ...  $\leq$  cmod  $\alpha$ 
    using abs-Re-le-cmod by simp
  also have ...  $\leq$  Max (cmod ' set-mset (eigenvalues A - {#1#}))
    using  $\alpha$ -def(1) by (intro Max-ge) auto
  also have ...  $\leq \gamma_a$  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**  $m = \text{card } (S \cap \text{verts } G) \ 2 * m \leq n$

**shows**  $\Lambda_e * m \leq \text{real } (\text{card } (\text{edges-betw } S \ (-S)))$

**proof** –

**define**  $S'$  **where**  $S' = S \cap \text{verts } G$

**have**  $\Lambda_e * m = \Lambda_e * \text{card } S'$

**using** *assms(1)*  $S'$ -**def** **by** *simp*

**also have**  $\dots \leq \text{real } (\text{card } (\text{edges-betw } S' \ (-S')))$

**using** *assms* **unfolding**  $S'$ -**def** **by** (*intro edge-expansionD*) *auto*

**also have**  $\dots = \text{real } (\text{card } (\text{edges-betw } S \ (-S)))$

**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*:

$\text{card } (\text{edges-betw } S \ T) = \text{card } (\text{edges-betw } T \ S)$  (**is**  $?L = ?R$ )

**proof** –

**have**  $?L = (\sum a \in \text{arcs } G. \text{ of-bool } (\text{tail } G \ a \in S \wedge \text{head } G \ a \in T))$

**unfolding** *edges-betw-def of-bool-def* **by** (*simp add:sum.If-cases Int-def*)

**also have**  $\dots = (\sum e \in \# \text{ edges } G. \text{ of-bool } (\text{fst } e \in S \wedge \text{snd } e \in T))$

**unfolding** *sum-unfold-sum-mset edges-def arc-to-ends-def*

**by** (*simp add:image-mset.compositionality comp-def*)

**also have**  $\dots = (\sum e \in \# \text{ edges } G. \text{ of-bool } (\text{snd } e \in S \wedge \text{fst } e \in T))$

**by** (*subst edges-sym[OF sym, symmetric]*)

(*simp add:image-mset.compositionality comp-def case-prod-beta*)

**also have**  $\dots = (\sum a \in \text{arcs } G. \text{ of-bool } (\text{tail } G \ a \in T \wedge \text{head } G \ a \in S))$

**unfolding** *sum-unfold-sum-mset edges-def arc-to-ends-def*

**by** (*simp add:image-mset.compositionality comp-def conj commute*)

**also have**  $\dots = ?R$

**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**  $S \subseteq \text{verts } G$

**shows**  $\text{card } (\text{edges-betw } S \ \text{UNIV}) = \text{card } S * d$  (**is**  $?L = ?R$ )

**proof** –

**have**  $?L = \text{card } (\bigcup (\text{out-arcs } G \ ` \ S))$

**unfolding** *edges-betw-def out-arcs-def* **by** (*intro arg-cong[where f=card]*) *auto*

**also have**  $\dots = (\sum i \in S. \text{card } (\text{out-arcs } G \ i))$

**using** *finite-subset[OF assms]* **unfolding** *out-arcs-def*

**by** (*intro card-UN-disjoint*) *auto*

**also have** ... =  $(\sum_{i \in S} \text{out-degree } G \ i)$   
**unfolding** *out-degree-def* **by** *simp*  
**also have** ... =  $(\sum_{i \in S} d)$   
**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**  $\Lambda_e \geq d * (1 - \Lambda_2) / 2$

**proof** –

**have**  $\text{real} (\text{card} (\text{edges-betw } S \ (-S))) \geq (d * (1 - \Lambda_2) / 2) * \text{real} (\text{card } S)$

**if**  $S \subseteq \text{verts } G$   $2 * \text{card } S \leq n$  **for**  $S$

**proof** –

**let**  $?ct = \text{real} (\text{card} (\text{verts } G - S))$

**let**  $?cs = \text{real} (\text{card } S)$

**have**  $\text{card} (\text{edges-betw } S \ S) + \text{card} (\text{edges-betw } S \ (-S)) = \text{card} (\text{edges-betw } S \ S \cup \text{edges-betw } S \ (-S))$

**unfolding** *edges-betw-def* **by** (*intro card-Un-disjoint[symmetric]*) *auto*

**also have** ... =  $\text{card} (\text{edges-betw } S \ \text{UNIV})$

**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**  $\text{card} (\text{edges-betw } S \ S) + \text{card} (\text{edges-betw } S \ (-S)) = d * ?cs$  **by** *simp*

**hence** 4:  $\text{card} (\text{edges-betw } S \ S) = d * ?cs - \text{card} (\text{edges-betw } S \ (-S))$

**by** *simp*

**have**  $\text{card} (\text{edges-betw } S \ (-S)) + \text{card} (\text{edges-betw } (-S) \ (-S)) = \text{card} (\text{edges-betw } S \ (-S) \cup \text{edges-betw } (-S) \ (-S))$

**unfolding** *edges-betw-def* **by** (*intro card-Un-disjoint[symmetric]*) *auto*

**also have** ... =  $\text{card} (\text{edges-betw } \text{UNIV} \ (\text{verts } G - S))$

**unfolding** *edges-betw-def* **by** (*intro arg-cong[where f=card]*) *auto*

**also have** ... =  $\text{card} (\text{edges-betw } (\text{verts } G - S) \ \text{UNIV})$

**by** (*intro edges-betw-sym*)

**also have** ... =  $d * ?ct$

**using** *edges-betw-reg* **by** *auto*

**finally have**  $\text{card} (\text{edges-betw } S \ (-S)) + \text{card} (\text{edges-betw } (-S) \ (-S)) = d * ?ct$  **by** *simp*

**hence** 5:  $\text{card} (\text{edges-betw } (-S) \ (-S)) = d * ?ct - \text{card} (\text{edges-betw } S \ (-S))$

**by** *simp*

**have** 6:  $\text{card} (\text{edges-betw } (-S) \ S) = \text{card} (\text{edges-betw } S \ (-S))$

**by** (*intro edges-betw-sym*)

**have**  $?cs + ?ct = \text{real} (\text{card} (S \cup (\text{verts } G - S)))$

**unfolding** *of-nat-add[symmetric]* **using** *finite-subset[OF that(1)]*

**by** (*intro-cong* [ $\sigma_1$  *of-nat*,  $\sigma_1$  *card*] *more:card-Un-disjoint[symmetric]*) *auto*

**also have** ... =  $\text{real } n$

**unfolding** *n-def* **using** *that(1)* **by** (*intro-cong* [ $\sigma_1$  *of-nat*,  $\sigma_1$  *card*]) *auto*

**finally have** 7:  $?cs + ?ct = n$  **by** *simp*

**define**  $f$  **where**

$f \ x = \text{real} (\text{card} (\text{verts } G - S)) * \text{of-bool} (x \in S) - \text{card } S * \text{of-bool} (x \notin S)$  **for**  $x$

**have**  $g\text{-inner } f \ (\lambda-. 1) = ?cs * ?ct - \text{real} (\text{card} (\text{verts } G \cap \{x. x \notin S\})) * ?cs$

**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* [ $\sigma_2$   $(-)$ ,  $\sigma_2$   $(*)$ ,  $\sigma_1$  *of-nat*,  $\sigma_1$  *card*]) *auto*

also have ... = 0 by simp  
 finally have 11:  $g\text{-inner } f (\lambda. 1) = 0$  by simp

have  $g\text{-norm } f^{\wedge} 2 = (\sum v \in \text{verts } G. f v^{\wedge} 2)$

unfolding  $g\text{-norm-sq } g\text{-inner-def } \text{conjugate-real-def}$  by (simp add: power2-eq-square)

also have ... =  $(\sum v \in \text{verts } G. ?ct^{\wedge} 2 * (\text{of-bool } (v \in S))^2) + (\sum v \in \text{verts } G. ?cs^{\wedge} 2 * (\text{of-bool } (v \notin S))^2)$

unfolding  $f\text{-def } \text{power2-diff}$  by (simp add: sum.distrib sum-subtractf power-mult-distrib)

also have ... =  $\text{real } (\text{card } (\text{verts } G \cap S)) * ?ct^{\wedge} 2 + \text{real } (\text{card } (\text{verts } G \cap \{v. v \notin S\})) * ?cs^{\wedge} 2$

unfolding  $\text{of-bool-def}$  by (simp add: if-distrib if-distribR sum.If-cases)

also have ... =  $\text{real } (\text{card } S) * (\text{real } (\text{card } (\text{verts } G - S)))^2 + \text{real } (\text{card } (\text{verts } G - S)) * (\text{real } (\text{card } S))^2$

using that(1) by (intro-cong [ $\sigma_2(+)$ ,  $\sigma_2(*)$ ,  $\sigma_2$  power,  $\sigma_1$  of-nat,  $\sigma_1$  card]) auto

also have ... =  $\text{real } (\text{card } S) * \text{real } (\text{card } (\text{verts } G - S)) * (?cs + ?ct)$

by (simp add: power2-eq-square algebra-simps)

also have ... =  $\text{real } (\text{card } S) * \text{real } (\text{card } (\text{verts } G - S)) * n$

unfolding 7 by simp

finally have 9:  $g\text{-norm } f^{\wedge} 2 = \text{real } (\text{card } S) * \text{real } (\text{card } (\text{verts } G - S)) * \text{real } n$  by simp

have  $(\sum a \in \text{arcs } G. f (\text{head } G a) * f (\text{tail } G a)) =$

$(\text{card } (\text{edges-betw } S S) * ?ct * ?ct) + (\text{card } (\text{edges-betw } (-S) (-S)) * ?cs * ?cs) -$

$(\text{card } (\text{edges-betw } S (-S)) * ?ct * ?cs) - (\text{card } (\text{edges-betw } (-S) S) * ?cs * ?ct)$

unfolding  $f\text{-def}$  by (simp add: of-bool-def algebra-simps Int-def if-distrib if-distribR edges-betw-def sum.If-cases)

also have ... =  $d * ?cs * ?ct * (?cs + ?ct) - \text{card } (\text{edges-betw } S (-S)) * (?ct * ?ct + 2 * ?ct * ?cs + ?cs * ?cs)$

unfolding 4 5 6 by (simp add: algebra-simps)

also have ... =  $d * ?cs * ?ct * n - (?ct + ?cs)^{\wedge} 2 * \text{card } (\text{edges-betw } S (-S))$

unfolding  $\text{power2-diff } 7$   $\text{power2-sum}$  by (simp add: ac-simps power2-eq-square)

also have ... =  $d * ?cs * ?ct * n - n^{\wedge} 2 * \text{card } (\text{edges-betw } S (-S))$

using 7 by (simp add: algebra-simps)

finally have 8:  $(\sum a \in \text{arcs } G. f (\text{head } G a) * f (\text{tail } G a)) = d * ?cs * ?ct * n - n^{\wedge} 2 * \text{card } (\text{edges-betw } S (-S))$

by simp

have  $d * ?cs * ?ct * n - n^{\wedge} 2 * \text{card } (\text{edges-betw } S (-S)) = (\sum a \in \text{arcs } G. f (\text{head } G a) * f (\text{tail } G a))$

unfolding 8 by simp

also have ...  $\leq d * (g\text{-inner } f (g\text{-step } f))$

unfolding  $g\text{-inner-step-eq}$  using  $d\text{-gt-0}$

by simp

also have ...  $\leq d * (\Lambda_2 * g\text{-norm } f^{\wedge} 2)$

by (intro mult-left-mono os-expanderD 11) auto

also have ... =  $d * \Lambda_2 * ?cs * ?ct * n$

unfolding 9 by simp

finally have  $d * ?cs * ?ct * n - n^{\wedge} 2 * \text{card } (\text{edges-betw } S (-S)) \leq d * \Lambda_2 * ?cs * ?ct * n$

by simp

hence  $n * n * \text{card } (\text{edges-betw } S (-S)) \geq n * (d * ?cs * ?ct * (1 - \Lambda_2))$

by (simp add: power2-eq-square algebra-simps)

hence 10:  $n * \text{card } (\text{edges-betw } S (-S)) \geq d * ?cs * ?ct * (1 - \Lambda_2)$

using  $n\text{-gt-0}$  by simp

have  $(d * (1 - \Lambda_2) / 2) * ?cs = (d * (1 - \Lambda_2) * (1 - 1 / 2)) * ?cs$

by simp

also have ...  $\leq d * (1 - \Lambda_2) * ((n - ?cs) / n) * ?cs$

using that  $n\text{-gt-0 } \Lambda_2\text{-le-1}$

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  $\bigwedge x. x \in B \implies g x \in A \wedge f (g x) = x$ 
  shows  $B \subseteq f \text{ ` } A$ 
  using assms by force

```

lemma *find-sorted-bij-1*:

```

  fixes  $g :: 'a \Rightarrow ('b :: \text{linorder})$ 
  assumes finite S
  shows  $\exists f. \text{bij-betw } f \{..\text{<card } S\} S \wedge \text{mono-on } \{..\text{<card } S\} (g \circ f)$ 

```

proof –

```

  define h where  $h x = \text{from-nat-into } S x$  for  $x$ 

```

```

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

```

```

  define xs where  $xs = \text{sort-key } (g \circ h) [0..\text{<card } S]$ 
  define f where  $f i = h (xs ! i)$  for  $i$ 

```

```

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

```

```

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

```

```

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

```

```

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

```

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

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

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

**moreover** **have**  $g x \leq g y$  **if**  $h x < h y$   $x \in S$   $y \in S$  **for**  $x y$   
**proof** –

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

**qed**  
**ultimately** **show**  $?thesis$  **by**  $\text{auto}$   
**qed**

**context**  $\text{regular-graph-tts}$   
**begin**

Normalized Laplacian of the graph

**definition**  $L$  **where**  $L = \text{mat } 1 - A$

**lemma**  $L\text{-pos-semidefinite}$ :

**fixes**  $v :: \text{real } ^n$   
**shows**  $v \cdot (L * v) \geq 0$

**proof** –

**have**  $0 = v \cdot v - \text{norm } v^2$  **unfolding**  $\text{power2-norm-eq-inner}$  **by**  $\text{simp}$   
**also** **have**  $\dots \leq v \cdot v - \text{abs } (v \cdot (A * v))$   
**by**  $(\text{intro diff-mono rayleigh-bound}) \text{ auto}$   
**also** **have**  $\dots \leq v \cdot v - v \cdot (A * v)$   
**by**  $(\text{intro diff-mono}) \text{ auto}$   
**also** **have**  $\dots = v \cdot (L * v)$   
**unfolding**  $L\text{-def}$  **by**  $(\text{simp add: algebra-simps})$   
**finally** **show**  $?thesis$  **by**  $\text{simp}$

**qed**

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

**lemma** *cheeger-aux-1*:

**assumes**  $n > 1$

**shows**  $\Lambda_e \leq d * \text{sqrt}(2 * (1 - \Lambda_2))$

**proof** –

**obtain**  $v$  **where**  $v\text{-def}$ :  $v \cdot 1 = 0 \ v \neq 0 \ A * v = \Lambda_2 * v$

**using**  $\Lambda_2\text{-eq-}\gamma_2 \ \gamma_2\text{-ev}$  [*OF assms*] **by** *auto*

**have**  $\text{False}$  **if**  $2 * \text{card} \{i. (1 * v) \$h i > 0\} > n \ 2 * \text{card} \{i. ((-1) * v) \$h i > 0\} > n$

**proof** –

**have**  $2 * n = n + n$  **by** *simp*

**also have**  $\dots < 2 * \text{card} \{i. (1 * v) \$h i > 0\} + 2 * \text{card} \{i. ((-1) * v) \$h i > 0\}$   
**by** (*intro add-strict-mono that*)

**also have**  $\dots = 2 * (\text{card} \{i. (1 * v) \$h i > 0\} + \text{card} \{i. ((-1) * v) \$h i > 0\})$   
**by** *simp*

**also have**  $\dots = 2 * (\text{card} (\{i. (1 * v) \$h i > 0\} \cup \{i. ((-1) * v) \$h i > 0\}))$

**by** (*intro arg-cong2* [**where**  $f = (*)$ ] *card-Un-disjoint* [*symmetric*]) *auto*

**also have**  $\dots \leq 2 * (\text{card} (\text{UNIV} :: 'n \text{ set}))$

**by** (*intro mult-left-mono card-mono*) *auto*

**finally have**  $2 * n < 2 * n$

**unfolding**  $n\text{-def}$  *card-n* **by** *auto*

**thus** *?thesis* **by** *simp*

**qed**

**then obtain**  $\beta :: \text{real}$  **where**  $\beta\text{-def}$ :  $\beta = 1 \vee \beta = (-1) \ 2 * \text{card} \{i. (\beta * v) \$h i > 0\} \leq n$

**unfolding** *not-le* [*symmetric*] **by** *blast*

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

**have**  $g\text{-orth}$ :  $g \cdot 1 = 0$  **unfolding**  $g\text{-def}$  **using**  $v\text{-def}(1)$

**by** (*simp add: scalar-mult-eq-scaleR*)

**have**  $g\text{-nz}$ :  $g \neq 0$

**unfolding**  $g\text{-def}$  **using**  $\beta\text{-def}(1)$   $v\text{-def}(2)$  **by** *auto*

**have**  $g\text{-ev}$ :  $A * g = \Lambda_2 * g$

**unfolding**  $g\text{-def}$  *scalar-mult-eq-scaleR* *matrix-vector-mult-scaleR*  $v\text{-def}(3)$  **by** *auto*

**have**  $g\text{-supp}$ :  $2 * \text{card} \{i. g \$h i > 0\} \leq n$

**unfolding**  $g\text{-def}$  **using**  $\beta\text{-def}(2)$  **by** *auto*

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

**have**  $(L * v \ f) \$h i \leq (1 - \Lambda_2) * g \$h i$  (**is**  $?L \leq ?R$ ) **if**  $g \$h i > 0$  **for**  $i$

**proof** –

**have**  $?L = f \$h i - (A * v \ f) \$h i$

**unfolding**  $L\text{-def}$  **by** (*simp add: algebra-simps*)

**also have**  $\dots = g \$h i - (\sum j \in \text{UNIV}. A \$h i \$h j * f \$h j)$

**unfolding** *matrix-vector-mult-def*  $f\text{-def}$  **using** *that* **by** *auto*

**also have**  $\dots \leq g \$h i - (\sum j \in \text{UNIV}. A \$h i \$h j * g \$h j)$

**unfolding**  $f\text{-def}$   $A\text{-def}$  **by** (*intro diff-mono sum-mono mult-left-mono*) *auto*

**also have**  $\dots = g \$h i - (A * v \ g) \$h i$

**unfolding** *matrix-vector-mult-def* **by** *simp*

**also have**  $\dots = (1 - \Lambda_2) * g \$h i$

**unfolding**  $g\text{-ev}$  **by** (*simp add: algebra-simps*)

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

**qed**

**moreover have**  $f \$h i \neq 0 \implies g \$h i > 0$  **for**  $i$

**unfolding**  $f\text{-def}$  **by** *simp*

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

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

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

**have**  $(x + y)^{\wedge 2} \leq 2 * (x^{\wedge 2} + y^{\wedge 2})$  **for**  $x y :: real$   
**proof** –

**have**  $(x + y)^{\wedge 2} = (x^{\wedge 2} + y^{\wedge 2}) + 2 * x * y$   
**unfolding** *power2-sum* **by** *simp*  
**also have**  $\dots \leq (x^{\wedge 2} + y^{\wedge 2}) + (x^{\wedge 2} + y^{\wedge 2})$   
**by** (*intro add-mono sum-squares-bound*) *auto*  
**finally show** *?thesis* **by** *simp*

**qed**

**hence**  $(\sum_{a \in arcs G}. (f' (tail G a) + f' (head G a))^2) \leq (\sum_{a \in arcs G}. 2 * (f' (tail G a)^{\wedge 2} + f' (head G a)^{\wedge 2}))$

**by** (*intro sum-mono*) *auto*  
**also have**  $\dots = 2 * ((\sum_{a \in arcs G}. f' (tail G a)^{\wedge 2}) + (\sum_{a \in arcs G}. f' (head G a)^{\wedge 2}))$   
**by** (*simp add:sum-distrib-left*)  
**also have**  $\dots = 4 * d * g\text{-norm } f^{\wedge 2}$   
**unfolding** *sum-arcs-tail* [**where**  $f = \lambda x. f' x^{\wedge 2}$ ] *sum-arcs-head* [**where**  $f = \lambda x. f' x^{\wedge 2}$ ]  
*g-norm-sq g-inner-def* **by** (*simp add:power2-eq-square*)  
**also have**  $\dots = 4 * d * \text{norm } f^{\wedge 2}$   
**unfolding** *g-norm-conv f'-alt* **by** *simp*  
**finally have**  $1$ :  $(\sum_{i \in arcs G}. (f' (tail G i) + f' (head G i))^2) \leq 4 * d * \text{norm } f^{\wedge 2}$   
**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**  $\dots = 2 * (d * (\sum_{v \in verts G}. (f' v)^2) - (\sum_{a \in arcs G}. f' (tail G a) * f' (head G a)))$   
**unfolding** *sum-arcs-tail* [**where**  $f = \lambda x. f' x^{\wedge 2}$ ] *sum-arcs-head* [**where**  $f = \lambda x. f' x^{\wedge 2}$ ] **by** *simp*  
**also have**  $\dots = 2 * (d * g\text{-inner } f' f' - d * g\text{-inner } f' (g\text{-step } f'))$   
**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**  $\dots = 2 * d * (g\text{-inner } f' f' - g\text{-inner } f' (g\text{-step } f'))$   
**by** (*simp add:algebra-simps*)  
**also have**  $\dots = 2 * d * (f \cdot f - f \cdot (L * v f))$   
**unfolding** *g-inner-conv g-step-conv f'-alt* **by** *simp*  
**also have**  $\dots = 2 * d * (f \cdot (L * v f))$   
**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** ...  $\leq$   $L2\text{-set } (\lambda a. f'(tail\ G\ a) + f'(head\ G\ a))\ (arcs\ G) *$   
 $L2\text{-set } (\lambda a. f'(tail\ G\ a) - f'(head\ G\ a))\ (arcs\ G)$   
**by**  $(intro\ L2\text{-set-mult-ineq})$   
**also have** ...  $\leq$   $\sqrt{4*d*norm\ f^2} * \sqrt{2 * d * (f \cdot (L * v\ f))}$   
**unfolding**  $L2\text{-set-def } 2$   
**by**  $(intro\ mult\text{-right-mono}\ iffD2[OF\ real\text{-sqrt-le-iff}]\ 1\ real\text{-sqrt-ge-zero}\ mult\text{-nonneg-nonneg}\ L\text{-pos-semidefinite})\ auto$   
**also have** ...  $= 2 * \sqrt{2} * d * norm\ f * \sqrt{f \cdot (L * v\ f)}$   
**by**  $(simp\ add:real\text{-sqrt-mult})$   
**finally have**  $hoory\text{-4-12}: B_f \leq 2 * \sqrt{2} * d * norm\ f * \sqrt{f \cdot (L * v\ f)}$   
**by**  $simp$

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

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

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

**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**  $(simp\ add:filter\text{-mset-ex-predicates})$

**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**  $bij\text{-betw-imp-inj-on}[OF\ \varrho\text{-bij}]\ edge\text{-set}$

**by**  $(intro\ arg\text{-cong2}[where\ f=(+)]\ filter\text{-mset-cong}\ refl\ inj\text{-on-eq-iff}[where\ A=verts\ G])\ auto$

**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**  $(intro\ arg\text{-cong2}[where\ f=(+)]\ filter\text{-mset-ex-predicates}[symmetric])\ auto$

**finally have**  $edges\text{-split}: edges\ G = \{\# e \in \# edges\ G. \varrho(fst\ e) < \varrho(snd\ e)\ \#\} +$   
 $\{\# e \in \# edges\ G. \varrho(fst\ e) > \varrho(snd\ e)\ \#\} + \{\# e \in \# edges\ G. fst\ e = snd\ e\ \#\}$

**by**  $simp$

**have**  $\varrho\text{-lt-n}: \varrho\ x < n$  **if**  $x \in verts\ G$  **for**  $x$

**using**  $bij\text{-betw-apply}[OF\ \varrho\text{-bij}]\ that$  **by**  $auto$

**have**  $\varphi\text{-}\varrho\text{-inv}: \varphi(\varrho\ x) = x$  **if**  $x \in verts\ G$  **for**  $x$

**unfolding**  $\varphi\text{-def}$  **using**  $bij\text{-betw-imp-inj-on}[OF\ \varrho\text{-bij}]$

**by**  $(intro\ the\text{-inv-into-f-f}\ that)\ auto$

**have**  $\varrho\text{-}\varphi\text{-inv}: \varrho(\varphi\ x) = x$  **if**  $x < n$  **for**  $x$

**unfolding**  $\varphi\text{-def}$  **using**  $bij\text{-betw-imp-inj-on}[OF\ \varrho\text{-bij}]\ bij\text{-betw-imp-surj-on}[OF\ \varrho\text{-bij}]\ that$

**by**  $(intro\ f\text{-the-inv-into-f})\ auto$

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

**have**  $\tau\text{-nonneg}: \tau\ k \geq 0$  **for**  $k$

**unfolding**  $\tau\text{-def}\ f'\text{-def}\ f\text{-def}$  **by**  $auto$

**have**  $\tau\text{-antimono}: \tau\ k \geq \tau\ l$  **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**  $\dots \leq f' (\varphi k)$   
**using**  $\varrho$ - $\varphi$ -inv *False* **that**  
**by** (*intro*  $\varrho$ -dec *bij-betw-apply*[*OF*  $\varphi$ -bij]) *auto*  
**also have**  $\dots = \tau k$   
**unfolding**  $\tau$ -def **using** *False* **that** **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

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

**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**  $x \in \varphi \text{ ' } \{..<m\}$  **for**  $x$   
**unfolding**  $\tau$ -def **using**  $m$ -le- $n$  **that** **by** *auto*

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

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

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

**have**  $\downarrow$ :  $\varrho v \leq x \iff v \in \varphi \text{ ' } \{..x\}$  **if**  $v \in \text{verts } G$   $x < n$  **for**  $v x$   
**proof** –

**have**  $\varrho v \leq x \iff \varrho v \in \{..x\}$   
**by** *simp*  
**also have**  $\dots \iff \varphi(\varrho v) \in \varphi \text{ ' } \{..x\}$   
**using** *bij-betw-imp-inj-on*[*OF*  $\varphi\text{-bij}$ ] *bij-betw-apply*[*OF*  $\varrho\text{-bij}$ ] that  
**by** (*intro inj-on-image-mem-iff*[**where**  $B = \{..<n\}$ , *symmetric*]) *auto*  
**also have**  $\dots \iff v \in \varphi \text{ ' } \{..x\}$   
**unfolding**  $\varphi\text{-}\varrho\text{-inv}$ [*OF* that(1)] **by** *simp*  
**finally show** *?thesis* **by** *simp*

**qed**

**have**  $B_f = (\sum a \in \text{arcs } G. |f'(tail\ G\ a)^{\wedge 2} - f'(head\ G\ a)^{\wedge 2}|)$   
**unfolding**  $B_f\text{-def}$  **by** *simp*

**also have**  $\dots = (\sum e \in \# \text{ edges } G. |f'(fst\ e)^{\wedge 2} - f'(snd\ e)^{\wedge 2}|)$   
**unfolding** *edges-def* *arc-to-ends-def* *sum-unfold-sum-mset*  
**by** (*simp add:image-mset.compositionality comp-def*)

**also have**  $\dots =$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(fst\ e) < \varrho(snd\ e)\#\}. |(f'(fst\ e))^2 - (f'(snd\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(fst\ e))^2 - (f'(snd\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. fst\ e = snd\ e\#\}. |(f'(fst\ e))^2 - (f'(snd\ e))^2|)$   
**by** (*subst edges-split*) *simp*

**also have**  $\dots =$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(fst\ e))^2 - (f'(snd\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(snd\ e))^2 - (f'(fst\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. fst\ e = snd\ e\#\}. |(f'(fst\ e))^2 - (f'(snd\ e))^2|)$   
**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**  $\dots =$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(snd\ e))^2 - (f'(fst\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(snd\ e))^2 - (f'(fst\ e))^2|) +$   
 $(\sum e \in \#\{ \#e \in \# \text{ edges } G. fst\ e = snd\ e\#\}. 0)$   
**by** (*intro-cong* [ $\sigma_2$  (+),  $\sigma_1$  *sum-mset*] *more:image-mset-cong*) *auto*

**also have**  $\dots = 2 * (\sum e \in \#\{ \#e \in \# \text{ edges } G. \varrho(snd\ e) < \varrho(fst\ e)\#\}. |(f'(snd\ e))^2 - (f'(fst\ e))^2|)$   
**by** *simp*

**also have**  $\dots = 2 * (\sum a | a \in \text{arcs } G \wedge \varrho(tail\ G\ a) > \varrho(head\ G\ a). |f'(head\ G\ a)^{\wedge 2} - f'(tail\ G\ a)^{\wedge 2}|)$   
**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**  $\dots = 2 *$   
 $(\sum a | a \in \text{arcs } G \wedge \varrho(tail\ G\ a) > \varrho(head\ G\ a). |\tau(\varrho(head\ G\ a))^{\wedge 2} - \tau(\varrho(tail\ G\ a))^{\wedge 2}|)$   
**unfolding**  $\tau\text{-def}$  **using**  $\varphi\text{-}\varrho\text{-inv}$   $\varrho\text{-lt-n}$

**by** (*intro arg-cong2*[**where**  $f = (*)$ ] *sum.cong refl*) *auto*  
**also have**  $\dots = 2 * (\sum a | a \in \text{arcs } G \wedge \varrho(tail\ G\ a) > \varrho(head\ G\ a). \tau(\varrho(head\ G\ a))^{\wedge 2} - \tau(\varrho(tail\ G\ a))^{\wedge 2})$

**using**  $\tau\text{-antimono}$  *power-mono*  $\tau\text{-nonneg}$   
**by** (*intro arg-cong2*[**where**  $f = (*)$ ] *sum.cong refl abs-of-nonneg*)(*auto*)

**also have**  $\dots = 2 *$   
 $(\sum a | a \in \text{arcs } G \wedge \varrho(tail\ G\ a) > \varrho(head\ G\ a). (-(\tau(\varrho(tail\ G\ a))^{\wedge 2})) - (-(\tau(\varrho(head\ G\ a))^{\wedge 2})))$   
**by** (*simp add:algebra-simps*)

**also have**  $\dots = 2 * (\sum a | a \in \text{arcs } G \wedge \varrho(tail\ G\ a) > \varrho(head\ G\ a).$   
 $(\sum i = \varrho(head\ G\ a)..<\varrho(tail\ G\ a). (-(\tau(Suc\ i)^{\wedge 2})) - (-(\tau\ i^{\wedge 2}))))$   
**by** (*intro arg-cong2*[**where**  $f = (*)$ ] *sum.cong refl sum-Suc-diff'*[*symmetric*]) *auto*

**also have**  $\dots = 2 * (\sum (a, i) \in (\text{SIGMA } x: \{a \in \text{arcs } G. \varrho(\text{head } G \ a) < \varrho(\text{tail } G \ a)\}. \{ \varrho(\text{head } G \ x) .. < \varrho(\text{tail } G \ x)\}). \tau \ i^{\wedge} 2 - \tau (\text{Suc } i)^{\wedge} 2)$   
**by** *(subst sum.Sigma) auto*  
**also have**  $\dots = 2 * (\sum p \in \{(a, i). a \in \text{arcs } G \wedge \varrho(\text{head } G \ a) \leq i \wedge i < \varrho(\text{tail } G \ a)\}. \tau(\text{snd } p)^{\wedge} 2 - \tau(\text{fst } p)^{\wedge} 2)$   
**by** *(intro arg-cong2[where f=(\*)] sum.cong refl) (auto simp add:Sigma-def)*  
**also have**  $\dots = 2 * (\sum p \in \{(i, a). a \in \text{arcs } G \wedge \varrho(\text{head } G \ a) \leq i \wedge i < \varrho(\text{tail } G \ a)\}. \tau(\text{fst } p)^{\wedge} 2 - \tau(\text{snd } p)^{\wedge} 2)$   
**by** *(intro sum.reindex-cong[where l=prod.swap] arg-cong2[where f=(\*)]) auto*  
**also have**  $\dots = 2 * (\sum (i, a) \in (\text{SIGMA } x: \{.. < n\}. \{a \in \text{arcs } G. \varrho(\text{head } G \ a) \leq x \wedge x < \varrho(\text{tail } G \ a)\}). \tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2)$   
**using** *less-trans[OF - ρ-lt-n] by (intro sum.cong arg-cong2[where f=(\*)]) auto*  
**also have**  $\dots = 2 * (\sum i < n. (\sum a | a \in \text{arcs } G \wedge \varrho(\text{head } G \ a) \leq i \wedge i < \varrho(\text{tail } G \ a). \tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**by** *(subst sum.Sigma) auto*  
**also have**  $\dots = 2 * (\sum i < n. \text{card } \{a \in \text{arcs } G. \varrho(\text{head } G \ a) \leq i \wedge i < \varrho(\text{tail } G \ a)\} * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**by** *simp*  
**also have**  $\dots = 2 * (\sum i < n. \text{card } \{a \in \text{arcs } G. \varrho(\text{head } G \ a) \leq i \wedge \neg(\varrho(\text{tail } G \ a) \leq i)\} * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**by** *(intro-cong [σ₂ (\*), σ₁ card, σ₁ of-nat] more:sum.cong Collect-cong) auto*  
**also have**  $\dots = 2 * (\sum i < n. \text{card } \{a \in \text{arcs } G. \text{head } G \ a \in \varphi\{..i\} \wedge \text{tail } G \ a \notin \varphi\{..i\}\} * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**using** *4*  
**by** *(intro-cong [σ₂ (\*), σ₁ card, σ₁ of-nat, σ₂ (∧)] more:sum.cong restr-Collect-cong) auto*  
**also have**  $\dots = 2 * (\sum i < n. \text{real } (\text{card } (\text{edges-betw } (-\varphi\{..i\}) (\varphi\{..i\}))) * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**unfolding** *edges-betw-def by (auto simp:conj.commute)*  
**also have**  $\dots = 2 * (\sum i < n. \text{real } (\text{card } (\text{edges-betw } (\varphi\{..i\}) (-\varphi\{..i\}))) * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**using** *edges-betw-sym by simp*  
**also have**  $\dots = 2 * (\sum i < m. \text{real } (\text{card } (\text{edges-betw } (\varphi\{..i\}) (-\varphi\{..i\}))) * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**using** *τ-supp m-le-n by (intro sum.mono-neutral-right arg-cong2[where f=(\*)]) auto*  
**finally have** *Bf-eq:*  
 $B_f = 2 * (\sum i < m. \text{real } (\text{card } (\text{edges-betw } (\varphi\{..i\}) (-\varphi\{..i\}))) * (\tau \ i^{\wedge} 2 - \tau (i+1)^{\wedge} 2))$   
**by** *simp*

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

**qed**

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

```

    using  $\tau$ -supp[of m] by simp
  also have ...  $\leq 2 * \Lambda_e * ((\sum i < m. \tau i^2) + (\sum i < m. i * \tau i^2 - (\text{Suc } i) * \tau (\text{Suc } i)^2))$ 
    by (subst sum-lessThan-telescope'[symmetric]) simp
  also have ...  $\leq 2 * (\sum i < m. (\Lambda_e * (i+1)) * (\tau i^2 - \tau (i+1)^2))$ 
    by (simp add:sum-distrib-left algebra-simps sum.distrib[symmetric])
  also have ...  $\leq 2 * (\sum i < m. \text{real } (\text{card } (\text{edges-betw } (\varphi \{..i\}) (-\varphi \{..i\}))) * (\tau i^2 - \tau (i+1)^2))$ 
    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:  $2 * \Lambda_e * \text{norm } f^2 \leq B_f$ 
    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  $(\sum i \in \text{UNIV}. 0 - g \ \$h \ i) = 0$ 
    using g-orth unfolding sum-subtractf inner-vec-def by auto
  ultimately have  $\forall x \in \text{UNIV}. -(g \ \$h \ x) = 0$ 
    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:  $\text{norm } f > 0$ 
  by simp

```

```

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

```

end

```

context regular-graph
begin

```

```

lemmas 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**  $\Lambda_e \in \{d * (1 - \Lambda_2) / 2.. d * \text{sqrt} (2 * (1 - \Lambda_2))\}$   
**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-Pros.Approximation*

*Expander-Graphs-Definition*

**begin**

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

**fun** *mgg-graph-step* ::  $\text{nat} \Rightarrow (\text{int} \times \text{int}) \Rightarrow (\text{nat} \times \text{int}) \Rightarrow (\text{int} \times \text{int})$

**where** *mgg-graph-step*  $n$  ( $i, j$ ) ( $l, \sigma$ ) =

[ (( $i + \sigma * (2 * j + 0)$ ) mod  $\text{int } n$ ,  $j$ ), ( $i$ , ( $j + \sigma * (2 * i + 0)$ ) mod  $\text{int } n$ )  
, (( $i + \sigma * (2 * j + 1)$ ) mod  $\text{int } n$ ,  $j$ ), ( $i$ , ( $j + \sigma * (2 * i + 1)$ ) mod  $\text{int } n$ ) ] !  $l$

**definition** *mgg-graph* ::  $\text{nat} \Rightarrow (\text{int} \times \text{int}, (\text{int} \times \text{int}, \text{nat} \times \text{int}) \text{arc})$  *pre-digraph* **where**

*mgg-graph*  $n$  =

(| *verts* =  $\{0..<n\} \times \{0..<n\}$ ,

*arcs* =  $(\lambda(t, l). (\text{Arc } t (\text{mgg-graph-step } n \ t \ l) \ l))'(\{0..<\text{int } n\} \times \{0..<\text{int } n\}) \times (\{..<4\} \times \{-1, 1\})$ ,

*tail* = *arc-tail*,

*head* = *arc-head* |)

**locale** *margulis-gaber-galil* =

**fixes**  $m :: \text{nat}$

**assumes** *m-gt-0*:  $m > 0$

**begin**

**abbreviation** *G* **where**  $G \equiv \text{mgg-graph } m$

**lemma** *wf-digraph*: *wf-digraph* (*mgg-graph*  $m$ )

**proof** –

**have**

*tail* (*mgg-graph*  $m$ )  $e \in \text{verts}$  (*mgg-graph*  $m$ ) (**is** ?*A*)

*head* (*mgg-graph*  $m$ )  $e \in \text{verts}$  (*mgg-graph*  $m$ ) (**is** ?*B*)

**if**  $a:e \in \text{arcs}$  (*mgg-graph*  $m$ ) **for**  $e$

**proof** –

```

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

lemma  $m\gg\text{-finite}$ :  $fin\text{-digraph}\ (m\gg\text{-graph}\ m)$ 
proof –
  have  $finite\ (verts\ (m\gg\text{-graph}\ m))\ finite\ (arcs\ (m\gg\text{-graph}\ m))$ 
  unfolding  $m\gg\text{-graph}\text{-def}$  by  $auto$ 
thus  $?thesis$ 
  using  $wf\text{-digraph}$ 
  unfolding  $fin\text{-digraph}\text{-def}\ fin\text{-digraph}\text{-axioms}\text{-def}$  by  $auto$ 
qed

interpretation  $fin\text{-digraph}\ m\gg\text{-graph}\ m$ 
using  $m\gg\text{-finite}$  by  $simp$ 

definition  $arcs\text{-pos}$  ::  $(int \times int, nat \times int)$   $arc\ set$ 
  where  $arcs\text{-pos} = (\lambda(t,l). (Arc\ t\ (m\gg\text{-graph}\text{-step}\ m\ t\ (l,1))\ (l,1)))\ (verts\ G \times \{..<4\})$ 
definition  $arcs\text{-neg}$  ::  $(int \times int, nat \times int)$   $arc\ set$ 
  where  $arcs\text{-neg} = (\lambda(h,l). (Arc\ (m\gg\text{-graph}\text{-step}\ m\ h\ (l,1))\ h\ (l,-1)))\ (verts\ G \times \{..<4\})$ 

lemma  $arcs\text{-sym}$ :
   $arcs\ G = arcs\text{-pos} \cup arcs\text{-neg}$ 
proof –
  have  $0$ :  $x \in arcs\ G$  if  $x \in arcs\text{-pos}$  for  $x$ 
  using  $that$  unfolding  $arcs\text{-pos}\text{-def}\ m\gg\text{-graph}\text{-def}$  by  $auto$ 
  have  $1$ :  $a \in arcs\ G$  if  $t:a \in arcs\text{-neg}$  for  $a$ 
  proof –
  obtain  $h\ l$  where  $hl\text{-def}$ :  $h \in verts\ G\ l \in \{..<4\}$   $a = Arc\ (m\gg\text{-graph}\text{-step}\ m\ h\ (l,1))\ h\ (l,-1)$ 
  using  $t$  unfolding  $arcs\text{-neg}\text{-def}$  by  $auto$ 

  define  $t$  where  $t = m\gg\text{-graph}\text{-step}\ m\ h\ (l,1)$ 

  have  $h\text{-ran}$ :  $h \in \{0..<int\ m\} \times \{0..<int\ m\}$ 
  using  $hl\text{-def}(1)$  unfolding  $m\gg\text{-graph}\text{-def}$  by  $simp$ 
  have  $l\text{-ran}$ :  $l \in set\ [0,1,2,3]$ 
  using  $hl\text{-def}(2)$  by  $auto$ 

  have  $t \in \{0..<int\ m\} \times \{0..<int\ m\}$ 
  using  $h\text{-ran}\ l\text{-ran}$ 
  unfolding  $t\text{-def}$  by  $(cases\ h,\ auto\ simp\ add:\text{mod}\text{-simps})$ 
hence  $t\text{-ran}$ :  $t \in verts\ G$ 
  unfolding  $m\gg\text{-graph}\text{-def}$  by  $simp$ 

```

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

**have**  $\text{card } (\text{arcs-pos} \cup \text{arcs-neg}) = \text{card } \text{arcs-pos} + \text{card } \text{arcs-neg}$   
**unfolding**  $\text{arcs-pos-def arcs-neg-def}$  **by**  $(\text{intro card-Un-disjoint finite-imageI}) \text{auto}$   
**also have**  $\dots = \text{card } (\text{verts } G \times \{..<4::\text{nat}\}) + \text{card } (\text{verts } G \times \{..<4::\text{nat}\})$   
**unfolding**  $\text{arcs-pos-def arcs-neg-def}$   
**by**  $(\text{intro arg-cong2}[\text{where } f=(+)] \text{card-image inj-onI}) \text{auto}$   
**also have**  $\dots = \text{card } (\text{verts } G \times \{..<4::\text{nat}\} \times \{-1, 1::\text{int}\})$   
**by**  $\text{simp}$   
**also have**  $\dots = \text{card } ((\lambda(t, l). \text{Arc } t \ (\text{mgg-graph-step } m \ t \ l) \ l) \ '(\text{verts } G \times \{..<4\} \times \{-1, 1\}))$   
**by**  $(\text{intro card-image[symmetric] inj-onI}) \text{auto}$   
**also have**  $\dots = \text{card } (\text{arcs } G)$   
**unfolding**  $\text{mgg-graph-def}$  **by**  $\text{simp}$   
**finally have**  $\text{card } (\text{arcs-pos} \cup \text{arcs-neg}) = \text{card } (\text{arcs } G)$   
**by**  $\text{simp}$   
**hence**  $\text{arcs-pos} \cup \text{arcs-neg} = \text{arcs } G$   
**using**  $0 \ 1$  **by**  $(\text{intro card-subset-eq, auto})$   
**thus**  $?thesis$  **by**  $\text{simp}$   
**qed**

**lemma**  $\text{sym: symmetric-multi-graph } (\text{mgg-graph } m)$

**proof** –

**define**  $f :: (\text{int} \times \text{int}, \text{nat} \times \text{int}) \text{arc} \Rightarrow (\text{int} \times \text{int}, \text{nat} \times \text{int}) \text{arc}$   
**where**  $f \ a = \text{Arc } (\text{arc-head } a) \ (\text{arc-tail } a) \ (\text{apsnd } (\lambda x. (-1) * x) \ (\text{arc-label } a))$  **for**  $a$

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

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

**have**  $c: \text{bij-betw } f \ (\text{arcs-pos} \cup \text{arcs-neg}) \ (\text{arcs-neg} \cup \text{arcs-pos})$   
**by**  $(\text{intro bij-betw-combine}[OF \ a \ b]) \ (\text{auto simp add:arcs-pos-def arcs-neg-def})$

**hence**  $c: \text{bij-betw } f \ (\text{arcs } G) \ (\text{arcs } G)$   
**unfolding**  $\text{arcs-sym}$  **by**  $(\text{subst } (2) \ \text{sup-commute, simp})$

**show**  $?thesis$

**by**  $(\text{intro symmetric-multi-graphI}[\text{where } f=f] \ \text{fin-digraph-axioms } c)$   
 $(\text{simp add:f-def mgg-graph-def})$

**qed**

**lemma**  $\text{out-deg:}$

**assumes**  $v \in \text{verts } G$

**shows**  $\text{out-degree } G \ v = 8$

**proof** –

**have**  $\text{out-degree } (\text{mgg-graph } m) \ v = \text{card } (\text{out-arcs } (\text{mgg-graph } m) \ v)$   
**unfolding**  $\text{out-degree-def}$  **by**  $\text{simp}$

**also have**  $\dots = \text{card } \{e. (\exists w \in \text{verts } (\text{mgg-graph } m). \exists l \in \{..<4\} \times \{-1, 1\}. e = \text{Arc } w \ (\text{mgg-graph-step } m \ w \ l) \ l \wedge \text{arc-tail } e = v)\}$

**unfolding**  $\text{mgg-graph-def out-arcs-def}$  **by**  $(\text{simp add:image-iff})$

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

**sublocale** *regular-graph* *m-gg-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*:  $v \in \text{verts } G$   
**using** *verts-ne* **by** *auto*  
**hence**  $0:(\text{SOME } v. v \in \text{verts } G) \in \text{verts } G$   
**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**  $g\text{-inner } (\lambda x. (\sum i \in I. f \ i \ x)) \ g = (\sum i \in I. g\text{-inner } (f \ i) \ g)$   
**using** *assms* **by** (*induction I* *rule:finite-induct*) (*auto simp add:g-inner-simps*)

**lemma** *g-inner-sum-right*:  
**assumes** *finite*  $I$   
**shows**  $g\text{-inner } f \ (\lambda x. (\sum i \in I. g \ i \ x)) = (\sum i \in I. g\text{-inner } f \ (g \ i))$   
**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**  $g\text{-inner } f \ g = g\text{-inner } (\lambda x. (f \ (h \ x))) \ (\lambda x. (g \ (h \ x)))$   
**unfolding** *g-inner-def*  
**by** (*subst sum.reindex-bij-betw*[*OF assms,symmetric*]) *simp*

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

**lemma**  *$\omega_F$ -simps*:  
 $\omega_F \ (x + y) = \omega_F \ x * \omega_F \ y$   
 $\omega_F \ (x - y) = \omega_F \ x * \omega_F \ (-y)$   
 $\text{cnj } (\omega_F \ x) = \omega_F \ (-x)$   
**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**  $x \ y :: \text{int}$

**assumes**  $x \bmod m = y \bmod m$   
**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**  $\dots = \omega_F (of-int\ x) * \omega_F (of-int\ (m*z))$   
**by** (*simp add:ω<sub>F</sub>-simps*)  
**also have**  $\dots = \omega_F (of-int\ x) * cis\ (2 * pi * of-int\ (z))$   
**unfolding** *ω<sub>F</sub>-def* **using** *m-gt-0*  
**by** (*intro arg-cong2*[**where**  $f=(*)$ ] *arg-cong*[**where**  $f=cis$ ]) *auto*  
**also have**  $\dots = \omega_F (of-int\ x) * 1$   
**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**  $\dots = 1$   
**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** *ω<sub>F</sub>-eq-1-iff*:  
**fixes**  $x :: int$   
**shows**  $\omega_F\ x = 1 \iff x \bmod m = 0$   
**proof**  
**assume**  $\omega_F (real-of-int\ x) = 1$   
**hence**  $cis\ (2 * pi * real-of-int\ x / real\ m) = 1$   
**unfolding** *ω<sub>F</sub>-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**  $x \bmod m = 0$  **by** *simp*  
**next**  
**assume**  $x \bmod m = 0$   
**hence**  $\omega_F\ x = \omega_F (of-int\ 0)$   
**by** (*intro ω<sub>F</sub>-cong*) *auto*  
**also have**  $\dots = 1$  **unfolding** *ω<sub>F</sub>-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**  $\bigwedge x. x \in \text{verts } G \implies f x = g x$   
**shows**  $FT f = FT g$   
**unfolding** *FT-def* **by** (*intro ext g-inner-cong assms refl*)

**lemma** *parseval*:  
 $g\text{-inner } f g = g\text{-inner } (FT f) (FT g) / m^{\wedge} 2$  (**is**  $?L = ?R$ )

**proof** –

**define**  $\delta :: (int \times int) \Rightarrow (int \times int) \Rightarrow \text{complex}$  **where**  $\delta x y = \text{of-bool } (x = y)$  **for**  $x y$

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

**have** *1*:  $(\sum x=0..<int m. \omega_F (z*x)) = m * \text{of-bool}(z \bmod m = 0)$  (**is**  $?L1 = ?R1$ ) **for**  $z :: int$

**proof** (*cases z mod m = 0*)

**case** *True*

**have**  $(\sum x=0..<int m. \omega_F (z*x)) = (\sum x=0..<int m. \omega_F (\text{of-int } 0))$

**using** *True* **by** (*intro sum.cong  $\omega_F$ -cong refl*) *auto*

**also have**  $\dots = m * \text{of-bool}(z \bmod m = 0)$

**unfolding**  $\omega_F$ -def *True* **by** *simp*

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

**next**

**case** *False*

**have**  $(1 - \omega_F z) * ?L1 = (1 - \omega_F z) * (\sum x \in int \text{ ‘ } \{..<m\}. \omega_F(z*x))$

**by** (*intro arg-cong2[where f=(\*)] sum.cong refl*)

(*simp add: image-atLeastZeroLessThan-int*)

**also have**  $\dots = (\sum x < m. \omega_F(z*real x) - \omega_F(z*(real (Suc x))))$

**by** (*subst sum.reindex, auto simp add:algebra-simps sum-distrib-left  $\omega_F$ -simps*)

**also have**  $\dots = \omega_F (z * 0) - \omega_F (z * m)$

**by** (*subst sum-lessThan-telescope'*) *simp*

**also have**  $\dots = \omega_F (\text{of-int } 0) - \omega_F (\text{of-int } 0)$

**by** (*intro arg-cong2[where f=(-)]  $\omega_F$ -cong*) *auto*

**also have**  $\dots = 0$

**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*  $(\delta v) (\delta w) = g\text{-inner } (FT (\delta v)) (FT (\delta w)) / m^{\wedge} 2$  (**is**  $?L1 = ?R1/-$ )

**if**  $v \in \text{verts } G$   $w \in \text{verts } G$  **for**  $v$   $w$   
**proof** –  
**have**  $?R1 = g\text{-inner}(\lambda x. \omega_F(-(fst\ v *fst\ x +snd\ v * snd\ x)))(\lambda x. \omega_F(-(fst\ w *fst\ x +snd\ w * snd\ x)))$   
**using that by** (*intro g-inner-cong, auto simp add:FT- $\delta$* )  
**also have**  $... = (\sum (x,y) \in \{0..<int\ m\} \times \{0..<int\ m\}. \omega_F((fst\ w -fst\ v)*x) * \omega_F((snd\ w - snd\ v)*y))$   
**unfolding g-inner-def by** (*simp add: $\omega_F$ -simps algebra-simps case-prod-beta mgg-graph-def*)  
**also have**  $... = (\sum x=0..<int\ m. \sum y = 0..<int\ m. \omega_F((fst\ w - fst\ v)*x) * \omega_F((snd\ w - snd\ v) * y))$   
**by** (*subst sum.cartesian-product[symmetric] simp*)  
**also have**  $... = (\sum x=0..<int\ m. \omega_F((fst\ w - fst\ v)*x)) * (\sum y = 0..<int\ m. \omega_F((snd\ w - snd\ v) * y))$   
**by** (*subst sum.swap*) (*simp add:sum-distrib-left sum-distrib-right*)  
**also have**  $... = of\text{-nat} (m * of\text{-bool}(fst\ v \text{ mod } m = fst\ w \text{ mod } m)) * of\text{-nat} (m * of\text{-bool}(snd\ v \text{ mod } m = snd\ w \text{ mod } m))$   
**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**  $... = m^2 * of\text{-bool}(v = w)$   
**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  $\delta$ -def by simp**  
**finally have**  $?R1 = m^2 * ?L1$  **by simp**  
**thus ?thesis using m-gt-0 by simp**  
**qed**

**have**  $?L = g\text{-inner} (\lambda x. (\sum v \in \text{verts } G. (f\ v) * \delta\ v\ x)) (\lambda x. (\sum v \in \text{verts } G. (g\ v) * \delta\ v\ x))$   
**unfolding  $\delta$ -def by** (*intro g-inner-cong*) *auto*  
**also have**  $... = (\sum v \in \text{verts } G. (f\ v) * (\sum w \in \text{verts } G. cnj (g\ w) * g\text{-inner} (\delta\ v) (\delta\ w)))$   
**by** (*simp add:g-inner-simps g-inner-sum-left g-inner-sum-right*)  
**also have**  $... = (\sum v \in \text{verts } G. (f\ v) * (\sum w \in \text{verts } G. cnj (g\ w) * g\text{-inner}(FT (\delta\ v)) (FT (\delta\ w)))) / m^2$   
**by** (*simp add:0 sum-divide-distrib sum-distrib-left algebra-simps*)  
**also have**  $... = g\text{-inner}(\lambda x. (\sum v \in \text{verts } G. (f\ v) * FT (\delta\ v) x)) (\lambda x. (\sum v \in \text{verts } G. (g\ v) * FT (\delta\ v) x)) / m^2$   
**by** (*simp add:g-inner-simps g-inner-sum-left g-inner-sum-right*)  
**also have**  $... = g\text{-inner}(FT(\lambda x. (\sum v \in \text{verts } G. (f\ v) * \delta\ v\ x))) (FT(\lambda x. (\sum v \in \text{verts } G. (g\ v) * \delta\ v\ x))) / m^2$   
**by** (*intro g-inner-cong arg-cong2[where f=(/)]*) (*simp-all add: FT-sum FT-scale*)  
**also have**  $... = g\text{-inner} (FT\ f) (FT\ g) / m^2$   
**unfolding  $\delta$ -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:**

$$(\sum v \in \text{verts } G. \text{norm } (f\ v)^2) = (\sum v \in \text{verts } G. \text{norm } (FT\ f\ v)^2) / m^2 \text{ (is } ?L = ?R)$$

**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**  $... = g\text{-inner} (FT\ f) (FT\ f) / m^2$   
**by** (*subst parseval*) *simp*  
**also have**  $... = \text{complex-of-real } ?R$   
**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*:

$FT (\lambda x. f (snd\ x, fst\ x)) (u, v) = FT\ f (v, u)$

**proof** –

**have**  $0:bij\text{-}betw\ (\lambda(x::int \times int). (snd\ x, fst\ x))\ (verts\ G)\ (verts\ G)$

**by**  $(intro\ bij\text{-}betwI[\mathbf{where}\ g=(\lambda(x::int \times int). (snd\ x, fst\ x))])$

$(auto\ simp\ add:mgg\text{-}graph\text{-}def)$

**show** *?thesis*

**unfolding** *FT-def*

**by**  $(subst\ g\text{-}inner\text{-}reindex[OF\ 0])\ (simp\ add:algebra\text{-}simps)$

**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**  $periodic\ f = (\forall x\ y. f\ (x, y) = f\ (x\ mod\ int\ m, y\ mod\ int\ m))$

**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*  $(\lambda x. g\ (f\ x))$

**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**  $(intro\ g\text{-}inner\text{-}cong\ \omega_F\text{-}cong\ ext)$

$(auto\ simp\ add:mod\text{-}simps\ cong:mod\text{-}add\text{-}cong)$

**thus** *?thesis*

**unfolding** *periodic-def* **by** *simp*

**qed**

**lemma** *FT-sheer-ax*:

**fixes**  $u\ v\ c\ d :: int$

**assumes** *periodic f*

**shows**  $FT\ (\lambda x. f\ (fst\ x, snd\ x+c*fst\ x+d))\ (u, v) = \omega_F\ (d* v) * FT\ f\ (u-c* v, v)$

$(is\ ?L = ?R)$

**proof** –

**define**  $s$  **where**  $s = (\lambda(x, y). (x, (y - c * x - d)\ mod\ m))$

**define**  $s0$  **where**  $s0 = (\lambda(x, y). (x, (y - c * x)\ mod\ m))$

**define**  $s1$  **where**  $s1 = (\lambda(x::int, y). (x, (y - d)\ mod\ m))$

**have**  $0:bij\text{-}betw\ s0\ (verts\ G)\ (verts\ G)$

**by**  $(intro\ bij\text{-}betwI[\mathbf{where}\ g=\lambda(x, y). (x, (y+c*x)\ mod\ m)])$

$(auto\ simp\ add:mgg\text{-}graph\text{-}def\ s0\text{-}def\ Pi\text{-}def\ mod\text{-}simps)$

**have** 1: *bij-betw* *s1* (*verts G*) (*verts G*)  
**by** (*intro* *bij-betwI*[**where**  $g = \lambda(x,y). (x, (y+d) \bmod m)$ ])  
(*auto simp add: mgg-graph-def s1-def Pi-def mod-simps*)  
**have** 2:  $s = (s1 \circ s0)$   
**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:  $(snd (s x) + c * fst x + d) \bmod int m = snd x \bmod m$  **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* ( $\lambda x. f (fst x, snd x + c * fst x + d)$ ) ( $\lambda x. \omega_F (fst x * u + snd x * v)$ )  
**unfolding** *FT-altdef* **by** *simp*  
**also have** ... = *g-inner* ( $\lambda x. f (fst x, (snd x + c * fst x + d) \bmod m)$ ) ( $\lambda x. \omega_F (fst x * u + snd x * v)$ )  
*v*)  
**by** (*intro* *g-inner-cong* *periodic-cong*[*OF* *assms*]) (*auto simp add: algebra-simps*)  
**also have** ... = *g-inner* ( $\lambda x. f (fst x, snd x \bmod m)$ ) ( $\lambda x. \omega_F (fst x * u + snd (s x) * v)$ )  
**by** (*subst* *g-inner-reindex*[*OF* 3]) (*simp add: 4 5*)  
**also have** ... =  
*g-inner* ( $\lambda x. f (fst x, snd x \bmod m)$ ) ( $\lambda x. \omega_F (fst x * u + ((snd x - c * fst x - d) \bmod m) * v)$ )  
**by** (*simp add: s-def case-prod-beta*)  
**also have** ... = *g-inner* *f* ( $\lambda x. \omega_F (fst x * (u - c * v) + snd x * v - d * v)$ )  
**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* ( $\lambda x. \omega_F (-d * v) * \omega_F (fst x * (u - c * v) + snd x * v)$ )  
**by** (*simp add:  $\omega_F$ -simps algebra-simps*)  
**also have** ... =  $\omega_F (d * v) * g$ -*inner* *f* ( $\lambda x. \omega_F (fst x * (u - c * v) + snd x * v)$ )  
**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* ( $\lambda x. f (fst x, snd x + c * fst x + d)$ ) (*u, v*) =  $\omega_F (d * v) * FT f (u - c * v, v)$  (**is** ?A)

*FT* ( $\lambda x. f (fst x, snd x + c * fst x)$ ) (*u, v*) = *FT f* (*u - c \* v, v*) (**is** ?B)

*FT* ( $\lambda x. f (fst x + c * snd x + d, snd x)$ ) (*u, v*) =  $\omega_F (d * u) * FT f (u, v - c * u)$  (**is** ?C)

*FT* ( $\lambda x. f (fst x + c * snd x, snd x)$ ) (*u, v*) = *FT f* (*u, v - c \* u*) (**is** ?D)

**proof** –

**have** 1: *periodic* ( $\lambda x. f (snd x, fst x)$ )

**using** *assms* **unfolding** *periodic-def* **by** *simp*

**have** 0:  $\omega_F 0 = 1$

**unfolding**  $\omega_F$ -*def* **by** *simp*

**show** ?A

**using** *FT-sheer-ax*[*OF* *assms*] **by** *simp*

**show** ?B

**using** 0 *FT-sheer-ax*[*OF* *assms*, **where** *d=0*] **by** *simp*

**show** ?C

**using** *FT-sheer-ax*[*OF* 1] **by** (*subst* (1 2) *FT-swap*[*symmetric*], *simp*)

**show** ?D

**using** 0 *FT-sheer-ax*[*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**  $\gamma\text{-aux} :: int \times int \Rightarrow real \times real$

**where**  $\gamma\text{-aux} x = (|fst x/m - 1/2|, |snd x/m - 1/2|)$

**definition**  $compare :: real \times real \Rightarrow real \times real \Rightarrow bool$

**where**  $compare x y = (fst x \leq fst y \wedge snd x \leq snd y \wedge x \neq y)$

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**  $\alpha\text{-inv}: 1/\alpha = \alpha/2$

**unfolding**  $\alpha\text{-def}$  **by** (*simp add: real-div-sqrt*)

**definition**  $\gamma :: int \times int \Rightarrow int \times int \Rightarrow real$

**where**  $\gamma x y = (if\ compare\ (\gamma\text{-aux}\ x)\ (\gamma\text{-aux}\ y)\ then\ \alpha\ else\ (if\ compare\ (\gamma\text{-aux}\ y)\ (\gamma\text{-aux}\ x)\ then\ (1 / \alpha)\ else\ 1))$

**lemma**  $\gamma\text{-sym}: \gamma x y * \gamma y x = 1$

**unfolding**  $\gamma\text{-def}$   $\alpha\text{-def}$   $compare\text{-def}$  **by** (*auto simp add: prod-eq-iff*)

**lemma**  $\gamma\text{-nonneg}: \gamma x y \geq 0$

**unfolding**  $\gamma\text{-def}$   $\alpha\text{-def}$  **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**  $\gamma'\text{-cases}$ :

$abs\ (x-1/2) = abs\ (y-1/2) \implies \gamma' x y = 1$

$abs\ (x-1/2) > abs\ (y-1/2) \implies \gamma' x y = 1/\alpha$

$abs\ (x-1/2) < abs\ (y-1/2) \implies \gamma' x y = \alpha$

**unfolding**  $\gamma'\text{-def}$  **by** *auto*

**lemma**  $if\text{-cong}\text{-direct}$ :

**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**  $\gamma'\text{-cong}$ :

**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**  $\gamma'\text{-def}$

**using** *assms* **by** (*intro if-cong-direct refl*) *auto*

**lemma** *add-swap-cong*:  
**fixes**  $x\ y\ u\ v :: 'a :: ab\text{-semigroup-add}$   
**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**  $frac\ x = (if\ x < (-1)\ then\ (x - [x])\ else\ (if\ x < 0\ then\ (x + 1)\ else\ (if\ x < 1\ then\ x\ else\ (if\ x < 2\ then\ (x - 1)\ else\ (x - [x])))$

**proof** –  
**have** *real-of-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**  $\dots = \{-pi/2..pi/2\}$   
**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**  $\text{apr}: 4 \leq 5 * \text{sqrt} (2::\text{real}) 8 * \cos (\text{pi} / 4) \leq 5 * \text{sqrt} (2::\text{real})$   
**by** (*approximation 5*)+

**have**  $\cos (\text{pi} * x) \geq 0$

**using** *assms(1,2,3)* **by** (*intro cos-pi-ge-0*) *simp*

**moreover have**  $\cos (\text{pi} * y) \geq 0$

**using** *assms(1,2,3)* **by** (*intro cos-pi-ge-0*) *simp*

**ultimately have**  $0: ?L = \cos(\text{pi}*x)*\varphi x y + \cos(\text{pi}*y)*\varphi y x$  (**is**  $- = ?T$ )  
**by** *simp*

**consider** (a)  $x+y < 1/2$  | (b)  $y = 1/2 - x$  | (c)  $x+y > 1/2$  **by** *argo*

**hence**  $?T \leq 2.5 * \text{sqrt} 2$  (**is**  $?T \leq ?R$ )

**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)* **by** *fastforce*

**thus** *?thesis*

**proof** (*cases*)

**case** *1*

**have**  $\varphi x y = \alpha + 1/\alpha$

**unfolding**  $\varphi$ -*def* **using** *1 a*

**by** (*intro arg-cong2[where f=(+)]*  $\gamma'$ -*cases*) (*auto simp add:frac-expand*)

**moreover have**  $\varphi y x = 1/\alpha + 1/\alpha$

**unfolding**  $\varphi$ -*def* **using** *1 a*

**by** (*intro arg-cong2[where f=(+)]*  $\gamma'$ -*cases*) (*auto simp add:frac-expand*)

**ultimately have**  $?T = \cos (\text{pi} * x) * (\alpha + 1/\alpha) + \cos (\text{pi} * y) * (1/\alpha + 1/\alpha)$

**by** *simp*

**also have**  $\dots \leq 1 * (\alpha + 1/\alpha) + 1 * (1/\alpha + 1/\alpha)$

**unfolding**  $\alpha$ -*def* **by** (*intro add-mono mult-right-mono*) *auto*

**also have**  $\dots = ?R$

**unfolding**  $\alpha$ -*def* **by** (*simp add:divide-simps*)

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

**next**

**case** *2*

**have** *y-range*:  $y \in \{0 < .. < 1/2\}$

**using** *assms 2* **by** *simp*

**have**  $\varphi 0 y = 1 + 1$

**unfolding**  $\varphi$ -*def* **using** *y-range*

**by** (*intro arg-cong2[where f=(+)]*  $\gamma'$ -*cases*) (*auto simp add:frac-expand*)

**moreover**

**have**  $|x| * 2 < 1 \iff x < 1/2 \wedge -x < 1/2$  **for**  $x :: \text{real}$  **by** *auto*

**hence**  $\varphi y 0 = 1 / \alpha + 1 / \alpha$

**unfolding**  $\varphi$ -*def* **using** *y-range*

**by** (*intro arg-cong2[where f=(+)]*  $\gamma'$ -*cases*) (*simp-all add:frac-expand*)

**ultimately have**  $?T = 2 + \cos (\text{pi} * y) * (2 / \alpha)$

**unfolding** *2* **by** *simp*

**also have**  $\dots \leq 2 + 1 * (2 / \alpha)$

**unfolding**  $\alpha$ -*def* **by** (*intro add-mono mult-right-mono*) *auto*

**also have**  $\dots \leq ?R$

**unfolding**  $\alpha$ -*def* **by** (*approximation 10*)

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

**next**

**case** *3*

**have**  $\varphi x y = 1 + 1/\alpha$

```

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

```

**also have**  $\dots = 0.8$  **by** *simp*  
**finally have** *cos- $\vartheta$* :  $\cos \vartheta = 0.8$  **by** *simp*  
**have** *sin- $\vartheta$* :  $\sin \vartheta = 0.6$   
**unfolding**  *$\vartheta$ -def* **by** *simp*

**have**  $\varphi x y = \alpha + \alpha$   
**unfolding**  *$\varphi$ -def* **using** *c 1*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**moreover have**  $\varphi y x = 1/\alpha + \alpha$   
**unfolding**  *$\varphi$ -def* **using** *c 1*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**ultimately have**  $?T = \cos(\pi * x) * (2 * \alpha) + \cos(\pi * y) * (\alpha + 1 / \alpha)$   
**by** *simp*  
**also have**  $\dots \leq \cos(\pi * (1/2 - y)) * (2 * \alpha) + \cos(\pi * y) * (\alpha + 1 / \alpha)$   
**unfolding**  *$\alpha$ -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**  $\dots = (2.5 * \alpha) * (\sin(\pi * y) * 0.8 + \cos(\pi * y) * 0.6)$   
**unfolding** *sin-cos-eq  $\alpha$ -inv* **by** (*simp add:algebra-simps*)  
**also have**  $\dots = (2.5 * \alpha) * \sin(\pi * y + \vartheta)$   
**unfolding** *sin-add cos- $\vartheta$  sin- $\vartheta$*   
**by** (*intro arg-cong2*[**where**  $f=(*)$ ] *arg-cong2*[**where**  $f=(+)$ ] *refl*)  
**also have**  $\dots \leq (?R) * 1$   
**unfolding**  *$\alpha$ -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**  $\varphi x y = \alpha + \alpha$   
**unfolding**  *$\varphi$ -def 2* **using** *x-range*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**moreover have**  $\varphi y x = 1 + 1$   
**unfolding**  *$\varphi$ -def 2* **using** *x-range*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**ultimately have**  $?T = \cos(\pi * x) * (2 * \alpha)$   
**unfolding** *2* **by** *simp*  
**also have**  $\dots \leq 1 * (2 * \text{sqrt } 2)$   
**unfolding**  *$\alpha$ -def* **by** (*intro mult-right-mono*) *auto*  
**also have**  $\dots \leq ?R$   
**by** (*approximation 5*)  
**finally show** *?thesis* **by** *simp*

**next**

**case 3**

**have** *x-range*:  $x \in \{1/4..1/2\}$  **using** *3 c* **by** *simp*  
**hence** *cos-bound*:  $\cos(\pi * x) \leq 0.71$   
**by** (*approximation 10*)  
**have**  $\varphi x y = 1 + \alpha$   
**unfolding**  *$\varphi$ -def 3* **using** *3 c*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**moreover have**  $\varphi y x = 1 + \alpha$   
**unfolding**  *$\varphi$ -def 3* **using** *3 c*  
**by** (*intro arg-cong2*[**where**  $f=(+)$ ]  *$\gamma'$ -cases*) (*auto simp add:frac-expand*)  
**ultimately have**  $?T = 2 * \cos(\pi * x) * (1 + \alpha)$   
**unfolding** *3* **by** *simp*  
**also have**  $\dots \leq 2 * 0.71 * (1 + \text{sqrt } 2)$   
**unfolding**  *$\alpha$ -def* **by** (*intro mult-right-mono mult-left-mono cos-bound*) *auto*  
**also have**  $\dots \leq ?R$   
**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  $|\cos(\pi*x)|*x y + |\cos(\pi*y)|*y x \leq 2.5 * \text{sqrt } 2$  (is ?L ≤ ?R)
proof (cases  $y < x$ )
  case True
  have ?L =  $|\cos(\pi*y)|*y x + |\cos(\pi*x)|*x y$ 
    by simp
  also have ... ≤ ?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  $|\cos(\pi*x)|*x y + |\cos(\pi*y)|*y x \leq 2.5 * \text{sqrt } 2$  (is ?L ≤ ?R)
proof (cases  $x > 1/2$ )
  case True
  define  $x'$  where  $x' = 1 - x$ 

  have  $|\text{frac } (x - 2 * y) - 1 / 2| = |\text{frac } (1 - x + 2 * y) - 1 / 2|$ 
  proof (cases  $x - 2 * y \in \mathbb{Z}$ )
    case True
    then obtain  $k$  where  $x\text{-eq: } x = 2*y + \text{of-int } k$  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  $|\text{frac } (x + 2 * y) - 1 / 2| = |\text{frac } (1 - x - 2 * y) - 1 / 2|$ 
  proof (cases  $x + 2 * y \in \mathbb{Z}$ )
    case True
    then obtain  $k$  where  $x\text{-eq: } x = \text{of-int } k - 2*y$  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  $\varphi y x = \varphi y x'$ 
  unfolding  $\varphi$ -def  $x'$ -def by (intro  $\gamma'$ -cong add-swap-cong) simp-all

moreover have  $\varphi x y = \varphi x' y$ 
  unfolding  $\varphi$ -def  $x'$ -def
  by (intro  $\gamma'$ -cong add-swap-cong refl arg-cong[where  $f=(\lambda x. \text{abs } (x-1/2))$ ]) 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  $?L = |\cos(\pi*x')|*\varphi x' y + |\cos(\pi*y)|*\varphi y x'$ 
  by simp
also have  $\dots \leq ?R$ 
  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  $x \in \{0..<1\}$   $y \in \{0..<1\}$   $(x,y) \neq (0,0)$ 
  shows  $|\cos(\pi*x)|*\varphi x y + |\cos(\pi*y)|*\varphi y x \leq 2.5 * \text{sqrt } 2$  (is  $?L \leq ?R$ )
proof (cases  $y > 1/2$ )
  case True
  define  $y'$  where  $y' = 1 - y$ 

  have  $|\text{frac } (y - 2 * x) - 1 / 2| = |\text{frac } (1 - y + 2 * x) - 1 / 2|$ 
  proof (cases  $y - 2 * x \in \mathbb{Z}$ )
    case True
    then obtain  $k$  where  $y$ -eq:  $y = 2*x + \text{of-int } k$  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  $1 - y + 2 * x \notin \mathbb{Z}$ 
      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  $|\text{frac } (y + 2 * x) - 1 / 2| = |\text{frac } (1 - y - 2 * x) - 1 / 2|$ 
  proof (cases  $y + 2 * x \in \mathbb{Z}$ )
    case True
    then obtain  $k$  where  $y$ -eq:  $y = \text{of-int } k - 2*x$  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  $1 - y - 2 * x \notin \mathbb{Z}$ 
      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  $\varphi x y = \varphi x y'$   
 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  $f=(\lambda x. \text{abs } (x-1/2))$ ]) 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  $\dots \leq ?R$   
 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  $x :: \text{int}$   
 shows *real-of-int*  $(x \bmod m) = m * \text{frac } (x/m)$  (is  $?L = ?R$ )  
**proof** –  
 obtain  $y$  where  $y$ -def:  $x \bmod m = x + \text{int } m * y$   
 by (metis *mod-eqE mod-mod-trivial*)  
  
 have  $0: x \bmod \text{int } m < m$   $x \bmod \text{int } m \geq 0$   
 using *m-gt-0* by auto  
  
 have  $?L = \text{real } m * (\text{of-int } (x \bmod m) / m)$   
 using *m-gt-0* by (simp add:algebra-simps)  
 also have  $\dots = \text{real } m * \text{frac } (\text{of-int } (x \bmod m) / m)$   
 using  $0$  by (subst *iffD2[OF frac-eq]*) auto  
 also have  $\dots = \text{real } m * \text{frac } (x / m + y)$   
 unfolding  $y$ -def using *m-gt-0* by (simp add:divide-simps *mult.commute*)  
 also have  $\dots = ?R$   
 unfolding *frac-def* by simp  
 finally show *?thesis* by simp  
qed  
  
**lemma** *fun-bound*:  
 assumes  $v \in \text{verts } G$   $v \neq (0,0)$   
 shows  $\tau(\text{fst } v)*(\gamma v (S_2 v)+\gamma v (T_2 v))+\tau(\text{snd } v)*(\gamma v (S_1 v)+\gamma v (T_1 v)) \leq 2.5 * \text{sqrt } 2$   
 (is  $?L \leq ?R$ )  
**proof** –  
 obtain  $x y$  where  $v$ -def:  $v = (x,y)$  by (cases  $v$ ) auto  
 define  $x'$  where  $x' = x/\text{real } m$   
 define  $y'$  where  $y' = y/\text{real } m$   
  
 have  $0:\gamma v (S_1 v) = \gamma' x' (\text{frac}(x'-2*y'))$   
 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  $1:\gamma v (T_1 v) = \gamma' x' (\text{frac}(x'+2*y'))$   
 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: $\gamma v (S_2 v) = \gamma' y' (\text{frac}(y'-2*x'))$   
 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: $\gamma v (T_2 v) = \gamma' y' (\text{frac}(y'+2*x'))$   
 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:  $\tau (\text{fst } v) = |\cos(\text{pi}*x')|$   $\tau (\text{snd } v) = |\cos(\text{pi}*y')|$   
 unfolding  $\tau$ -def v-def  $x'$ -def  $y'$ -def by auto

have  $x \in \{0..<\text{int } m\}$   $y \in \{0..<\text{int } m\}$   $(x,y) \neq (0,0)$   
 using assms unfolding v-def m-gg-graph-def by auto  
 hence 5: $x' \in \{0..<1\}$   $y' \in \{0..<1\}$   $(x',y') \neq (0,0)$   
 unfolding  $x'$ -def  $y'$ -def by auto

have ?L =  $|\cos(\text{pi}*x')|*\varphi x' y' + |\cos(\text{pi}*y')|*\varphi y' x'$   
 unfolding 0 1 2 3 4  $\varphi$ -def by simp  
 also have ...  $\leq ?R$   
 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  $f :: \text{int} \times \text{int} \Rightarrow \text{real}$   
 assumes  $\bigwedge x. f x \geq 0$   
 assumes  $f (0,0) = 0$   
 assumes periodic f  
 shows  $g\text{-inner } f (\lambda x. f(S_2 x)*\tau (\text{fst } x)+f(S_1 x)*\tau (\text{snd } x)) \leq 1.25* \text{sqrt } 2*g\text{-norm } f^2$   
 (is ?L  $\leq$  ?R)

proof –

have 0:  $2 * f x * f y \leq \gamma x y * f x^2 + \gamma y x * f y^2$  (is ?L1  $\leq$  ?R1) for  $x y$

proof –

have 0  $\leq ((\text{sqrt } (\gamma x y) * f x) - (\text{sqrt } (\gamma y x) * f y))^2$

by simp

also have ... = ?R1 -  $2 * (\text{sqrt } (\gamma x y) * f x) * (\text{sqrt } (\gamma y x) * f y)$

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 -  $2 * \text{sqrt } (\gamma x y * \gamma y x) * f x * f y$

unfolding real-sqrt-mult by simp

also have ... = ?R1 - ?L1

unfolding  $\gamma$ -sym by simp

finally have 0  $\leq$  ?R1 - ?L1 by simp

thus ?thesis by simp

qed

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

have  $S_2$ -inv [simp]:  $T_2 (S_2 x) = x$  if  $x \in \text{verts } G$  for  $x$   
 using that unfolding  $T_2$ -def  $S_2$ -def m-gg-graph-def  
 by (cases x,simp add:mod-simps)

have  $S_1$ -inv [simp]:  $T_1 (S_1 x) = x$  if  $x \in \text{verts } G$  for  $x$   
 using that unfolding  $T_1$ -def  $S_1$ -def m-gg-graph-def  
 by (cases x,simp add:mod-simps)

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

**have**  $S1\text{-inj}$ :  $\text{inj-on } S_1 \text{ (verts } G)$   
**using**  $S\text{-1-inv}$  **by** ( $\text{intro inj-on-inverseI[where } g=T_1]$ )

**have**  $S_2 \text{ 'verts } G \subseteq \text{verts } G$   
**unfolding**  $\text{mgg-graph-def } S_2\text{-def}$   
**by** ( $\text{intro image-subsetI}$ )  $\text{auto}$   
**hence**  $S2\text{-ran}$ :  $S_2 \text{ 'verts } G = \text{verts } G$   
**by** ( $\text{intro card-subset-eq card-image } S2\text{-inj}$ )  $\text{auto}$

**have**  $S_1 \text{ 'verts } G \subseteq \text{verts } G$   
**unfolding**  $\text{mgg-graph-def } S_1\text{-def}$   
**by** ( $\text{intro image-subsetI}$ )  $\text{auto}$   
**hence**  $S1\text{-ran}$ :  $S_1 \text{ 'verts } G = \text{verts } G$   
**by** ( $\text{intro card-subset-eq card-image } S1\text{-inj}$ )  $\text{auto}$

**have**  $2$ :  $g \ v * f \ v^{\wedge} 2 \leq 2.5 * \text{sqrt } 2 * f \ v^{\wedge} 2$  **if**  $g \ v \leq 2.5 * \text{sqrt } 2 \vee v = (0,0)$  **for**  $v \ g$   
**proof** ( $\text{cases } v=(0,0)$ )  
**case**  $\text{True}$   
**then show**  $?thesis$  **using**  $\text{assms}(2)$  **by**  $\text{simp}$   
**next**  
**case**  $\text{False}$   
**then show**  $?thesis$  **using**  $\text{that}$  **by** ( $\text{intro mult-right-mono}$ )  $\text{auto}$   
**qed**

**have**  $2 * ?L = (\sum v \in \text{verts } G. \tau(\text{fst } v) * (2 * f \ v * f(S_2 \ v))) + (\sum v \in \text{verts } G. \tau(\text{snd } v) * (2 * f \ v * f(S_1 \ v)))$   
**unfolding**  $g\text{-inner-def}$  **by** ( $\text{simp add: algebra-simps sum-distrib-left sum.distrib}$ )  
**also have**  $\dots \leq$   
 $(\sum v \in \text{verts } G. \tau(\text{fst } v) * (\gamma \ v \ (S_2 \ v) * f \ v^{\wedge} 2 + \gamma \ (S_2 \ v) \ v * f(S_2 \ v)^{\wedge} 2)) +$   
 $(\sum v \in \text{verts } G. \tau(\text{snd } v) * (\gamma \ v \ (S_1 \ v) * f \ v^{\wedge} 2 + \gamma \ (S_1 \ v) \ v * f(S_1 \ v)^{\wedge} 2))$   
**unfolding**  $\tau\text{-def}$  **by** ( $\text{intro add-mono sum-mono mult-left-mono } 0$ )  $\text{auto}$   
**also have**  $\dots =$   
 $(\sum v \in \text{verts } G. \tau(\text{fst } v) * \gamma \ v \ (S_2 \ v) * f \ v^{\wedge} 2) + (\sum v \in \text{verts } G. \tau(\text{fst } v) * \gamma \ (S_2 \ v) \ v * f(S_2 \ v)^{\wedge} 2) +$   
 $(\sum v \in \text{verts } G. \tau(\text{snd } v) * \gamma \ v \ (S_1 \ v) * f \ v^{\wedge} 2) + (\sum v \in \text{verts } G. \tau(\text{snd } v) * \gamma \ (S_1 \ v) \ v * f(S_1 \ v)^{\wedge} 2)$   
**by** ( $\text{simp add: sum.distrib algebra-simps}$ )  
**also have**  $\dots =$   
 $(\sum v \in \text{verts } G. \tau(\text{fst } v) * \gamma \ v \ (S_2 \ v) * f \ v^{\wedge} 2) +$   
 $(\sum v \in \text{verts } G. \tau(\text{fst } (S_2 \ v)) * \gamma \ (S_2 \ v) \ (T_2 \ (S_2 \ v)) * f(S_2 \ v)^{\wedge} 2) +$   
 $(\sum v \in \text{verts } G. \tau(\text{snd } v) * \gamma \ v \ (S_1 \ v) * f \ v^{\wedge} 2) +$   
 $(\sum v \in \text{verts } G. \tau(\text{snd } (S_1 \ v)) * \gamma \ (S_1 \ v) \ (T_1 \ (S_1 \ v)) * f(S_1 \ v)^{\wedge} 2)$   
**by** ( $\text{intro arg-cong2[where } f=(+)] \text{ sum.cong refl}$ )  $\text{simp-all}$   
**also have**  $\dots =$   
 $(\sum v \in \text{verts } G. \tau(\text{fst } v) * \gamma \ v \ (S_2 \ v) * f \ v^{\wedge} 2) + (\sum v \in S_2 \text{ 'verts } G. \tau(\text{fst } v) * \gamma \ v \ (T_2 \ v) * f \ v^{\wedge} 2) +$   
 $(\sum v \in \text{verts } G. \tau(\text{snd } v) * \gamma \ v \ (S_1 \ v) * f \ v^{\wedge} 2) + (\sum v \in S_1 \text{ 'verts } G. \tau(\text{snd } v) * \gamma \ v \ (T_1 \ v) * f \ v^{\wedge} 2)$   
**using**  $S1\text{-inj } S2\text{-inj}$  **by** ( $\text{simp add: sum.reindex}$ )  
**also have**  $\dots =$   
 $(\sum v \in \text{verts } G. (\tau(\text{fst } v) * (\gamma \ v \ (S_2 \ v) + \gamma \ v \ (T_2 \ v)) + \tau(\text{snd } v) * (\gamma \ v \ (S_1 \ v) + \gamma \ v \ (T_1 \ v))) * f \ v^{\wedge} 2)$   
**unfolding**  $S1\text{-ran } S2\text{-ran}$  **by** ( $\text{simp add: algebra-simps sum.distrib}$ )  
**also have**  $\dots \leq (\sum v \in \text{verts } G. 2.5 * \text{sqrt } 2 * f \ v^{\wedge} 2)$   
**using**  $\text{fun-bound}$  **by** ( $\text{intro sum-mono } 2$ )  $\text{auto}$   
**also have**  $\dots \leq 2.5 * \text{sqrt } 2 * g\text{-norm } f^{\wedge} 2$   
**unfolding**  $g\text{-norm-sq } g\text{-inner-def}$   
**by** ( $\text{simp add: algebra-simps power2-eq-square sum-distrib-left}$ )  
**finally have**  $2 * ?L \leq 2.5 * \text{sqrt } 2 * g\text{-norm } f^{\wedge} 2$  **by**  $\text{simp}$   
**thus**  $?thesis$  **by**  $\text{simp}$   
**qed**

lemma *hoory-8-7*:

```

fixes  $f :: \text{int} \times \text{int} \Rightarrow \text{complex}$ 
assumes  $f (0,0) = 0$ 
assumes periodic  $f$ 
shows  $\text{norm}(g\text{-inner } f (\lambda x. f (S_2 x) * (1 + \omega_F (\text{fst } x)) + f (S_1 x) * (1 + \omega_F (\text{snd } x))))$ 
   $\leq (2.5 * \text{sqrt } 2) * (\sum v \in \text{verts } G. \text{norm } (f v)^{\wedge} 2)$  (is  $?L \leq ?R$ )
proof –
  define  $g :: \text{int} \times \text{int} \Rightarrow \text{real}$  where  $g x = \text{norm } (f x)$  for  $x$ 

  have  $g\text{-zero}: g (0,0) = 0$ 
    using assms(1) unfolding  $g\text{-def}$  by simp
  have  $g\text{-nonneg}: g x \geq 0$  for  $x$ 
    unfolding  $g\text{-def}$  by simp
  have  $g\text{-periodic}: \text{periodic } g$ 
    unfolding  $g\text{-def}$  by (intro periodic-comp[OF assms(2)])

  have  $0: \text{norm}(1 + \omega_F x) = 2 * \tau x$  for  $x :: \text{int}$ 
  proof –
    have  $\text{norm}(1 + \omega_F x) = \text{norm}(\omega_F (-x/2) * (\omega_F 0 + \omega_F x))$ 
      unfolding  $\omega_F\text{-def}$   $\text{norm-mult}$  by simp
    also have  $\dots = \text{norm } (\omega_F (0 - x/2) + \omega_F (x - x/2))$ 
      unfolding  $\omega_F\text{-simps}$  by (simp add: algebra-simps)
    also have  $\dots = \text{norm } (\omega_F (x/2) + \text{cnj } (\omega_F (x/2)))$ 
      unfolding  $\omega_F\text{-simps(3)}$  by (simp add: algebra-simps)
    also have  $\dots = |2 * \text{Re } (\omega_F (x/2))|$ 
      unfolding complex-add-cnj norm-of-real by simp
    also have  $\dots = 2 * |\cos(\pi * x / m)|$ 
      unfolding  $\omega_F\text{-def}$  cis.simps by simp
    also have  $\dots = 2 * \tau x$  unfolding  $\tau\text{-def}$  by simp
    finally show  $?thesis$  by simp
  qed

  have  $?L \leq \text{norm}(\sum v \in \text{verts } G. f v * \text{cnj}(f(S_2 v) * (1 + \omega_F (\text{fst } v)) + f(S_1 v) * (1 + \omega_F (\text{snd } v))))$ 
    unfolding  $g\text{-inner-def}$  by (simp add: case-prod-beta)
  also have  $\dots \leq (\sum v \in \text{verts } G. \text{norm}(f v * \text{cnj}(f(S_2 v) * (1 + \omega_F (\text{fst } v)) + f(S_1 v) * (1 + \omega_F (\text{snd } v))))$ 
    by (intro norm-sum)
  also have  $\dots = (\sum v \in \text{verts } G. g v * \text{norm}(f(S_2 v) * (1 + \omega_F (\text{fst } v)) + f(S_1 v) * (1 + \omega_F (\text{snd } v))))$ 
    unfolding  $\text{norm-mult } g\text{-def}$  complex-mod-cnj by simp
  also have  $\dots \leq (\sum v \in \text{verts } G. g v * (\text{norm}(f(S_2 v) * (1 + \omega_F (\text{fst } v))) + \text{norm}(f(S_1 v) * (1 + \omega_F (\text{snd } v))))$ 
    by (intro sum-mono norm-triangle-ineq mult-left-mono g-nonneg)
  also have  $\dots = 2 * g\text{-inner } g (\lambda x. g(S_2 x) * \tau (\text{fst } x) + g(S_1 x) * \tau (\text{snd } x))$ 
    unfolding  $g\text{-def}$   $g\text{-inner-def}$   $\text{norm-mult } 0$ 
    by (simp add: sum-distrib-left algebra-simps case-prod-beta)
  also have  $\dots \leq 2 * (1.25 * \text{sqrt } 2 * g\text{-norm } g^{\wedge} 2)$ 
    by (intro mult-left-mono hoory-8-8 g-nonneg g-zero g-periodic) auto
  also have  $\dots = ?R$ 
    unfolding  $g\text{-norm-sq } g\text{-def}$   $g\text{-inner-def}$  by (simp add: power2-eq-square)
  finally show  $?thesis$  by simp
  qed

lemma hoory-8-3:
  assumes  $g\text{-inner } f (\lambda \cdot. 1) = 0$ 
  assumes periodic  $f$ 
  shows  $|(\sum (x,y) \in \text{verts } G. f(x,y) * (f(x+2*y,y) + f(x+2*y+1,y) + f(x,y+2*x) + f(x,y+2*x+1)))|$ 
     $\leq (2.5 * \text{sqrt } 2) * g\text{-norm } f^{\wedge} 2$  (is  $|?L| \leq ?R$ )
proof –
  let  $?f = (\lambda x. \text{complex-of-real } (f x))$ 

```

**define**  $Ts :: (int \times int \Rightarrow int \times int)$  list **where**  
 $Ts = [(\lambda(x,y).(x+2*y,y)),(\lambda(x,y).(x+2*y+1,y)),(\lambda(x,y).(x,y+2*x)),(\lambda(x,y).(x,y+2*x+1))]$   
**have**  $p$ : *periodic*  $?f$   
**by** (*intro periodic-comp*[*OF assms*(2)])

**have**  $0$ :  $(\sum T \leftarrow Ts. FT (?f \circ T) v) = FT ?f (S_2 v) * (1 + \omega_F (fst v)) + FT ?f (S_1 v) * (1 + \omega_F (snd v))$   
**(is**  $?L1 = ?R1$ ) **for**  $v :: int \times int$   
**proof** –  
**obtain**  $x y$  **where**  $v$ -*def*:  $v = (x,y)$  **by** (*cases*  $v$ , *auto*)  
**have**  $?L1 = (\sum T \leftarrow Ts. FT (?f \circ T) (x,y))$   
**unfolding**  $v$ -*def* **by** *simp*  
**also have**  $\dots = FT ?f (x,y-2*x) * (1 + \omega_F x) + FT ?f (x-2*y,y) * (1 + \omega_F y)$   
**unfolding**  $Ts$ -*def* **by** (*simp add*:*FT-shear*[*OF p*] *case-prod-beta comp-def*) (*simp add*:*algebra-simps*)  
**also have**  $\dots = ?R1$   
**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**  $\dots = abs (of-nat m^2)$  **by** (*intro norm-of-real*)  
**also have**  $\dots = real m^2$  **by** *simp*  
**finally have**  $1: cmod ((of-nat m)^2) = (real m)^2$  **by** *simp*

**have**  $FT (\lambda x. complex-of-real (f x)) (0, 0) = complex-of-real (g-inner f (\lambda . 1))$   
**unfolding**  $FT$ -*def*  $g$ -*inner-def*  $g$ -*inner-def*  $\omega_F$ -*def* **by** *simp*  
**also have**  $\dots = 0$   
**unfolding** *assms* **by** *simp*  
**finally have**  $2: FT (\lambda x. complex-of-real (f x)) (0, 0) = 0$   
**by** *simp*

**have**  $abs ?L = norm (complex-of-real ?L)$   
**unfolding** *norm-of-real* **by** *simp*  
**also have**  $\dots = norm (\sum T \leftarrow Ts. (g-inner ?f (?f \circ T)))$   
**unfolding**  $Ts$ -*def* **by** (*simp add*:*algebra-simps g-inner-def sum.distrib comp-def case-prod-beta*)  
**also have**  $\dots = norm (\sum T \leftarrow Ts. (g-inner (FT ?f) (FT (?f \circ T)))) / m^2$   
**by** (*subst parseval*) *simp*  
**also have**  $\dots = norm (g-inner (FT ?f) (\lambda x. (\sum T \leftarrow Ts. (FT (?f \circ T) x)))) / m^2$   
**unfolding**  $Ts$ -*def* **by** (*simp add*:*g-inner-simps case-prod-beta add-divide-distrib*)  
**also have**  $\dots = norm (g-inner (FT ?f) (\lambda x. (FT ?f (S_2 x) * (1 + \omega_F (fst x)) + FT ?f (S_1 x) * (1 + \omega_F (snd x))))) / m^2$   
**by** (*subst 0*) (*simp add*:*norm-divide 1*)  
**also have**  $\dots \leq (2.5 * sqrt 2) * (\sum v \in verts G. norm (FT f v)^2) / m^2$   
**by** (*intro divide-right-mono hoory-8-7*[**where**  $f=FT f$ ] *2 periodic-FT*) *auto*  
**also have**  $\dots = (2.5 * sqrt 2) * (\sum v \in verts G. cmod (f v)^2)$   
**by** (*subst* (2) *plancharel*) *simp*  
**also have**  $\dots = (2.5 * sqrt 2) * (g-inner f f)$   
**unfolding**  $g$ -*inner-def* *norm-of-real* **by** (*simp add*: *power2-eq-square*)  
**also have**  $\dots = ?R$   
**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**  $g\text{-inner } f (\lambda\cdot. 1) = 0$   
**shows**  $|\sum a \in \text{arcs } G. f (\text{head } G a) * f (\text{tail } G a)| \leq (5 * \text{sqrt } 2) * g\text{-norm } f^{\wedge} 2$  (**is**  $?L \leq ?R$ )  
**proof** –  
**define**  $g$  **where**  $g x = f (\text{fst } x \text{ mod } m, \text{snd } x \text{ mod } m)$  **for**  $x :: \text{int} \times \text{int}$   
**have**  $0:g x = f x$  **if**  $x \in \text{verts } G$  **for**  $x$   
**unfolding**  $g\text{-def}$  **using** *that*  
**by** (*auto simp add:mgg-graph-def mem-Times-iff*)  
  
**have**  $g\text{-mod-simps}[simp]: g (x, y \text{ mod } m) = g (x, y) g (x \text{ mod } m, y) = g (x, y)$  **for**  $x y :: \text{int}$   
**unfolding**  $g\text{-def}$  **by** *auto*  
  
**have** *periodic-g: periodic g*  
**unfolding** *periodic-def* **by** *simp*  
  
**have**  $g\text{-inner } g (\lambda\cdot. 1) = g\text{-inner } f (\lambda\cdot. 1)$   
**by** (*intro g-inner-cong 0*) *auto*  
**also have**  $\dots = 0$   
**using** *assms* **by** *simp*  
**finally have**  $1:g\text{-inner } g (\lambda\cdot. 1) = 0$  **by** *simp*  
  
**have**  $2:g\text{-norm } g = g\text{-norm } f$   
**by** (*intro g-norm-cong 0*) (*auto*)  
  
**have**  $?L = |\sum a \in \text{arcs } G. g (\text{head } G a) * g (\text{tail } G a)|$   
**using** *wellformed*  
**by** (*intro arg-cong[where f=abs] sum.cong arg-cong2[where f=(\*)] 0[symmetric]*) *auto*  
**also have**  $\dots = |\sum a \in \text{arcs-pos. } g(\text{head } G a) * g(\text{tail } G a) + \sum a \in \text{arcs-neg. } g(\text{head } G a) * g(\text{tail } G a)|$   
**unfolding** *arcs-sym arcs-pos-def arcs-neg-def*  
**by** (*intro arg-cong[where f=abs] sum.union-disjoint*) *auto*  
**also have**  $\dots = |2 * (\sum (v,l) \in \text{verts } G \times \{..<4\}. g v * g (\text{mgg-graph-step } m v (l, 1)))|$   
**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**  $\dots = 2 * |(\sum v \in \text{verts } G. (\sum l \in \{..<4\}. g v * g (\text{mgg-graph-step } m v (l, 1))))|$   
**by** (*subst sum.cartesian-product*) (*simp add:abs-mult*)  
**also have**  $\dots = 2 * |(\sum (x,y) \in \text{verts } G. (\sum l \leftarrow [0..<4]. g(x,y) * g (\text{mgg-graph-step } m (x,y) (l,1))))|$   
**by** (*subst interv-sum-list-conv-sum-set-nat*)  
*(auto simp add:atLeast0LessThan case-prod-beta simp del:mgg-graph-step.simps)*  
**also have**  $\dots = 2 * |\sum (x,y) \in \text{verts } G. g (x,y) * (g(x+2*y,y) + g(x+2*y+1,y) + g(x,y+2*x) + g(x,y+2*x+1))|$   
**by** (*simp add:case-prod-beta numeral-eq-Suc algebra-simps*)  
**also have**  $\dots \leq 2 * ((2.5 * \text{sqrt } 2) * g\text{-norm } g^{\wedge} 2)$   
**by** (*intro mult-left-mono hoory-8-3 1 periodic-g*) *auto*  
**also have**  $\dots \leq ?R$  **unfolding**  $2$  **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**definition**  $MGG\text{-bound} :: \text{real}$   
**where**  $MGG\text{-bound} = 5 * \text{sqrt } 2 / 8$

Main result: Theorem 8.2 in Hoory.

**lemma**  $mgg\text{-numerical-radius}: \Lambda_a \leq MGG\text{-bound}$

**proof** –  
**have**  $\Lambda_a \leq (5 * \text{sqrt } 2) / \text{real } d$   
**by** (*intro expander-intro mgg-numerical-radius-aux*) *auto*  
**also have**  $\dots = MGG\text{-bound}$   
**unfolding**  $MGG\text{-bound-def } d\text{-eq-8}$  **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*  $\Rightarrow$  *nat*  $\Rightarrow$  ('a list) *multiset*

**where**

*walks' G 0 = image-mset* ( $\lambda x. [x]$ ) (*mset-set* (*verts G*)) |

*walks' G (Suc n) =*

*concat-mset* { $\#\{\#w @ [z]. z \in \# \text{vertices-from } G \text{ (last } w)\#\}$ .  $w \in \# \text{walks' } G \text{ } n\#\}$ }

**definition** *walks G l = (case l of 0  $\Rightarrow$  { $\#\ []\#\}$  | Suc pl  $\Rightarrow$  *walks' G pl*)*

**lemma** *Union-image-mono*: ( $\bigwedge x. x \in A \Rightarrow f x \subseteq g x$ )  $\Longrightarrow \bigcup (f \text{ ` } A) \subseteq \bigcup (g \text{ ` } A)$

**by** *auto*

**context** *fin-digraph*

**begin**

**lemma** *count-walks'*:

**assumes** *set xs  $\subseteq$  verts G*

**assumes** *length xs = l+1*

**shows** *count (walks' G l) xs =* ( $\prod i \in \{..<l\}$ . *count (edges G) (xs ! i, xs ! (i+1))*)

**proof** –

**have** *a:xs  $\neq []$  using assms(2) by auto*

**have** *count (walks' G (length xs-1)) xs =* ( $\prod i < \text{length } xs - 1$ . *count (edges G) (xs ! i, xs ! (i + 1))*)

**using** *a assms(1)*

**proof** (*induction xs rule:rev-nonempty-induct*)

**case** (*single x*)

**hence** *x  $\in$  verts G by simp*

**hence** *count { $\#[x]$ .  $x \in \# \text{mset-set (verts G)\#\}$ } [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  $\subseteq$  verts G using snoc by simp*

**define** *l where l = length xs - 1*

**have**  $l$ -xs:  $\text{length } xs = l + 1$  **unfolding**  $l$ -def **using**  $snoc$  **by**  $simp$   
**have**  $\text{count } (\text{walks}' G (\text{length } (xs @ [x]) - 1)) (xs @ [x]) =$   
 $(\sum_{ys \in \# \text{walks}' G l. \text{count } \{\#ys @ [z]. z \in \# \text{vertices-from } G (\text{last } ys)\# \}} (xs @ [x]))$   
**by** ( $simp \text{ add:l-xs count-concat-mset image-mset.compositionality comp-def}$ )  
**also have**  $\dots = (\sum_{ys \in \# \text{walks}' G l.}$   
 $(\text{if } ys = xs \text{ then } \text{count } \{\#xs @ [z]. z \in \# \text{vertices-from } G (\text{last } xs)\# \} (xs @ [x]) \text{ else } 0))$   
**by** ( $intro \text{ arg-cong}[\text{where } f = \text{sum-mset}] \text{ image-mset-cong} (\text{auto } \text{intro!} : \text{count-image-mset-0-triv})$ )  
**also have**  $\dots = (\sum_{ys \in \# \text{walks}' G l. (\text{if } ys = xs \text{ then } \text{count } (\text{vertices-from } G (\text{last } xs)) x \text{ else } 0))$   
**by** ( $subst \text{ count-image-mset-inj, auto } \text{simp add:inj-def}$ )  
**also have**  $\dots = \text{count } (\text{walks}' G l) xs * \text{count } (\text{vertices-from } G (\text{last } xs)) x$   
**by** ( $subst \text{ sum-mset-delta, simp}$ )  
**also have**  $\dots = \text{count } (\text{walks}' G l) xs * \text{count } (\text{edges } G) (\text{last } xs, x)$   
**unfolding**  $\text{vertices-from-def count-mset-exp image-mset-filter-mset-swap}[\text{symmetric}]$   
 $\text{filter-filter-mset}$  **by** ( $simp \text{ add:prod-eq-iff}$ )  
**also have**  $\dots = \text{count } (\text{walks}' G l) xs * \text{count } (\text{edges } G) ((xs @ [x])!l, (xs @ [x])!(l+1))$   
**using**  $snoc(1)$  **unfolding**  $l$ -def  $\text{nth-append last-conv-nth}[OF \text{ snoc}(1)]$  **by**  $simp$   
**also have**  $\dots = (\prod_{i < l+1. \text{count } (\text{edges } G) ((xs @ [x])!i, (xs @ [x])!(i+1)))$   
**unfolding**  $l$ -def  $snoc(2)[OF \text{ set-xs}]$  **by** ( $simp \text{ add:nth-append}$ )  
**finally have**  $\text{count } (\text{walks}' G (\text{length } (xs @ [x]) - 1)) (xs @ [x]) =$   
 $(\prod_{i < \text{length } (xs @ [x]) - 1. \text{count } (\text{edges } G) ((xs @ [x])!i, (xs @ [x])!(i+1)))$   
**unfolding**  $l$ -def **using**  $snoc(1)$  **by**  $simp$   
**then show**  $?case$  **by**  $simp$   
**qed**  
**moreover have**  $l = \text{length } xs - 1$  **using**  $a \text{ assms}$  **by**  $simp$   
**ultimately show**  $?thesis$  **by**  $simp$   
**qed**

**lemma**  $\text{count-walks}$ :

**assumes**  $\text{set } xs \subseteq \text{verts } G$   
**assumes**  $\text{length } xs = l \ l > 0$   
**shows**  $\text{count } (\text{walks } G l) xs = (\prod_{i \in \{..<l-1\}. \text{count } (\text{edges } G) (xs ! i, xs ! (i+1))})$   
**using**  $assms$  **unfolding**  $\text{walks-def}$  **by** ( $\text{cases } l, \text{auto } \text{simp add:count-walks}'$ )

**lemma**  $\text{set-walks}'$ :

$\text{set-mset } (\text{walks}' G l) \subseteq \{xs. \text{set } xs \subseteq \text{verts } G \wedge \text{length } xs = (l+1)\}$

**proof** ( $\text{induction } l$ )

**case**  $0$

**then show**  $?case$  **by**  $\text{auto}$

**next**

**case** ( $\text{Suc } l$ )

**have**  $\text{set-mset } (\text{walks}' G (\text{Suc } l)) =$

$(\bigcup_{x \in \text{set-mset } (\text{walks}' G l). (\lambda z. x @ [z]) ' \text{set-mset } (\text{vertices-from } G (\text{last } x))}$

**by** ( $simp \text{ add:set-mset-concat-mset}$ )

**also have**  $\dots \subseteq (\bigcup_{x \in \{xs. \text{set } xs \subseteq \text{verts } G \wedge \text{length } xs = l + 1\}.}$

$(\lambda z. x @ [z]) ' \text{set-mset } (\text{vertices-from } G (\text{last } x))}$

**by** ( $intro \text{ Union-mono image-mono Suc}$ )

**also have**  $\dots \subseteq (\bigcup_{x \in \{xs. \text{set } xs \subseteq \text{verts } G \wedge \text{length } xs = l + 1\}. (\lambda z. x @ [z]) ' \text{verts } G}$

**by** ( $intro \text{ Union-image-mono image-mono set-mset-vertices-from}$ )

**also have**  $\dots \subseteq \{xs. \text{set } xs \subseteq \text{verts } G \wedge \text{length } xs = (\text{Suc } l + 1)\}$

**by** ( $intro \text{ subsetI} \text{ auto}$ )

**finally show**  $?case$  **by**  $simp$

**qed**

**lemma**  $\text{set-walks}$ :

$\text{set-mset } (\text{walks } G l) \subseteq \{xs. \text{set } xs \subseteq \text{verts } G \wedge \text{length } xs = l\}$

**unfolding**  $\text{walks-def}$  **using**  $\text{set-walks}'$  **by** ( $\text{cases } l, \text{auto}$ )

**lemma** *set-walks-2*:

**assumes**  $xs \in\# \text{walks}' G l$   
**shows**  $\text{set } xs \subseteq \text{verts } G \text{ } xs \neq []$

**proof** –

**have**  $a:xs \in \text{set-mset } (\text{walks}' G l)$   
**using** *assms* **by** *simp*  
**thus**  $\text{set } xs \subseteq \text{verts } G$   
**using** *set-walks'* **by** *auto*  
**have**  $\text{length } xs \neq 0$   
**using** *set-walks' a* **by** *fastforce*  
**thus**  $xs \neq []$  **by** *simp*

**qed**

**lemma** *set-walks-3*:

**assumes**  $xs \in\# \text{walks } G l$   
**shows**  $\text{set } xs \subseteq \text{verts } G \text{ } \text{length } xs = l$   
**using** *set-walks* *assms* **by** *auto*

**end**

**lemma** *measure-pmf-of-multiset*:

**assumes**  $A \neq \{\#\}$   
**shows**  $\text{measure } (\text{pmf-of-multiset } A) S = \text{real } (\text{size } (\text{filter-mset } (\lambda x. x \in S) A)) / \text{size } A$   
**(is**  $?L = ?R$ **)**

**proof** –

**have**  $\text{sum } (\text{count } A) (S \cap \text{set-mset } A) = \text{size } (\text{filter-mset } (\lambda x. x \in S \cap \text{set-mset } A) A)$   
**by** (*intro sum-count-2*) *simp*  
**also have**  $\dots = \text{size } (\text{filter-mset } (\lambda x. x \in S) A)$   
**by** (*intro arg-cong[where f=size] filter-mset-cong*) *auto*  
**finally have**  $a: \text{sum } (\text{count } A) (S \cap \text{set-mset } A) = \text{size } (\text{filter-mset } (\lambda x. x \in S) A)$   
**by** *simp*

**have**  $?L = \text{measure } (\text{pmf-of-multiset } A) (S \cap \text{set-mset } A)$   
**using** *assms* **by** (*intro measure-eq-AE AE-pmfI*) *auto*  
**also have**  $\dots = \text{sum } (\text{pmf } (\text{pmf-of-multiset } A)) (S \cap \text{set-mset } A)$   
**by** (*intro measure-measure-pmf-finite*) *simp*  
**also have**  $\dots = (\sum x \in S \cap \text{set-mset } A. \text{count } A x / \text{size } A)$   
**using** *assms* **by** (*intro sum.cong, auto*)  
**also have**  $\dots = (\sum x \in S \cap \text{set-mset } A. \text{count } A x) / \text{size } A$   
**by** (*simp add:sum-divide-distrib*)  
**also have**  $\dots = ?R$   
**using** *a* **by** *simp*  
**finally show** *?thesis*  
**by** *simp*

**qed**

**lemma** *pmf-of-multiset-image-mset*:

**assumes**  $A \neq \{\#\}$   
**shows**  $\text{pmf-of-multiset } (\text{image-mset } f A) = \text{map-pmf } f (\text{pmf-of-multiset } A)$   
**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'*:

$\text{size } (\text{walks}' G l) = \text{card } (\text{verts } G) * d \wedge l$

**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)) = (∑ 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 ... = (∑ 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 Λa
proof -
  have norm (A *v v) ≤ Λa * norm v if v · 1 = (0::real) for v::real^n
  unfolding Λe-eq-Λ

```

by (intro  $\gamma_a$ -real-bound that)  
 thus ?thesis  
 unfolding spec-bound-def using  $\Lambda$ -ge-0 by auto  
 qed

A spectral expansion rule that does not require orthogonality of the vector for the stationary distribution:

lemma expansionD3:

$|g\text{-inner } f (g\text{-step } f)| \leq \Lambda_a * g\text{-norm } f^2 + (1 - \Lambda_a) * g\text{-inner } f (\lambda \cdot 1)^2 / n$  (is ?L ≤ ?R)

proof –

define v where  $v = (\chi \ i. f (enum\text{-verts } i))$   
 define v1 ::  $\text{real}^n$  where  $v1 = ((v \cdot 1) / n) *_{\mathbb{R}} 1$   
 define v2 ::  $\text{real}^n$  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  $v2 \cdot 1 = v \cdot 1 - v1 \cdot 1$   
 unfolding v2-def by (simp add: algebra-simps)  
 also have  $\dots = v \cdot 1 - v \cdot 1 * \text{real } \text{CARD}(n) / \text{real } n$   
 unfolding v1-def by (simp add: inner-1-1)  
 also have  $\dots = 0$   
 using verts-non-empty unfolding card n-def by simp  
 finally have 4:  $v2 \cdot 1 = 0$  by simp  
 hence 2:  $v1 \cdot v2 = 0$   
 unfolding v1-def by (simp add: inner-commute)

define f2 where  $f2 \ i = v2 \ \$ (enum\text{-verts}\text{-inv } i)$  for  $i$   
 have f2-def:  $v2 = (\chi \ i. f2 (enum\text{-verts } i))$   
 unfolding f2-def Rep-inverse by simp

have 6:  $g\text{-inner } f2 (\lambda \cdot 1) = 0$   
 unfolding g-inner-conv f2-def[symmetric] one-vec-def[symmetric] 4 by simp

have  $|v2 \cdot (A * v \ v2)| = |g\text{-inner } f2 (g\text{-step } f2)|$   
 unfolding f2-def g-inner-conv g-step-conv by simp  
 also have  $\dots \leq \Lambda_a * (g\text{-norm } f2)^2$   
 by (intro expansionD1 6)  
 also have  $\dots = \Lambda_a * (\text{norm } v2)^2$   
 unfolding g-norm-conv f2-def by simp  
 finally have 5:  $|v2 \cdot (A * v \ v2)| \leq \Lambda_a * (\text{norm } v2)^2$  by simp

have 3:  $\text{norm } (1 :: \text{real}^n)^2 = n$   
 unfolding power2-norm-eq-inner inner-1-1 card n-def by presburger

have ?L =  $|v \cdot (A * v \ v)|$   
 unfolding g-inner-conv g-step-conv v-def by simp  
 also have  $\dots = |v1 \cdot (A * v \ v1) + v2 \cdot (A * v \ v1) + v1 \cdot (A * v \ v2) + v2 \cdot (A * v \ v2)|$   
 unfolding v-eq by (simp add: algebra-simps)  
 also have  $\dots = |v1 \cdot v1 + v2 \cdot v1 + v1 \cdot v2 + v2 \cdot (A * v \ v2)|$   
 unfolding dot-lmul-matrix[where  $x=v1, \text{symmetric}$ ] 0 1 by simp  
 also have  $\dots = |v1 \cdot v1 + v2 \cdot (A * v \ v2)|$

```

    using 2 by (simp add:inner-commute)
  also have ... ≤ |norm v1^2| + |v2 · (A * v2)|
    unfolding power2-norm-eq-inner by (intro abs-triangle-ineq)
  also have ... ≤ norm v1^2 + Λa * norm v2^2
    by (intro add-mono 5) auto
  also have ... = Λa * (norm v1^2 + norm v2^2) + (1 - Λa) * norm v1^2
    by (simp add:algebra-simps)
  also have ... = Λa * norm v^2 + (1 - Λa) * norm v1^2
    unfolding v-eq pythagoras[OF 2] by simp
  also have ... = Λa * norm v^2 + ((1 - Λa)) * ((v · 1)^2 * n) / n^2
    unfolding v1-def by (simp add:power-divide power-mult-distrib 3)
  also have ... = Λa * norm v^2 + ((1 - Λa) / n) * (v · 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*  $S = \text{diag} (\text{ind-vec} (\text{enum-verts} - 'S))$

**lemma** *walk-distr*:

```

  measure (pmf-of-multiset (walks G l)) {ω. (∀ i < l. ω ! i ∈ S i)} =
  foldl (λx M. M * v x) stat (intersperse A (map (λi. ind-mat (S i)) [0..<l])) . 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 ⊆ verts G ∧ length w = l}
  let ?V = {(w::'n list). length w = l}

```

```

  have a: set-mset (walks G l) ⊆ ?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 (λi. ind-mat (S i)) [0..<l]
  have xs ≠ [] 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**  $\text{card } ?W = \text{card } (\text{verts } G) \frown$   
**by** *(intro card-lists-length-eq) simp*  
**also have**  $\dots = \text{card } \{w. \text{set } w \subseteq (\text{UNIV} :: 'n \text{ set}) \wedge \text{length } w = l\}$   
**unfolding**  $\text{card}[\text{symmetric}]$  **by** *(intro card-lists-length-eq[symmetric]) simp*  
**also have**  $\dots = \text{card } ?V$   
**by** *(intro arg-cong[where f=card]) auto*  
**also have**  $\dots = \text{card } (\text{map } \text{enum-verts } ' ?V)$   
**by** *(intro card-image[symmetric] inj-map-enum-verts)*  
**finally have**  $\text{card } ?W = \text{card } (\text{map } \text{enum-verts } ' ?V)$   
**by** *simp*  
**hence**  $\text{map } \text{enum-verts } ' ?V = ?W$   
**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**  $?L = \text{size } \{\# w \in \# \text{walks } G \text{ l. } \forall i < l. w ! i \in S i \# \} / (?n * ?d^{l-1})$   
**using** *True unfolding size-walks measure-pmf-of-multiset[OF walks-nonempty] by simp*  
**also have**  $\dots = (\sum w \in ?W. \text{real } (\text{count } (\text{walks } G \text{ l}) w) * \text{of-bool } (\forall i < l. w ! i \in S i)) / (?n * ?d^{l-1})$   
**unfolding** *size-filter-mset-conv sum-mset-conv-2[OF a b] by simp*  
**also have**  $\dots = (\sum w \in ?W. (\prod i < l-1. \text{real } (\text{count } (\text{edges } G) (w ! i, w !(i+1)))) * (\prod i < l. \text{of-bool } (w ! i \in S i))) / (?n * ?d^{l-1})$   
**using** *True by (intro sum.cong arg-cong2[where f=()]) (auto simp add: count-walks)*  
**also have**  $\dots = (\sum w \in ?W. (\prod i < l-1. \text{real } (\text{count } (\text{edges } G) (w ! i, w !(i+1))) / ?d) * (\prod i < l. \text{of-bool } (w ! i \in S i))) / ?n$   
**using** *True unfolding prod-dividef by (simp add: sum-divide-distrib algebra-simps)*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < l-1. \text{count } (\text{edges } G) (\text{map } \text{enum-verts } w ! i, \text{map } \text{enum-verts } w !(i+1)) / ?d) * (\prod i < l. \text{of-bool } (\text{map } \text{enum-verts } w ! i \in S i))) / ?n$   
**by** *(intro sum.reindex-bij-betw[symmetric] arg-cong2[where f=()] refl bij-map-enum-verts)*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. A \$ w !(i+1) \$ w ! i) * (\prod i < \text{Suc } lp. \text{of-bool}(\text{enum-verts } (w ! i) \in S i))) / ?n$   
**unfolding** *A-def lp-def using True by simp*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. A \$ w !(i+1) \$ w ! i) * (\prod i \in \text{insert } 0 (\text{Suc } ' \{.. < lp\}). \text{of-bool}(\text{enum-verts } (w ! i) \in S i))) / ?n$   
**using** *lessThan-Suc-eq-insert-0*  
**by** *(intro sum.cong arg-cong2[where f=()] arg-cong2[where f=(\*)] prod.cong) auto*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. \text{of-bool}(\text{enum-verts } (w !(i+1)) \in S (i+1)) * A \$ w !(i+1) \$ w ! i) * \text{of-bool}(\text{enum-verts } (w ! 0) \in S 0)) / ?n$   
**by** *(simp add: prod.reindex algebra-simps prod.distrib)*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. (\text{ind-mat } (S (i+1)) ** A) \$ w !(i+1) \$ w ! i) * \text{of-bool}(\text{enum-verts } (w ! 0) \in S 0)) / ?n$   
**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**  $\dots = (\sum w \in ?V. (\prod i < lp. (xs !(i+1) ** A) \$ w !(i+1) \$ w ! i) * \text{of-bool}(\text{enum-verts } (w ! 0) \in S 0)) / ?n$   
**unfolding** *xs-def lp-def True*  
**by** *(intro sum.cong arg-cong2[where f=()] arg-cong2[where f=(\*)] prod.cong refl) auto*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. (xt ! i ** A) \$ w !(i+1) \$ w ! i) * \text{of-bool}(\text{enum-verts } (w ! 0) \in S 0)) / ?n$   
**unfolding** *xh-xt[symmetric] by auto*  
**also have**  $\dots = (\sum w \in ?V. (\prod i < lp. (xt ! i ** A) \$ w !(i+1) \$ w ! i) * (\text{ind-mat } (S 0) * v \text{ stat}) \$ w ! 0)$   
**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**  $\dots = (\sum_{w \in ?V}. (\prod_{i < lp}. (xt ! i ** A) \$ w!(i+1) \$ w!i) * (xh *v stat) \$ w ! 0)$   
**unfolding** *xh-eq* **by** *simp*  
**also have**  $\dots = foldl (\lambda x M. M *v x) (xh *v stat) (map (\lambda x. x ** A) xt) \cdot 1$   
**using** *True unfolding foldl-matrix-mult-expand-2* **by** (*simp add:len-xt lp-def*)  
**also have**  $\dots = foldl (\lambda x M. M *v (A *v x)) (xh *v stat) xt \cdot 1$   
**by** (*simp add: matrix-vector-mul-assoc foldl-map*)  
**also have**  $\dots = foldl (\lambda x M. M *v x) stat (intersperse A (xh\#xt)) \cdot 1$   
**by** (*subst foldl-intersperse-2, simp*)  
**also have**  $\dots = ?R$  **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**  $S \subseteq \text{verts } G$   
**assumes**  $I \subseteq \{..<l\}$   
**defines**  $\mu \equiv \text{real } (card S) / card (\text{verts } G)$   
**shows**  $\text{measure } (pmf\text{-of-multiset } (walks G l)) \{w. set (nth\ s\ w\ I) \subseteq S\} \leq (\mu + \Lambda_a * (1 - \mu)) \wedge^{card I}$   
**(is**  $?L \leq ?R$ **)**

**proof** –

**define**  $T$  **where**  $T = (\lambda i. \text{if } i \in I \text{ then } S \text{ else } UNIV)$

**have**  $0$ : *ind-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**  $S \subseteq \text{range enum-verts}$   
**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*:  $\mu = \text{real } (card (\text{enum-verts} - 'S)) / CARD ('n)$

**unfolding**  $\mu$ -*def card* **by** (*subst card-vimage-inj*) *auto*

**have**  $?L = \text{measure } (pmf\text{-of-multiset } (walks G l)) \{w. \forall i < l. w ! i \in T i\}$

**using** *walks-nonempty set-walks-3* **unfolding** *T-def set-nths*

**by** (*intro measure-eq-AE AE-pmfI*) *auto*

**also have**  $\dots = foldl (\lambda x M. M *v x) stat$

(*intersperse A (map (\lambda i. (if i \in I then ind-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**  $\dots \leq ?R$

**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**  $i < l$   $x \in \text{verts } G$

**shows**  $\text{measure } (pmf\text{-of-multiset } (walks G l)) \{w. w ! i = x\} = 1 / \text{real } (card (\text{verts } G))$

**(is**  $?L = ?R$ **)**

**proof** –

**obtain**  $xi$  **where**  $xi$ -*def*:  $\text{enum-verts } xi = x$

**using** *assms(2) bij-betw-imp-surj-on[OF enum-verts]* **by** *force*

```

define  $T$  where  $T = (\lambda j. \text{if } j = i \text{ then } \{x\} \text{ else } UNIV)$ 

have  $\text{diag } (ind\text{-vec } UNIV) = \text{mat } 1$ 
  unfolding  $\text{diag-def } ind\text{-vec-def } Finite\text{-Cartesian-Product.mat-def}$  by  $\text{vector}$ 
moreover have  $\text{enum-verts} - ' \{x\} = \{xi\}$ 
  using  $\text{bij-betw-imp-inj-on}[OF \text{ enum-verts}]$ 
  unfolding  $\text{vimage-def } xi\text{-def}[symmetric]$  by  $(\text{auto simp add:inj-on-def})$ 
ultimately have  $0: ind\text{-mat } (T j) = (\text{if } j = i \text{ then } \text{diag } (ind\text{-vec } \{xi\}) \text{ else } \text{mat } 1)$  for  $j$ 
  unfolding  $T\text{-def } ind\text{-mat-def}$  by  $(\text{cases } j = i, \text{auto})$ 

have  $?L = \text{measure } (pmf\text{-of-multiset } (\text{walks } G l)) \{w. \forall j < l. w ! j \in T j\}$ 
  unfolding  $T\text{-def}$  using  $\text{assms}(1)$  by  $\text{simp}$ 
also have  $\dots = \text{foldl } (\lambda x M. M * v x) \text{stat } (\text{intersperse } A (\text{map } (\lambda j. ind\text{-mat } (T j)) [0..<l])) \cdot 1$ 
  unfolding  $\text{walk-distr}$  by  $\text{simp}$ 
also have  $\dots = 1 / \text{CARD}('n)$ 
  unfolding  $0 \text{ uniform-property-alg}[OF \text{ assms}(1) \text{ markov}]$  by  $\text{simp}$ 
also have  $\dots = ?R$ 
  unfolding  $\text{card}$  by  $\text{simp}$ 
finally show  $?thesis$  by  $\text{simp}$ 
qed

end

context  $\text{regular-graph}$ 
begin

lemmas  $\text{expansionD3} =$ 
   $\text{regular-graph-tts.expansionD3}[OF \text{ eg-tts-1},$ 
   $\text{internalize-sort } 'n :: \text{finite}, OF - \text{regular-graph-axioms},$ 
   $\text{unfolded } \text{remove-finite-premise}, \text{cancel-type-definition}, OF \text{ verts-non-empty}]$ 

lemmas  $\text{g-step-remains-orth} =$ 
   $\text{regular-graph-tts.g-step-remains-orth}[OF \text{ eg-tts-1},$ 
   $\text{internalize-sort } 'n :: \text{finite}, OF - \text{regular-graph-axioms},$ 
   $\text{unfolded } \text{remove-finite-premise}, \text{cancel-type-definition}, OF \text{ verts-non-empty}]$ 

lemmas  $\text{hitting-property} =$ 
   $\text{regular-graph-tts.hitting-property}[OF \text{ eg-tts-1},$ 
   $\text{internalize-sort } 'n :: \text{finite}, OF - \text{regular-graph-axioms},$ 
   $\text{unfolded } \text{remove-finite-premise}, \text{cancel-type-definition}, OF \text{ verts-non-empty}]$ 

lemmas  $\text{uniform-property-2} =$ 
   $\text{regular-graph-tts.uniform-property}[OF \text{ eg-tts-1},$ 
   $\text{internalize-sort } 'n :: \text{finite}, OF - \text{regular-graph-axioms},$ 
   $\text{unfolded } \text{remove-finite-premise}, \text{cancel-type-definition}, OF \text{ verts-non-empty}]$ 

theorem  $\text{uniform-property}$ :
  assumes  $i < l$ 
  shows  $\text{map-pmf } (\lambda w. w ! i) (pmf\text{-of-multiset } (\text{walks } G l)) = pmf\text{-of-set } (\text{verts } G) \text{ (is } ?L = ?R)$ 
proof  $(\text{rule } pmf\text{-eqI})$ 
  fix  $x :: 'a$ 
  have  $a:\text{measure } (pmf\text{-of-multiset } (\text{walks } G l)) \{w. w ! i = x\} = 0 \text{ (is } ?L1 = ?R1)$ 
  if  $x \notin \text{verts } G$ 
proof  $-$ 
  have  $?L1 \leq \text{measure } (pmf\text{-of-multiset } (\text{walks } G l)) \{w. \text{set } w \subseteq \text{verts } G \wedge x \in \text{set } w\}$ 
  using  $\text{walks-nonempty set-walks-3 assms}(1)$ 
  by  $(\text{intro } pmf\text{-mono}) \text{auto}$ 
  also have  $\dots \leq \text{measure } (pmf\text{-of-multiset } (\text{walks } G l)) \{\}$ 

```

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<sub>a</sub> \* (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<sub>a</sub> \* (1-μ))) (is ?L ≤ ?R)

proof –

let ?δ = (∑ i < l. μ + Λ<sub>a</sub> \* (1-μ)) / l

have a: measure (pmf-of-multiset (walks G l)) {w. ∀ i ∈ T. w ! i ∈ S} ≤ (μ + Λ<sub>a</sub> \* (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  $\Rightarrow$  'a  $\Rightarrow$  'b list  $\Rightarrow$  bool

where

*is-arc-walk* G - [] = True |

*is-arc-walk* G y (x#xs) = (*is-arc-walk* G (head G x) xs  $\wedge$  tail G x = y  $\wedge$  x  $\in$  arcs G)

definition *arc-walk-head* :: ('a, 'b) pre-digraph  $\Rightarrow$  ('a  $\times$  'b list)  $\Rightarrow$  '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])  $\longleftrightarrow$  *is-arc-walk* G y xs  $\wedge$  x  $\in$  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  $\subseteq$  arcs G

using assms by (induction w arbitrary: u, auto)

lemma (in wf-digraph) *awalk-is-arc-walk*:

assumes u  $\in$  verts G

shows *is-arc-walk* G u w  $\longleftrightarrow$  awalk u w (awlast u w)

using assms unfolding *awalk-def* by (induction w arbitrary: u, auto)

definition *arc-walks* :: ('a, 'b) pre-digraph  $\Rightarrow$  nat  $\Rightarrow$  ('a  $\times$  'b list) set

where

*arc-walks* G l = {(u,w). u  $\in$  verts G  $\wedge$  *is-arc-walk* G u w  $\wedge$  length w = l}

lemma *arc-walks-len*:

assumes x  $\in$  *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  $\in$  *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**  $w \in \text{arc-walks } G \ l$   
**shows**  $\text{awlast } (fst \ w) \ (snd \ w) = \text{arc-walk-head } G \ w$   
**unfolding** *awalk-verts-conv arc-walk-head-def* **by** *simp*

**lemma** (in *wf-digraph*) *arc-walk-head-wellformed*:  
**assumes**  $w \in \text{arc-walks } G \ l$   
**shows**  $\text{arc-walk-head } G \ w \in \text{verts } G$   
**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**  $0:\text{is-arc-walk } G \ (fst \ w) \ (snd \ w)$  **using** *assms unfolding arc-walks-def* **by** *auto*  
**have**  $last \ (snd \ w) \in \text{set } (snd \ w)$   
**using** *False last-in-set* **by** *auto*  
**also have**  $\dots \subseteq \text{arcs } G$   
**by** (*intro is-arc-walk-set[OF 0]*)  
**finally have**  $last \ (snd \ w) \in \text{arcs } G$  **by** *simp*  
**thus** *?thesis* **unfolding** *arc-walk-head-def* **using** *False* **by** *simp*

**qed**

**lemma** (in *wf-digraph*) *arc-walk-tail-wellformed*:  
**assumes**  $w \in \text{arc-walks } G \ l$   
**shows**  $fst \ w \in \text{verts } G$   
**using** *assms unfolding arc-walks-def* **by** *auto*

**lemma** (in *fin-digraph*) *arc-walks-fin*:  
*finite* ( $\text{arc-walks } G \ l$ )  
**proof** –  
**have**  $0:\text{finite } (\text{verts } G \times \{w. \text{set } w \subseteq \text{arcs } G \wedge \text{length } w = l\})$   
**by** (*intro finite-cartesian-product finite-lists-length-eq*) *auto*  
**show** *finite* ( $\text{arc-walks } G \ l$ )  
**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**  $w \in \text{arc-walks } G \ l$   
**shows**  $\text{awalk-verts } (fst \ w) \ (snd \ w) = \text{fst } w \# \text{map } (\text{head } G) \ (snd \ w)$  (**is**  $?L = ?R$ )  
**proof** –  
**obtain**  $u \ v$  **where**  $w\text{-def}: w = (u,v)$  **by** *fastforce*

**have**  $\text{awalk } u \ v \ (\text{awlast } u \ v)$   
**using** *assms unfolding w-def arc-walks-def*  
**by** (*intro iffD1[OF awalk-is-arc-walk]*) *auto*  
**hence**  $\text{cas}: \text{cas } u \ v \ (\text{awlast } u \ v)$   
**unfolding** *awalk-def* **by** *simp*

**have**  $0:\text{tail } G \ (\text{hd } v) = u$  **if**  $v \neq []$   
**using** *cas that* **by** (*cases v*) *auto*

**have**  $?L = \text{awalk-verts } u \ v$   
**unfolding** *w-def* **by** *simp*  
**also have**  $\dots = (\text{if } v = [] \ \text{then } [u] \ \text{else } \text{tail } G \ (\text{hd } v) \ \# \ \text{map } (\text{head } G) \ v)$   
**by** (*intro awalk-verts-conv'[OF cas]*)  
**also have**  $\dots = u \# \ \text{map } (\text{head } G) \ v$

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 =  $\lambda x. \text{fst } x \# \text{map } (\text{head } G) (\text{snd } x)$

have walks' G 0 =  $\{\#[x]. x \in \# \text{mset-set } (\text{verts } G) \# \}$

by simp

also have ... = image-mset ?g (image-mset ( $\lambda x. (x, [])$ ) (mset-set (verts G)))

unfolding image-mset.compositionality by (simp add:comp-def)

also have ... = image-mset ?g (mset-set (( $\lambda x. (x, [])$ ) 'verts G))

by (intro arg-cong2[where f=image-mset] image-mset-mset-set inj-onI) auto

also have ... = image-mset ?g (mset-set ( $\{(u, w). u \in \text{verts } G \wedge w = []\}$ ))

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 =  $\lambda(u, w) a. (u, w @ [a])$

let ?g =  $\lambda x. \text{fst } x \# \text{map } (\text{head } G) (\text{snd } x)$

have arc-walks G (l+1) = case-prod ?f '  $\{(x, y). ?f x y \in \text{arc-walks } G (l+1)\}$

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 '  $\{(x, y). x \in \text{arc-walks } G l \wedge y \in \text{out-arcs } G (\text{arc-walk-head } G x)\}$

unfolding arc-walks-def using is-arc-walk-snoc[where G=G]

by (intro-cong [ $\sigma_2$  image]) auto

also have ... =  $(\bigcup w \in \text{arc-walks } G l. ?f w \text{ 'out-arcs } G (\text{arc-walk-head } G w))$

by (auto simp add:image-iff)

finally have 0:arc-walks G (l+1) =  $(\bigcup w \in \text{arc-walks } G l. ?f w \text{ 'out-arcs } G (\text{arc-walk-head } G w))$

by simp

have mset-set (arc-walks G (l+1)) = concat-mset (image-mset (mset-set  $\circ$

$(\lambda w. ?f w \text{ 'out-arcs } G (\text{arc-walk-head } G w))$ ) (mset-set (arc-walks G l)))

unfolding 0 by (intro concat-disjoint-union-mset arc-walks-fin finite-imageI) auto

also have ... = concat-mset  $\{\# \text{mset-set } (?f x \text{ 'out-arcs } G (\text{arc-walk-head } G x)).$

$x \in \# \text{mset-set}(\text{arc-walks } G l) \# \}$

by (simp add:comp-def case-prod-beta)

also have ... = concat-mset  $\{\# \{\# ?f x y. y \in \# \text{mset-set } (\text{out-arcs } G (\text{arc-walk-head } G x)) \# \}.$

$x \in \# \text{mset-set } (\text{arc-walks } G l) \# \}$

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

$\{\# \{\# ?f x y. y \in \# \text{mset-set } (\text{out-arcs } G (\text{arc-walk-head } G x)) \# \}. x \in \# \text{mset-set } (\text{arc-walks } G l) \# \}$

G l)#}

by simp

have walks' G (l+1) = concat-mset  $\{\# \{\# w @ [z]. z \in \# \text{vertices-from } G (\text{last } w) \# \}. w \in \# \text{walks}'$

$G \ l\#\}$   
**by** *simp*  
**also have** ... = *concat-mset*  $\{\#\}$   
 $\{\#\text{awalk-verts } (fst\ x)\ (snd\ x)\ @\ [z].\ z \in\ \#\ \text{vertices-from } G\ (awlast\ (fst\ x)\ (snd\ x))\#\}$ .  
 $x \in\ \#\ \text{mset-set } (arc\ \text{walks } G\ l)\#\}$   
**unfolding** *Suc* **by** (*simp add:image-mset.compositionality comp-def case-prod-beta*)  
**also have** ... = *concat-mset*  $\{\#\}$   
 $\{\#\ ?g\ x\ @\ [z].\ z \in\ \#\ \text{vertices-from } G\ (awlast\ (fst\ x)\ (snd\ x))\#\}$ .  
 $x \in\ \#\ \text{mset-set } (arc\ \text{walks } G\ l)\#\}$   
**using** *arc-walks-fin*  
**by** (*intro-cong*  $[\sigma_1\ \text{concat-mset}]$  *more:image-mset-cong*) (*auto simp: awalk-verts-unfold*)  
**also have** ... = *concat-mset*  $\{\#\ \{\#\ ?g\ x\ @\ [z].\ z \in\ \#\ \text{vertices-from } G\ (arc\ \text{walk-head } G\ x)\#\}\}$ .  
 $x \in\ \#\ \text{mset-set } (arc\ \text{walks } G\ l)\#\}$   
**using** *arc-walks-fin awlast-of-arc-walk*  
**by** (*intro-cong*  $[\sigma_1\ \text{concat-mset}, \sigma_2\ \text{image-mset}]$  *more: image-mset-cong*) *auto*  
**also have** ... = (*concat-mset*  $\{\#\ \{\#\ ?g\ (fst\ x,\ snd\ x@[y]).$   
 $y \in\ \#\ \text{mset-set } (out\ \text{arcs } G\ (arc\ \text{walk-head } G\ x))\#\}$ .  $x \in\ \#\ \text{mset-set } (arc\ \text{walks } G\ l)\#\}$ )  
**unfolding** *verts-from-alt* **by** (*simp add:image-mset.compositionality comp-def*)  
**also have** ... = *image-mset*  $\ ?g\ (concat\ \text{mset } \{\#\ \{\#\ ?f\ x\ y.$   
 $y \in\ \#\ \text{mset-set } (out\ \text{arcs } G\ (arc\ \text{walk-head } G\ x))\#\}$ .  $x \in\ \#\ \text{mset-set } (arc\ \text{walks } G\ l)\#\}$ )  
**unfolding** *image-concat-mset*  
**by** (*auto simp add:comp-def case-prod-beta image-mset.compositionality*)  
**also have** ... = *image-mset*  $\ ?g\ (\text{mset-set } (arc\ \text{walks } G\ (l+1)))$   
**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*:  
 $walks\ G\ (l+1) = \text{image-mset } (\text{case-prod } awalk\ \text{verts}) (\text{mset-set } (arc\ \text{walks } G\ l))$   
**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**  $1: a \neq []$  **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* ::  $('a,\ 'b)\ \text{pre-digraph} \Rightarrow \text{nat} \Rightarrow ('a,\ ('a \times 'b\ \text{list}))\ \text{pre-digraph}$   
**where** *graph-power*  $G\ l =$   
 $(\ | \ \text{verts} = \text{verts } G,\ \text{arcs} = \text{arc-walks } G\ l,\ \text{tail} = \text{fst},\ \text{head} = \text{arc-walk-head } G \ |)$

**lemma** (*in wf-digraph*) *graph-power-wf*:  
 $wf\ \text{digraph } (\text{graph-power } G\ l)$

**proof** –

**have**  $tail\ (\text{graph-power } G\ l)\ a \in \text{verts } (\text{graph-power } G\ l)$   
 $head\ (\text{graph-power } G\ l)\ a \in \text{verts } (\text{graph-power } G\ l)$   
**if**  $a \in \text{arcs } (\text{graph-power } G\ l)$  **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**  $S \equiv \{x. \text{length } x = l + 1 \wedge \text{set } x \subseteq \text{verts } G \wedge \text{hd } x = v \wedge \text{last } x = w\}$

**shows**  $\text{count} (\text{edges} (\text{graph-power } G \ l)) (v, w) = (\sum x \in S. (\prod i < l. \text{count} (\text{edges } G) (x!i, x!(i+1))))$   
*(is ?L = ?R)*

**proof** –

**interpret** *H:fin-digraph graph-power G l*  
**using** *graph-power-fin* **by** *auto*

**have**  $0 : \text{finite} \{x. \text{set } x \subseteq \text{verts } G \wedge \text{length } x = l + 1\}$   
**by** *(intro finite-lists-length-eq) auto*

**have** *fin-S: finite S*

**unfolding** *S-def* **by** *(intro finite-subset[OF - 0]) auto*

**have**  $?L = \text{size} \{\#x \in \# \text{mset-set} (\text{arc-walks } G \ l). \text{fst } x = v \wedge \text{arc-walk-head } G \ x = w\# \}$

**unfolding** *graph-power-def edges-def arc-to-ends-def*  
**by** *(simp add:count-mset-exp image-mset-filter-mset-swap[symmetric])*

**also have**  $\dots = \text{size}$

$\{\#x \in \# \text{mset-set} (\text{arc-walks } G \ l). \text{awhd} (\text{fst } x) (\text{snd } x) = v \wedge \text{awlast} (\text{fst } x) (\text{snd } x) = w\# \}$

**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**  $\dots = \text{size} \{\#x \in \# \text{walks } G \ (l+1). \text{hd } x = v \wedge \text{last } x = w\# \}$

**unfolding** *arc-walks-map-walks*

**by** *(simp add:image-mset-filter-mset-swap[symmetric] case-prod-beta)*

**also have**  $\dots = \text{size} \{\#x \in \# \text{walks } G \ (l+1). x \in S\# \}$

**unfolding** *S-def* **using** *set-walks-3*

**by** *(intro arg-cong[where f=size] filter-mset-cong refl) auto*

**also have**  $\dots = \text{sum} (\text{count} (\text{walks } G \ (l+1))) \ S$

**by** *(intro sum-count-2[symmetric] fin-S)*

**also have**  $\dots = (\sum x \in S. (\prod i < l + 1 - 1. \text{count} (\text{edges } G) (x!i, x!(i+1))))$

**unfolding** *S-def*

**by** *(intro sum.cong refl count-walks) auto*

**also have**  $\dots = ?R$

**by** *simp*

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

**qed**

**lemma** (in *fin-digraph*) *graph-power-sym-aux*:  
**assumes** *symmetric-multi-graph*  $G$   
**assumes**  $v \in \text{verts } (\text{graph-power } G \ l)$   $w \in \text{verts } (\text{graph-power } G \ l)$   
**shows**  $\text{card } (\text{arcs-betw } (\text{graph-power } G \ l) \ v \ w) = \text{card } (\text{arcs-betw } (\text{graph-power } G \ l) \ w \ v)$   
(is ?L = ?R)  
**proof** –  
**interpret**  $H:\text{fin-digraph } \text{graph-power } G \ l$   
**using** *graph-power-fin* **by** *auto*  
  
**define**  $S$  **where**  $S \ v \ w = \{x. \text{length } x=l+1 \wedge \text{set } x \subseteq \text{verts } G \wedge \text{hd } x = v \wedge \text{last } x = w\}$  **for**  $v \ w$   
  
**have**  $0: \text{bij-betw } \text{rev } (S \ w \ v) \ (S \ v \ w)$   
**unfolding**  $S\text{-def}$  **by** (intro *bij-betwI*[**where**  $g=\text{rev}$ ]) (*auto simp add:hd-rev last-rev*)  
  
**have**  $1: \text{bij-betw } ((-) \ (l - 1)) \ \{..<l\} \ \{..<l\}$   
**by** (intro *bij-betwI*[**where**  $g=\lambda x. (l-1-x)$ ]) *auto*  
  
**have**  $?L = \text{count } (\text{edges } (\text{graph-power } G \ l)) \ (v, w)$   
**unfolding**  $H.\text{count-edges}$  **by** *simp*  
**also have**  $\dots = (\sum x \in S \ v \ w. (\prod i < l. \text{count } (\text{edges } G) \ (x!i, x!(i+1))))$   
**unfolding**  $S\text{-def}$  *graph-power-count-edges* **by** *simp*  
**also have**  $\dots = (\sum x \in S \ w \ v. (\prod i < l. \text{count } (\text{edges } G) \ (\text{rev } x!i, \text{rev } x!(i+1))))$   
**by** (intro *sum.reindex-bij-betw*[*symmetric*]  $0$ )  
**also have**  $\dots = (\sum x \in S \ w \ v. (\prod i < l. \text{count } (\text{edges } G) \ (x!((l-1-i)+1), x!(l-1-i))))$   
**unfolding**  $S\text{-def}$  **by** (intro *sum.cong refl prod.cong*) (*simp-all add: rev-nth Suc-diff-Suc*)  
**also have**  $\dots = (\sum x \in S \ w \ v. (\prod i < l. \text{count } (\text{edges } G) \ (x!(i+1), x!i)))$   
**by** (intro *sum.cong prod.reindex-bij-betw refl 1*)  
**also have**  $\dots = (\sum x \in S \ w \ v. (\prod i < l. \text{count } (\text{edges } G) \ (x!i, x!(i+1))))$   
**by** (intro *sum.cong prod.cong count-edges-sym*[*OF assms(1)*] *refl*)  
**also have**  $\dots = \text{count } (\text{edges } (\text{graph-power } G \ l)) \ (w, v)$   
**unfolding**  $S\text{-def}$  *graph-power-count-edges* **by** *simp*  
**also have**  $\dots = ?R$   
**unfolding**  $H.\text{count-edges}$  **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** (in *fin-digraph*) *graph-power-sym*:  
**assumes** *symmetric-multi-graph*  $G$   
**shows** *symmetric-multi-graph* ( $\text{graph-power } G \ l$ )  
**proof** –  
**interpret**  $H:\text{fin-digraph } \text{graph-power } G \ l$   
**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**  $\text{reg: } \bigwedge v. v \in \text{verts } G \implies \text{out-degree } G \ v = d$   
**assumes**  $v \in \text{verts } (\text{graph-power } G \ l)$   
**shows**  $\text{out-degree } (\text{graph-power } G \ l) \ v = d \wedge l$  (is ?L = ?R)  
**proof** –  
**interpret**  $H:\text{fin-digraph } \text{graph-power } G \ l$   
**using** *graph-power-fin* **by** *auto*  
  
**have**  $v\text{-vert}: v \in \text{verts } G$

**using** *assms unfolding graph-power-def* **by** *simp*

**have**  $?L = \text{size } (\text{vertices-from } (\text{graph-power } G \ l) \ v)$   
**unfolding** *out-degree-def H.verts-from-alt* **by** *simp*

**also have**  $\dots = \text{size } (\{\# \ e \in \# \ \text{edges } (\text{graph-power } G \ l). \ \text{fst } e = v \ \#\})$   
**unfolding** *vertices-from-def* **by** *simp*

**also have**  $\dots = \text{size } \{\# w \in \# \ \text{mset-set } (\text{arc-walks } G \ l). \ \text{fst } w = v \ \#\}$   
**unfolding** *graph-power-def edges-def arc-to-ends-def*  
**by** (*simp add:count-mset-exp image-mset-filter-mset-swap[symmetric]*)

**also have**  $\dots = \text{size } \{\# w \in \# \ \text{mset-set } (\text{arc-walks } G \ l). \ \text{awhd } (\text{fst } w) \ (\text{snd } w) = v \ \#\}$   
**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**  $\dots = \text{size } \{\# x \in \# \ \text{walks}' \ G \ l. \ \text{hd } x = v \ \#\}$   
**unfolding** *arc-walks-map-walks'*  
**by** (*simp add:image-mset-filter-mset-swap[symmetric] case-prod-beta*)

**also have**  $\dots = d^{\wedge} l$

**proof** (*induction l*)

**case** *0*

**have**  $\text{size } \{\# x \in \# \ \text{walks}' \ G \ 0. \ \text{hd } x = v \ \#\} = \text{card } \{x. \ x = v \wedge x \in \text{verts } G\}$   
**by** (*simp add:image-mset-filter-mset-swap[symmetric]*)

**also have**  $\dots = \text{card } \{v\}$   
**using** *v-vert* **by** (*intro arg-cong[where f=card]*) *auto*

**also have**  $\dots = d^{\wedge} 0$  **by** *simp*

**finally show** *?case* **by** *simp*

**next**

**case** (*Suc l*)

**have**  $\text{size } \{\# x \in \# \ \text{walks}' \ G \ (\text{Suc } l). \ \text{hd } x = v \ \#\} =$   
 $(\sum x \in \# \ \text{walks}' \ G \ l. \ \text{size } \{\# y \in \# \ \text{vertices-from } G \ (\text{last } x). \ \text{hd } (x \ @ \ [y]) = v \ \#\})$   
**by** (*simp add:size-concat-mset image-mset-filter-mset-swap[symmetric]*  
*filter-concat-mset image-mset.compositionality comp-def*)

**also have**  $\dots = (\sum x \in \# \ \text{walks}' \ G \ l. \ \text{size } \{\# y \in \# \ \text{vertices-from } G \ (\text{last } x). \ \text{hd } x = v \ \#\})$   
**using** *set-walks-2*  
**by** (*intro-cong [\sigma\_1 sum-mset, \sigma\_1 size] more:image-mset-cong filter-mset-cong*) *auto*

**also have**  $\dots = (\sum x \in \# \ \text{walks}' \ G \ l. \ (\text{if } \text{hd } x = v \ \text{then } \text{out-degree } G \ (\text{last } x) \ \text{else } 0))$   
**unfolding** *verts-from-alt out-degree-def*  
**by** (*simp add:filter-mset-const if-distribR if-distrib cong:if-cong*)

**also have**  $\dots = (\sum x \in \# \ \text{walks}' \ G \ l. \ d * \text{of-bool } (\text{hd } x = v))$   
**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**  $\dots = d * (\sum x \in \# \ \text{walks}' \ G \ l. \ \text{of-bool } (\text{hd } x = v))$   
**by** (*simp add:sum-mset-distrib-left image-mset.compositionality comp-def*)

**also have**  $\dots = d * (\text{size } \{\# x \in \# \ \text{walks}' \ G \ l. \ \text{hd } x = v \ \#\})$   
**by** (*simp add:size-filter-mset-conv*)

**also have**  $\dots = d * d^{\wedge} l$   
**using** *Suc* **by** *simp*

**also have**  $\dots = d^{\wedge} \text{Suc } l$   
**by** *simp*

**finally show** *?case* **by** *simp*

**qed**

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

**qed**

**lemma** (*in regular-graph*) *graph-power-out-degree*:  
**assumes**  $v \in \text{verts } (\text{graph-power } G \ l)$   
**shows**  $\text{out-degree } (\text{graph-power } G \ l) \ v = d^{\wedge} l$  (**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<sup>l</sup>  
using d-gt-0 by simp  
  
ultimately show ?thesis  
using graph-power-out-degree  
by (intro regular-graphI[where d=d<sup>l</sup>] graph-power-sym sym)  
qed

lemma (in regular-graph) graph-power-degree:  
regular-graph.d (graph-power G l) = d<sup>l</sup> (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<sup>l</sup>) 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<sup>0</sup>) 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 [σ2 (/),σ1 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-stepl) f (tail G a) / d)
    by (intro sum.cong refl arg-cong2[where f=(/)] Suc) auto
  also have ... = g-step ((g-stepl) f) x
    unfolding g-step-def by simp
  also have ... = (g-stepl+1) f x
    by simp
  finally show ?case by simp
qed

```

lemma (in regular-graph) graph-power-expansion:

*regular-graph*. $\Lambda_a$  (*graph-power*  $G$   $l$ )  $\leq \Lambda_a \wedge l$

**proof** –

**interpret**  $H$ :*regular-graph* *graph-power*  $G$   $l$

**using** *graph-power-regular* **by** *auto*

**have**  $|H.g\text{-inner } f (H.g\text{-step } f)| \leq \Lambda_a \wedge l * (H.g\text{-norm } f)^2$  (**is**  $?L \leq ?R$ )

**if**  $H.g\text{-inner } f (\lambda-. 1) = 0$  **for**  $f$

**proof** –

**have**  $g\text{-inner } f (\lambda-. 1) = H.g\text{-inner } f (\lambda-.1)$

**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**  $1:g\text{-inner } f (\lambda-. 1) = 0$  **by** *simp*

**have**  $2: g\text{-inner } ((g\text{-step } \wedge l) f) (\lambda-. 1) = 0$  **for**  $l$

**using** *g-step-remains-orth 1* **by** (*induction l, auto*)

**have**  $0: g\text{-norm } ((g\text{-step } \wedge l) f) \leq \Lambda_a \wedge l * g\text{-norm } f$

**proof** (*induction l*)

**case**  $0$

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

**next**

**case** (*Suc l*)

**have**  $g\text{-norm } ((g\text{-step } \wedge \text{Suc } l) f) = g\text{-norm } (g\text{-step } ((g\text{-step } \wedge l) f))$

**by** *simp*

**also have** ...  $\leq \Lambda_a * g\text{-norm } (((g\text{-step } \wedge l) f))$

**by** (*intro expansionD2 2*)

**also have** ...  $\leq \Lambda_a * (\Lambda_a \wedge l * g\text{-norm } f)$

**by** (*intro mult-left-mono  $\Lambda\text{-ge-0 Suc}$* )

**also have** ... =  $\Lambda_a \wedge (l+1) * g\text{-norm } f$  **by** *simp*

**finally show**  $?case$  **by** *simp*

**qed**

**have**  $?L = |g\text{-inner } f (H.g\text{-step } f)|$

**unfolding** *H.g-inner-def g-inner-def*

**by** (*intro-cong [ $\sigma_1$  abs] more:sum.cong*) (*auto simp add:graph-power-def*)

**also have** ... =  $|g\text{-inner } f ((g\text{-step } \wedge l) f)|$

**by** (*intro-cong [ $\sigma_1$  abs] more:g-inner-cong graph-power-step*) *auto*

**also have** ...  $\leq g\text{-norm } f * g\text{-norm } ((g\text{-step } \wedge l) f)$

**by** (*intro g-inner-cauchy-schwartz*)

**also have** ...  $\leq g\text{-norm } f * (\Lambda_a \wedge l * g\text{-norm } f)$

**by** (*intro mult-left-mono 0 g-norm-nonneg*)

**also have** ... =  $\Lambda_a \wedge l * g\text{-norm } f^2$

**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 [ $\sigma_2$  (\*)] more:sum.cong*) (*auto simp add:graph-power-def*)

**finally show**  $?thesis$  **by** *simp*

**qed**

**moreover have**  $0 \leq \Lambda_a \wedge l$

**using**  *$\Lambda\text{-ge-0}$*  **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*

*Finite-Fields.Finite-Fields-Nth-Root-Code*

**begin**

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

**no-notation** *Digraph.dominates* ( $\lambda - \rightarrow_1 - \rightarrow [100, 100] 40$ )

**record** *strongly-explicit-expander* =

*see-size* :: nat

*see-degree* :: nat

*see-step* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat

**definition** *graph-of* :: *strongly-explicit-expander*  $\Rightarrow$  (nat, (nat,nat) arc) *pre-digraph*

**where** *graph-of* e =

$\langle$  *verts* =  $\{..<see-size\ e\}$ ,

*arcs* =  $(\lambda(v, i). \text{Arc } v (see-step\ e\ i\ v)\ i) \text{ ' } (\{..<see-size\ e\} \times \{..<see-degree\ e\})$ ,

*tail* = *arc-tail*,

*head* = *arc-head*  $\rangle$

**definition** *is-expander* e  $\Lambda_a \longleftrightarrow$

*regular-graph* (*graph-of* e)  $\wedge$  *regular-graph*. $\Lambda_a$  (*graph-of* e)  $\leq \Lambda_a$

**lemma** *is-expander-mono*:

**assumes** *is-expander* e a  $a \leq b$

**shows** *is-expander* e b

**using** *assms* **unfolding** *is-expander-def* **by** *auto*

**lemma** *graph-of-finI*:

**assumes** *see-step* e  $\in (\{..<see-degree\ e\} \rightarrow (\{..<see-size\ e\} \rightarrow \{..<see-size\ e\}))$

**shows** *fin-digraph* (*graph-of* e)

**proof** –

**let** ?G = *graph-of* e

**have** *head* ?G a  $\in$  *verts* ?G  $\wedge$  *tail* ?G a  $\in$  *verts* ?G **if** a  $\in$  *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*:  
 $edges(graph-of\ e) = \{\#(v, see-step\ e\ i\ v). (v, i) \in \#mset-set\ (\{..<see-size\ e\} \times \{..<see-degree\ e\})\#\}$

**proof** –  
**have** 0:  $mset-set\ ((\lambda(v, i). Arc\ v\ (see-step\ e\ i\ v)\ i) \text{ ‘ } (\{..<see-size\ e\} \times \{..<see-degree\ e\}))$   
 $= \{\# Arc\ v\ (see-step\ e\ i\ v)\ i. (v, i) \in \#mset-set\ (\{..<see-size\ e\} \times \{..<see-degree\ e\})\#\}$   
**by** (*intro image-mset-mset-set[symmetric] inj-onI*) *auto*

**have** *edges (graph-of e) =*  
 $\{\#(fst\ p, see-step\ e\ (snd\ p)\ (fst\ p)). p \in \#mset-set\ (\{..<see-size\ e\} \times \{..<see-degree\ e\})\#\}$   
**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** ... =  $\{\#(v, see-step\ e\ i\ v). (v, i) \in \#mset-set\ (\{..<see-size\ e\} \times \{..<see-degree\ e\})\#\}$   
**by** (*intro image-mset-cong*) *auto*

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

**qed**

**lemma** *out-degree-see*:  
**assumes**  $v \in vertices\ (graph-of\ e)$   
**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**  $?L = card\ \{a. (\exists x \in \{..<?n\}. \exists y \in \{..<?d\}. a = Arc\ x\ (see-step\ e\ y\ x)\ y) \wedge arc-tail\ a = v\}$   
**unfolding** *out-degree-def out-arcs-def graph-of-def* **by** (*simp add:image-iff*)

**also have** ... =  $card\ \{a. (\exists y \in \{..<?d\}. a = Arc\ v\ (see-step\ e\ y\ v)\ y)\}$   
**using** 0 **by** (*intro arg-cong[where f=card]*) *auto*

**also have** ... =  $card\ ((\lambda y. Arc\ v\ (see-step\ e\ y\ v)\ y) \text{ ‘ } \{..<?d\})$   
**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**  $card\ (arc-walks\ (graph-of\ e)\ n) = see-degree\ e\ \widehat{n} * see-size\ e\ (is\ ?L = ?R)$

**proof** –  
**let**  $?G = graph-of\ e$   
**interpret** *fin-digraph ?G*  
**using** *assms* **by** *auto*

**have**  $?L = card\ (\bigcup v \in vertices\ ?G. \{x. fst\ x = v \wedge is-arc-walk\ ?G\ v\ (snd\ x) \wedge length\ (snd\ x) = n\})$

**unfolding** *arc-walks-def* **by** (*intro arg-cong[where f=card]*) *auto*

**also have** ... =  $(\sum v \in \text{verts } ?G. \text{card } \{x. \text{fst } x=v \wedge \text{is-arc-walk } ?G \ v \ (\text{snd } x) \wedge \text{length } (\text{snd } x) = n\})$   
**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**  $B=\text{verts } ?G \times \{x. \text{set } x \subseteq \text{arcs } ?G \wedge \text{length } x = n\}$ ]) *force+*  
**also have** ... =  $(\sum v \in \text{verts } ?G. \text{out-degree } (\text{graph-power } ?G \ n) \ v)$   
**unfolding** *out-degree-def graph-power-def out-arcs-def arc-walks-def*  
**by** (*intro sum.cong arg-cong* [**where**  $f=\text{card}$ ]) *auto*  
**also have** ... =  $(\sum v \in \text{verts } ?G. \text{see-degree } e \wedge n)$   
**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**  $v\text{-set}: v \in \text{verts } (\text{graph-of } e)$   
**using** *verts-non-empty* **by** *auto*  
**hence**  $?L = \text{out-degree } (\text{graph-of } e) \ v$   
**using**  $v\text{-set}$  *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* ::  $\text{nat} \Rightarrow \text{strongly-explicit-expander} \Rightarrow \text{strongly-explicit-expander}$

**where** *see-compress*  $m \ e =$   
 $(\downarrow \text{see-size} = m, \text{see-degree} = \text{see-degree } e * 2$   
 $, \text{see-step} = (\lambda k \ v.$   
 $\text{if } k < \text{see-degree } e$   
 $\text{then } (\text{see-step } e \ k \ v) \ \text{mod } m$   
 $\text{else } (\text{if } v+m < \text{see-size } e \ \text{then } (\text{see-step } e \ (k-\text{see-degree } e) \ (v+m)) \ \text{mod } m \ \text{else } v)) \ \downarrow)$

**lemma** *edges-of-compress*:

**fixes**  $e \ m$   
**assumes**  $2*m \geq \text{see-size } e \ m \leq \text{see-size } e$   
**defines**  $A \equiv \{\#\ (x \ \text{mod } m, y \ \text{mod } m). (x,y) \in \# \text{edges } (\text{graph-of } e)\#\}$   
**defines**  $B \equiv \text{repeat-mset } (\text{see-degree } e) \ \{\#\ (x,x). x \in \# (\text{mset-set } \{\text{see-size } e - m..<m\})\#\}$   
**shows** *edges* (*graph-of* (*see-compress*  $m \ e$ )) =  $A + B$  (**is**  $?L = ?R$ )

**proof** –

**let**  $?d = \text{see-degree } e$   
**let**  $?c = \text{see-step } (\text{see-compress } m \ e)$   
**let**  $?n = \text{see-size } e$   
**let**  $?s = \text{see-step } e$

**have**  $\exists m \leq v \implies v < ?n \implies v - m = v \ \text{mod } m$  **for**  $v$   
**using** *assms* **by** (*simp add: le-mod-geq*)

**let**  $?M = \text{mset-set } (\{..<m\} \times \{..<2*?d\})$   
**define**  $M1$  **where**  $M1 = \text{mset-set } (\{..<m\} \times \{..<?d\})$   
**define**  $M2$  **where**  $M2 = \text{mset-set } (\{..<?n-m\} \times \{?d..<2*?d\})$   
**define**  $M3$  **where**  $M3 = \text{mset-set } (\{?n-m..<m\} \times \{?d..<2*?d\})$

**have**  $M2 = \text{mset-set } ((\lambda(x,y). (x-m, y+?d)) \text{ ‘ } (\{m..<?n\} \times \{..<?d\}))$   
**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**  $\dots = \text{image-mset } (\lambda(x,y). (x-m, y+?d)) (\text{mset-set } (\{m..<?n\} \times \{..<?d\}))$   
**by** (*intro image-mset-mset-set[symmetric] inj-onI*) *auto*  
**finally have**  $M2\text{-eq}: M2 = \text{image-mset } (\lambda(x,y). (x-m, y+?d)) (\text{mset-set } (\{m..<?n\} \times \{..<?d\}))$   
**by** *simp*

**have**  $?M = \text{mset-set } (\{..<m\} \times \{..<?d\} \cup \{..<?n-m\} \times \{?d..<2*?d\} \cup \{?n-m..<m\} \times \{?d..<2*?d\})$   
**using** *assms(1,2)* **by** (*intro arg-cong[where f=mset-set]*) *auto*  
**also have**  $\dots = \text{mset-set } (\{..<m\} \times \{..<?d\} \cup \{..<?n-m\} \times \{?d..<2*?d\}) + M3$   
**unfolding** *M3-def* **by** (*intro mset-set-Union*) *auto*  
**also have**  $\dots = M1 + M2 + M3$   
**unfolding** *M1-def M2-def*  
**by** (*intro arg-cong2[where f=(+)] mset-set-Union*) *auto*  
**finally have**  $0:\text{mset-set } (\{..<m\} \times \{..<2*?d\}) = M1 + M2 + M3$  **by** *simp*

**have**  $1:\{\#(v, ?c \ i \ v). (v, i) \in \#M1 \ \#\} = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{..<m\} \times \{..<?d\}) \ \#\}$   
**unfolding** *M1-def* **by** (*intro image-mset-cong*) *auto*

**have**  $\{\#(v, ?c \ i \ v). (v, i) \in \#M2 \ \#\} = \{\#(\text{fst } x - m, ?c(\text{snd } x + ?d)(\text{fst } x - m)). x \in \# \text{mset-set } (\{m..<?n\} \times \{..<?d\}) \ \#\}$   
**unfolding** *M2-eq*  
**by** (*simp add:image-mset.compositionality comp-def case-prod-beta del:see-compress.simps*)  
**also have**  $\dots = \{\#(v - m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{m..<?n\} \times \{..<?d\}) \ \#\}$   
**by** (*intro image-mset-cong*) *auto*  
**also have**  $\dots = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{m..<?n\} \times \{..<?d\}) \ \#\}$   
**using** 7 **by** (*intro image-mset-cong*) *auto*  
**finally have** 2:  
 $\{\#(v, ?c \ i \ v). (v, i) \in \#M2 \ \#\} = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{m..<?n\} \times \{..<?d\}) \ \#\}$   
**by** *simp*

**have**  $\{\#(v, ?c \ i \ v). (v, i) \in \#M3 \ \#\} = \{\#(v, v). (v, i) \in \# \text{mset-set } (\{?n-m..<m\} \times \{?d..<2*?d\}) \ \#\}$   
**unfolding** *M3-def* **by** (*intro image-mset-cong*) *auto*  
**also have**  $\dots = \text{concat-mset } \{\#\{\#(x, x). x \in \# \text{mset-set } \{?d..<2 * ?d\} \ \#\}. x \in \# \text{mset-set } \{?n - m..<m\} \ \#\}$   
**by** (*subst mset-prod-eq*) (*auto simp:image-mset.compositionality image-concat-mset comp-def*)  
**also have**  $\dots = \text{concat-mset } \{\#\text{replicate-mset } ?d \ (x, x). x \in \# \text{mset-set } \{?n - m..<m\} \ \#\}$   
**unfolding** *image-mset-const-eq* **by** *simp*  
**also have**  $\dots = B$   
**unfolding** *B-def repeat-image-concat-mset* **by** *simp*  
**finally have** 3: $\{\#(v, ?c \ i \ v). (v, i) \in \#M3 \ \#\} = B$  **by** *simp*

**have**  $A = \{\#(\text{fst } x \ \text{mod} \ m, ?s (\text{snd } x) (\text{fst } x) \ \text{mod} \ m). x \in \# \text{mset-set } (\{..<?n\} \times \{..<?d\}) \ \#\}$   
**unfolding** *A-def edges-graph-of* **by** (*simp add:image-mset.compositionality comp-def case-prod-beta*)  
**also have**  $\dots = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{..<?n\} \times \{..<?d\}) \ \#\}$   
**by** (*intro image-mset-cong*) *auto*  
**finally have** 4:  $A = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{..<?n\} \times \{..<?d\}) \ \#\}$   
**by** *simp*

**have**  $?L = \{\#(v, ?c \ i \ v). (v, i) \in \# ?M \ \#\}$   
**unfolding** *edges-graph-of* **by** (*simp add:ac-simps*)  
**also have**  $\dots = \{\#(v, ?c \ i \ v). (v, i) \in \#M1 \ \#\} + \{\#(v, ?c \ i \ v). (v, i) \in \#M2 \ \#\} + \{\#(v, ?c \ i \ v). (v, i) \in \#M3 \ \#\}$   
**unfolding** 0 *image-mset-union* **by** *simp*  
**also have**  $\dots = \{\#(v \ \text{mod} \ m, ?s \ i \ v \ \text{mod} \ m). (v, i) \in \# \text{mset-set } (\{..<m\} \times \{..<?d\}) \cup \{m..<?n\} \times \{..<?d\}) \ \#\} + B$   
**unfolding** 1 2 3 *image-mset-union[symmetric]*  
**by** (*intro-cong*  $[\sigma_2 (+), \sigma_2 \ \text{image-mset}]$  *more: mset-set-Union[symmetric]*) *auto*

**also have** ... =  $\{\#(v \bmod m, ?s \ i \ v \bmod m). (v, i) \in \# \text{mset-set}(\{.. < ?n\} \times \{.. < ?d\}) \#\} + B$   
**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**  $2 * m \geq \text{see-size } e \ m \leq \text{see-size } e$   
**assumes** *symmetric-multi-graph* (*graph-of*  $e$ )  
**shows** *symmetric-multi-graph* (*graph-of* (*see-compress*  $m \ e$ ))

**proof** –

**let**  $?c = \text{see-compress } m \ e$   
**let**  $?d = \text{see-degree } e$   
**let**  $?G = \text{graph-of } e$   
**let**  $?H = \text{graph-of } (\text{see-compress } m \ e)$

**interpret**  $G: \text{fin-digraph } ?G$   
**by** (*intro symmetric-multi-graphD2*[*OF assms(3)*])  
**interpret**  $H: \text{fin-digraph } ?H$   
**by** (*intro graph-of-finI*) *simp*

**have** *deg-compres*:  $\text{see-degree } ?c = 2 * \text{see-degree } e$   
**by** *simp*

**have** 1:  $\text{card } (\text{arcs-betw } ?H \ v \ w) = \text{card } (\text{arcs-betw } ?H \ w \ v)$  (**is**  $?L = ?R$ )  
**if**  $v \in \text{verts } ?H \ w \in \text{verts } ?H$  **for**  $v \ w$

**proof** –

**define**  $b$  **where**  $b = \text{count } \{\#(x, x). x \in \# \text{mset-set } \{\text{see-size } e - m.. < m\} \#\} (v, w)$

**have** *b-alt-def*:  $b = \text{count } \{\#(x, x). x \in \# \text{mset-set } \{\text{see-size } e - m.. < m\} \#\} (w, v)$   
**unfolding** *b-def count-mset-exp*  
**by** (*simp add: case-prod-beta image-mset-filter-mset-swap*[*symmetric*] *ac-simps*)

**have**  $?L = \text{count } (\text{edges } ?H) (v, w)$   
**unfolding** *H.count-edges* **by** *simp*

**also have** ... =  $\text{count } \{\#(x \bmod m, y \bmod m). (x, y) \in \# \text{edges } (\text{graph-of } e) \#\} (v, w) + ?d * b$   
**unfolding** *edges-of-compress*[*OF assms(1,2)*] *b-def* **by** *simp*

**also have** ... =  $\text{count } \{\#(\text{snd } e \bmod m, \text{fst } e \bmod m). e \in \# \text{edges } (\text{graph-of } e) \#\} (v, w) + ?d * b$   
**by** (*subst G.edges-sym*[*OF assms(3)*, *symmetric*])

(*simp add: image-mset.compositionality comp-def case-prod-beta*)

**also have** ... =  $\text{count } \{\#(x \bmod m, y \bmod m). (x, y) \in \# \text{edges } (\text{graph-of } e) \#\} (w, v) + ?d * b$   
**unfolding** *count-mset-exp*

**by** (*simp add: image-mset-filter-mset-swap*[*symmetric*] *ac-simps case-prod-beta*)

**also have** ... =  $\text{count } (\text{edges } ?H) (w, v)$

**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**  $2*m \geq \text{see-size } e \ m \leq \text{see-size } e$   
**shows** *is-expander* (*see-compress*  $m e$ ) ( $\Lambda_a/2 + 1/2$ )  
**proof** –  
**let**  $?H = \text{graph-of } (\text{see-compress } m e)$   
**let**  $?G = \text{graph-of } e$   
**let**  $?d = \text{see-degree } e$   
**let**  $?n = \text{see-size } e$   
  
**interpret**  $G:\text{regular-graph graph-of } e$   
**using** *assms(1)* *is-expander-def* **by** *simp*  
  
**have**  $d\text{-eq}: ?d = G.d$   
**using** *regular-graph-degree-eq-see-degree*[*OF G.regular-graph-axioms*] **by** *simp*  
  
**have**  $n\text{-eq}: G.n = ?n$   
**unfolding**  $G.n\text{-def}$  **by** (*simp add:graph-of-def*)  
  
**have**  $n\text{-gt-1}: ?n > 0$   
**using**  $G.n\text{-gt-0 } n\text{-eq}$  **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**  $G.\text{verts-non-empty}$  **unfolding** *graph-of-def* **by** *auto*  
**hence**  $m > 0$  **using** *assms(2)* **by** *simp*  
**hence** *verts* (*graph-of* (*see-compress*  $m e$ ))  $\neq \{\}$   
**unfolding** *graph-of-def* **by** *auto*  
**moreover** **have**  $1:0 < \text{see-degree } e$   
**using**  $d\text{-eq } G.d\text{-gt-0}$  **by** *auto*  
**hence**  $0 < \text{see-degree } (\text{see-compress } m e)$  **by** *simp*  
**ultimately** **have**  $0:\text{regular-graph } ?H$   
**by** (*intro regular-graphI[where d=see-degree (see-compress m e)] out-degree-see*) *auto*  
**interpret**  $H:\text{regular-graph } ?H$   
**using**  $0$  **by** *auto*  
  
**have**  $|\sum a \in \text{arcs } ?H. f(\text{head } ?H a) * f(\text{tail } ?H a)| \leq (\text{real } G.d * G.\Lambda_a + G.d) * (H.g\text{-norm } f)^2$   
**(is ?L ≤ ?R) if**  $H.g\text{-inner } f(\lambda. 1) = 0$  **for**  $f$   
**proof** –  
**define**  $f'$  **where**  $f' x = f(x \bmod m)$  **for**  $x$   
**let**  $?L1 = G.g\text{-norm } f'^{\wedge}2 + |\sum x = ?n - m .. < m. f x^{\wedge}2|$   
**let**  $?L2 = G.g\text{-inner } f'(\lambda. 1)^{\wedge}2 / G.n + |\sum x = ?n - m .. < m. f x^{\wedge}2|$   
  
**have**  $?L1 = (\sum x < ?n. f(x \bmod m)^{\wedge}2) + |\sum x = ?n - m .. < m. f x^{\wedge}2|$   
**unfolding**  $G.g\text{-norm-sq } G.g\text{-inner-def } f'\text{-def}$  **by** (*simp add:graph-of-def power2-eq-square*)  
**also** **have**  $\dots = (\sum x \in \{0..<m\} \cup \{m..<?n\}. f(x \bmod m)^{\wedge}2) + (\sum x = ?n - m .. < m. f x^{\wedge}2)$   
**using** *assms(3)* **by** (*intro-cong [\sigma\_2 (+)] more:sum.cong abs-of-nonneg sum-nonneg*) *auto*  
**also** **have**  $\dots = (\sum x = 0 .. < m. f(x \bmod m)^{\wedge}2) + (\sum x = m .. < ?n. f(x \bmod m)^{\wedge}2) + (\sum x = ?n - m .. < m. f x^{\wedge}2)$   
**by** (*intro-cong [\sigma\_2 (+)] more:sum.union-disjoint*) *auto*  
**also** **have**  $\dots = (\sum x = 0 .. < m. f(x \bmod m)^{\wedge}2) + (\sum x = 0 .. < ?n - m. f x^{\wedge}2) + (\sum x = ?n - m .. < m. f x^{\wedge}2)$   
**using** *assms(2,3)*  
**by** (*intro-cong [\sigma\_2 (+)] more: sum.reindex-bij-betw bij-betwI[where g=(\lambda x. x+m)]*)  
*(auto simp add:le-mod-geq)*  
**also** **have**  $\dots = (\sum x = 0 .. < m. f x^{\wedge}2) + (\sum x = 0 .. < ?n - m. f x^{\wedge}2) + (\sum x = ?n - m .. < m. f x^{\wedge}2)$   
**by** (*intro sum.cong arg-cong2[where f=(+)]*) *auto*  
**also** **have**  $\dots = (\sum x = 0 .. < m. f x^{\wedge}2) + ((\sum x = 0 .. < ?n - m. f x^{\wedge}2) + (\sum x = ?n - m .. < m. f x^{\wedge}2))$

**by simp**  
**also have** ... =  $(\sum_{x=0..<m}. f x^2) + (\sum_{x \in \{0..<?n-m\} \cup \{?n-m..<m\}}. f x^2)$   
**by** (*intro sum.union-disjoint[symmetric] arg-cong2[where f=(+)] auto*)  
**also have** ... =  $(\sum_{x<m}. f x^2) + (\sum_{x<m}. f x^2)$   
**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**  $?L2 = (\sum_{x \in \{..<m\} \cup \{m..<?n\}}. f (x \text{ mod } m))^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**unfolding** *G.g-inner-def f'-def* **using** *assms(2,3)*  
**by** (*intro-cong [\sigma\_2 (+), \sigma\_2 (/), \sigma\_2 (power)] more: sum.cong abs-of-nonneg sum-nonneg*)  
*(auto simp add:graph-of-def)*  
**also have** ... =  $(\sum_{x<m}. f (x \text{ mod } m)) + (\sum_{x=m..<?n}. f (x \text{ mod } m))^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**by** (*intro-cong [\sigma\_2 (+), \sigma\_2 (/), \sigma\_2 (power)] more: sum.union-disjoint*) *auto*  
**also have** ... =  $(\sum_{x<m}. f (x \text{ mod } m)) + (\sum_{x=0..<?n-m}. f x)^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**using** *assms(2,3) by (intro-cong [\sigma\_2 (+), \sigma\_2 (/), \sigma\_2 (power)]*  
*more: sum.reindex-bij-betw bij-betwI[where g=(\lambda x. x+m)] (auto simp add:le-mod-geq)*)  
**also have** ... =  $(H.g-inner f (\lambda-. 1) + (\sum_{x<?n-m}. f x))^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**unfolding** *H.g-inner-def*  
**by** (*intro-cong [\sigma\_2 (+), \sigma\_2 (/), \sigma\_2 (power)] more: sum.cong*) *(auto simp:graph-of-def)*  
**also have** ... =  $(\sum_{x<?n-m}. f x)^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**unfolding** *that* **by simp**  
**also have** ...  $\leq (\sum_{x<?n-m}. |f x| * |1|)^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**by** (*intro add-mono divide-right-mono iffD1[OF abs-le-square-iff]*) *auto*  
**also have** ...  $\leq (L2\text{-set } f \{..<?n-m\} * L2\text{-set } (\lambda-. 1) \{..<?n-m\})^2 / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**by** (*intro add-mono divide-right-mono power-mono L2-set-mult-ineq sum-nonneg*) *auto*  
**also have** ... =  $((\sum_{x<?n-m}. f x^2) * (?n-m)) / G.n + (\sum_{x=?n-m..<m}. f x^2)$   
**unfolding** *power-mult-distrib L2-set-def real-sqrt-mult*  
**by** (*intro-cong [\sigma\_2 (+), \sigma\_2 (/), \sigma\_2 (\*)] more: real-sqrt-pow2 sum-nonneg*) *auto*  
**also have** ... =  $(\sum_{x<?n-m}. f x^2) * ((?n-m) / ?n) + (\sum_{x=?n-m..<m}. f x^2)$   
**unfolding** *n-eq* **by simp**  
**also have** ...  $\leq (\sum_{x<?n-m}. f x^2) * 1 + (\sum_{x=?n-m..<m}. f x^2)$   
**using** *assms(3) n-gt-1* **by** (*intro mult-left-mono add-mono sum-nonneg*) *auto*  
**also have** ... =  $(\sum_{x \in \{..<?n-m\} \cup \{?n-m..<m\}}. f x^2)$   
**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**  $3: ?L2 \leq H.g-norm f^2$  **by simp**

**have**  $?L = |\sum_{(u,v) \in \# \text{edges } ?H}. f v * f u|$   
**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** ... =  $|\sum_{x \in \# \text{edges } ?G}. f(\text{snd } x \text{ mod } m) * f(\text{fst } x \text{ mod } m)| + (\sum_{x=?n-m..<m}. ?d * (f x^2))|$   
**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** ... =  $|\sum_{(u,v) \in \# \text{edges } ?G}. f(u \text{ mod } m) * f(v \text{ mod } m)| + (\sum_{x=?n-m..<m}. ?d * (f x^2))|$   
**by** (*intro-cong [\sigma\_1 abs, \sigma\_2 (+), \sigma\_1 sum-mset] more:image-mset-cong*)  
*(simp-all add:case-prod-beta)*  
**also have** ...  $\leq |\sum_{(u,v) \in \# \text{edges } ?G}. f(u \text{ mod } m) * f(v \text{ mod } m)| + |\sum_{x=?n-m..<m}. ?d * (f x^2)|$   
**by** (*intro abs-triangle-ineq*)

**also have** ... = ?d \* (| $\sum (u,v) \in \# \text{ edges } ?G.f(v \text{ mod } m) * f(u \text{ mod } m) / G.d + |\sum x = ?n - m .. < m. (f x^2)$ |)

**unfolding** *d-eq using G.d-gt-0*

**by** (*simp add:divide-simps ac-simps sum-distrib-left[symmetric] abs-mult*)

**also have** ... = ?d \* (| $G.g\text{-inner } f' (G.g\text{-step } f')$ | + | $\sum x = ?n - m .. < m. f x^2$ |)

**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** ...  $\leq$  ?d \* (( $G.\Lambda_a * G.g\text{-norm } f'^2 + (1 - G.\Lambda_a) * G.g\text{-inner } f' (\lambda.1)^2 / G.n$ ) + | $\sum x = ?n - m .. < m. f x^2$ |)

**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** ...  $\leq$  ?d \* ( $G.\Lambda_a * (2 * H.g\text{-norm } f'^2) + (1 - G.\Lambda_a) * H.g\text{-norm } f'^2$ )

**unfolding** *2 using G. $\Lambda$ -ge-0 G. $\Lambda$ -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**  $H.\Lambda_a \leq (G.d * G.\Lambda_a + G.d) / H.d$

**using** *G.d-gt-0 G. $\Lambda$ -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** ...  $\leq \Lambda_a / 2 + 1 / 2$

**using** *assms(1)* **unfolding** *is-expander-def* **by** *simp*

**finally have**  $H.\Lambda_a \leq \Lambda_a / 2 + 1 / 2$  **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*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *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*  $\Rightarrow$  *nat list*  $\Rightarrow$  *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  $\subseteq$  {..**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  $\subseteq$  {..**b**}*

**shows** *from-digits b xs < b^n*

**proof** (*cases b > 0*)

**case** *True*

```

have from-digits b xs ≤ blength 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 * (blength xs-1)
    using Cons(2) by (intro add-mono mult-left-mono Cons(1)) auto
  also have ... = blength (a#xs) - 1
    using True by (simp add:algebra-simps)
  finally show from-digits b (a # xs) ≤ blength (a#xs) - 1 by simp
qed
also have ... < bn
  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 el
    , 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**  $length\ xs = length\ ys$   $map\ arc-label\ xs = map\ arc-label\ ys$   
 $is-arc-walk\ ?G\ u\ xs \wedge is-arc-walk\ ?G\ u\ ys \wedge u \in verts\ ?G$  **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**  $arc-label\ x = arc-label\ y$   $u \in verts\ ?G$   $x \in out-arcs\ ?G$   $u \in out-arcs\ ?G$   
**using** *Cons* **by** *auto*  
**hence**  $a:x = y$   
**unfolding** *graph-of-def* **by** *auto*  
**moreover have**  $head\ ?G\ y \in verts\ ?G$  **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**  $5:inj-on\ (\lambda(u,v). (u, map\ arc-label\ v))$  (*arc-walks ?G n*)  
**unfolding** *arc-walks-def* **by** (*intro inj-onI*) *auto*  
**have**  $3:set\ (map\ arc-label\ (snd\ xs)) \subseteq \{..<?d\}$   $length\ (snd\ xs) = n$   
**if**  $xs \in arc-walks\ ?G\ n$  **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**  $set\ (snd\ xs) \subseteq arcs\ ?G$   
**using** *subsetD[OF is-arc-walk-set[where G=?G]]* **that** **unfolding** *arc-walks-def* **by** *auto*  
**thus**  $set\ (map\ arc-label\ (snd\ xs)) \subseteq \{..<?d\}$   
**unfolding** *graph-of-def* **by** *auto*  
**qed**  
**hence**  $7:inj-on\ (\lambda(u,v). (u, from-digits\ ?d\ (map\ arc-label\ v)))$  (*arc-walks ?G n*)  
**using** *inj-onD[OF 5]* *inj-onD[OF from-digits-inj]* **by** (*intro inj-onI*) *auto*  
**hence** *inj-on*  $\varphi$  (*arc-walks ?G n*)  
**unfolding** *inj-on-def*  $\varphi$ -*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**  $a \in arcs\ ?P$  **for**  $a$   
**unfolding**  $\varphi$ -*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**  $card((\lambda(u, v).(u, from-digits\ ?d\ (map\ arc-label\ v)))'arc-walks\ ?G\ n) = card(arc-walks\ ?G\ n)$   
**by** (*intro card-image 7*)  
**also have**  $\dots = ?d^{\wedge}n * ?n$   
**by** (*intro card-arc-walks-see fin-digraph-axioms*)  
**finally have**  $card((\lambda(u, v).(u, from-digits\ ?d\ (map\ arc-label\ v)))'arc-walks\ ?G\ n) = ?d^{\wedge}n * ?n$   
**by** *simp*  
**moreover have**  $fst\ v \in \{..<?n\}$  **if**  $v \in arc-walks\ ?G\ n$  **for**  $v$   
**using** *that* **unfolding** *arc-walks-def* *graph-of-def* **by** *auto*  
**moreover have**  $from-digits\ ?d\ (map\ arc-label\ (snd\ v)) < ?d^{\wedge}n$  **if**  $v \in arc-walks\ ?G\ n$  **for**  $v$   
**using**  $3[OF\ that]$  **by** (*intro from-digits-range*) *auto*

ultimately have 2:

$\{..<?n\} \times \{..<?d\hat{n}\} = (\lambda(u,v). (u, \text{from-digits } ?d (\text{map arc-label } v))) \text{ ' arc-walks } ?G n$   
 by (intro card-subset-eq[symmetric]) auto

have foldl ( $\lambda y x. \text{see-step } e x y$ )  $u (\text{map arc-label } w) = \text{arc-walk-head } ?G (u,w)$

if is-arc-walk  $?G u w$   $u \in \text{verts } ?G$  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  $x \in \text{arcs } ?G$  by (simp add:is-arc-walk-snoc)

hence  $\text{see-step } e (\text{arc-label } x) (\text{tail } ?G x) = (\text{head } ?G x)$

unfolding graph-of-def by (auto simp add:image-iff)

also have  $\dots = \text{arc-walk-head } (\text{graph-of } e) (u, xs @ [x])$

unfolding arc-walk-head-def by simp

finally have  $\text{see-step } e (\text{arc-label } x) (\text{tail } ?G x) = \text{arc-walk-head } (\text{graph-of } e) (u, xs @ [x])$

by simp

thus ?case using snoc by (simp add:is-arc-walk-snoc)

qed

hence 4: foldl ( $\lambda y x. \text{see-step } e x y$ ) (fst  $x$ ) ( $\text{map arc-label } (\text{snd } x)$ ) =  $\text{arc-walk-head } ?G x$

if  $x \in \text{arc-walks } (\text{graph-of } e) n$  for  $x$

using that unfolding arc-walks-def by (simp add:case-prod-beta)

have arcs  $?H = (\lambda(v, i). \text{Arc } v (\text{see-step } (\text{see-power } n e) i v) i) \text{ ' } (\{..<?n\} \times \{..<?d\hat{n}\})$

unfolding graph-of-def by simp

also have  $\dots = (\lambda(v,w). \text{Arc } v (\text{see-step } (\text{see-power } n e) (\text{from-digits } ?d (\text{map arc-label } w)) v) (\text{from-digits } ?d (\text{map arc-label } w))) \text{ ' arc-walks } ?G n$

unfolding 2 image-image by (simp del:see-power.simps add: case-prod-beta comp-def)

also have  $\dots = (\lambda(v,w). \text{Arc } v (\text{foldl } (\lambda y x. \text{see-step } e x y) v (\text{map arc-label } w)) (\text{from-digits } ?d (\text{map arc-label } w))) \text{ ' arc-walks } ?G n$

using 3 by (intro image-cong refl) (simp add:case-prod-beta to-from-digits)

also have  $\dots = \varphi \text{ ' arc-walks } ?G n$

unfolding  $\varphi$ -def using 4 by (simp add:case-prod-beta)

also have  $\dots = \text{iso-arcs iso ' arcs } ?P$

unfolding iso-def graph-power-def by simp

finally have arcs  $?H = \text{iso-arcs iso ' arcs } ?P$

by simp

moreover have  $\text{verts } ?H = \text{iso-verts iso ' verts } ?P$

unfolding iso-def graph-of-def graph-power-def by simp

moreover have  $\text{tail } ?H = \text{iso-tail iso}$

unfolding iso-def graph-of-def by simp

moreover have  $\text{head } ?H = \text{iso-head iso}$

unfolding iso-def graph-of-def by simp

ultimately have 1:  $?H = \text{app-iso iso } ?P$

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  $n e$ ) ( $\Lambda_a \hat{n}$ )

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.Λa (graph-of (see-power n e)) = H.Λa
  using H.regular-graph-iso-expansion[OF 0] by auto
also have ... ≤ G.Λa  $\hat{n}$ 
  by (intro G.graph-power-expansion)
also have ... ≤ Λa  $\hat{n}$ 
  using assms(1) unfolding is-expander-def
  by (intro power-mono G.Λ-ge-0) auto
finally have regular-graph.Λa (graph-of (see-power n e)) ≤ Λa  $\hat{n}$ 
  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 = n2, see-degree = 8,
    see-step = (λi 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**  $\varphi$  **where**  $\varphi a = \text{Arc } (?vi \text{ (arc-tail } a)) \text{ } (?vi \text{ (arc-head } a)) \text{ } (?ai \text{ (arc-label } a))$  **for**  $a$

**define**  $iso$  **where**  $iso =$

(|  $iso\text{-verts} = \text{mgg-vert-inv } n$ ,  $iso\text{-arcs} = \varphi$ ,  $iso\text{-head} = \text{arc-head}$ ,  $iso\text{-tail} = \text{arc-tail}$  |)

**interpret**  $M$ : *margulis-gaber-galil*  $n$

**using** *assms* **by** *unfold-locales*

**have**  $inj\text{-vi}$ :  $inj\text{-on } ?vi \text{ (verts } M.G)$

**unfolding** *mgg-graph-def* *mgg-vert-inv-def*

**by** (*intro inj-on-inverseI*[**where**  $g = \text{mgg-vert } n$ ]) (*auto simp:mgg-vert-def*)

**have**  $card \text{ } (?vi \text{ ' } \text{verts } M.G) = card \text{ (verts } M.G)$

**by** (*intro card-image inj-vi*)

**moreover** **have**  $card \text{ (verts } M.G) = n^2$

**unfolding** *mgg-graph-def* **by** (*auto simp:power2-eq-square*)

**moreover** **have**  $\text{mgg-vert-inv } n \ x \in \{..<n^2\}$  **if**  $x \in \text{verts } M.G$  **for**  $x$

**proof** –

**have**  $\text{mgg-vert-inv } n \ x = nat \text{ (fst } x) + nat \text{ (snd } x) * n$

**unfolding** *mgg-vert-inv-def* **by** *simp*

**also** **have**  $... \leq (n-1) + (n-1) * n$

**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**  $\text{mgg-vert-inv } n \ x < n^2$  **by** *simp*

**thus** *?thesis* **by** *simp*

**qed**

**ultimately** **have**  $0:\{..<n^2\} = ?vi \text{ ' } \text{verts } M.G$

**by** (*intro card-subset-eq[symmetric] image-subsetI*) *auto*

**have**  $inj\text{-ai}$ :  $inj\text{-on } ?ai \text{ } (\{..<4\} \times \{-1,1\})$

**unfolding** *mgg-arc-inv-def* **by** (*intro inj-onI*) *auto*

**have**  $card \text{ } (?ai \text{ ' } (\{..<4\} \times \{-1,1\})) = card \text{ } (\{..<4::nat\} \times \{-1,1::int\})$

**by** (*intro card-image inj-ai*)

**hence**  $1:\{..<8\} = ?ai \text{ ' } (\{..<4\} \times \{-1,1\})$

**by** (*intro card-subset-eq[symmetric] image-subsetI*) (*auto simp add:mgg-arc-inv-def*)

**have**  $arcs \ ?G = (\lambda(v, i). \text{Arc } v \text{ } (?vi \text{ } (?s \text{ } (?v \ v) \text{ } (?a \ i))) \ i) \text{ ' } (\{..<n^2\} \times \{..<8\})$

**by** (*simp add:see-mgg-def graph-of-def*)

**also** **have**  $... = (\lambda(v, i). \text{Arc } (?vi \ v) \text{ } (?vi \text{ } (?s \text{ } (?v \text{ } (?vi \ v)) \text{ } (?a \text{ } (?ai \ i)))) \text{ } (?ai \ i) \text{ ' } (verts \ M.G \times (\{..<4\} \times \{-1,1\})))$

**unfolding**  $0 \ 1$  *mgg-arc-inv* **by** (*auto simp add:image-iff*)

**also** **have**  $... = (\lambda(v, i). \text{Arc } (?vi \ v) \text{ } (?vi \text{ } (?s \ v \ i)) \text{ } (?ai \ i) \text{ ' } (verts \ M.G \times (\{..<4\} \times \{-1,1\})))$

**using** *mgg-vert-inv[OF assms]* *mgg-arc-inv* **unfolding** *mgg-graph-def* **by** (*intro image-cong*)

*auto*

**also** **have**  $... = (\varphi \circ (\lambda(t, l). \text{Arc } t \text{ } (?s \ t \ l) \ l)) \text{ ' } (verts \ M.G \times (\{..<4\} \times \{-1,1\}))$

**unfolding**  $\varphi\text{-def}$  **by** (*intro image-cong refl*) (*simp add:comp-def case-prod-beta*)

**also** **have**  $... = \varphi \text{ ' } arcs \ M.G$

**unfolding** *mgg-graph-def* **by** (*simp add:image-image*)

**also** **have**  $... = iso\text{-arcs } iso \text{ ' } arcs \text{ (mgg-graph } n)$

**unfolding** *iso-def* **by** *simp*

**finally** **have**  $arcs \text{ (graph-of } (see\text{-mgg } n)) = iso\text{-arcs } iso \text{ ' } arcs \text{ (mgg-graph } n)$

**by** *simp*

**moreover** **have**  $verts \ ?G = iso\text{-verts } iso \text{ ' } verts \text{ (mgg-graph } n)$

**unfolding** *iso-def graph-of-def see-mgg-def* **using**  $0$  **by** *simp*

**moreover** **have**  $tail \ ?G = iso\text{-tail } iso$

**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**  $\dots \leq (5 * \text{sqrt } 2 / 8)$   
**using** *G.mgg-numerical-radius* **unfolding** *G.MGG-bound-def* **by** *simp*  
**finally have** *regular-graph*. $\Lambda_\alpha$  (*graph-of* (*see-mgg*  $n$ ))  $\leq (5 * \text{sqrt } 2 / 8)$   
**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*  $n = \text{see-compress } n$  (*see-mgg* ( $\text{nat } \lceil \text{sqrt } n \rceil$ ))

**lemma** *see-standard-aux*:

**assumes**  $n > 0$

**shows**

*is-expander* (*see-standard-aux*  $n$ )  $((8+5 * \text{sqrt } 2) / 16)$  (**is** *?A*)

*see-degree* (*see-standard-aux*  $n$ ) = 16 (**is** *?B*)

*see-size* (*see-standard-aux*  $n$ ) =  $n$  (**is** *?C*)

**proof** –

**have**  $2:\text{sqrt } (\text{real } n) > -1$

**by** (*rule less-le-trans*[**where**  $y=0$ ]) *auto*

**have**  $0:\text{real } n \leq \text{of-int } \lceil \text{sqrt } (\text{real } n) \rceil^2$

**by** (*simp add:sqrt-le-D*)

**consider** (*a*)  $n = 1$  | (*b*)  $n \geq 2 \wedge n \leq 4$  | (*c*)  $n \geq 5 \wedge n \leq 9$  | (*d*)  $n \geq 10$

**using** *assms* **by** *linarith*

**hence**  $1:\text{of-int } \lceil \text{sqrt } (\text{real } n) \rceil^2 \leq 2 * \text{real } n$

**proof** (*cases*)

**case** *a* **then show** *?thesis* **by** *simp*

**next**

**case** *b*

**hence**  $\text{real-of-int } \lceil \text{sqrt } (\text{real } n) \rceil^2 \leq \text{of-int } \lceil \text{sqrt } (\text{real } 4) \rceil^2$

**using** 2

**by** (*intro power-mono iffD2*[*OF of-int-le-iff*] *ceiling-mono iffD2*[*OF real-sqrt-le-iff*]) *auto*

**also have**  $\dots = 2 * \text{real } 2$  **by** *simp*

**also have**  $\dots \leq 2 * \text{real } n$

**using** *b* **by** (*intro mult-left-mono*) *auto*

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

**next**

**case** *c*

**hence**  $\text{real-of-int } \lceil \text{sqrt } (\text{real } n) \rceil^2 \leq \text{of-int } \lceil \text{sqrt } (\text{real } 9) \rceil^2$

**using** 2

**by** (*intro power-mono iffD2*[*OF of-int-le-iff*] *ceiling-mono iffD2*[*OF real-sqrt-le-iff*]) *auto*

**also have**  $\dots = 9$  **by** *simp*

**also have**  $\dots \leq 2 * \text{real } 5$  **by** *simp*

**also have**  $\dots \leq 2 * \text{real } n$

**using** *c* **by** (*intro mult-left-mono*) *auto*

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

**next**

**case** *d*

**have**  $\text{real-of-int } \lceil \text{sqrt } (\text{real } n) \rceil^2 \leq (\text{sqrt } (\text{real } n)+1)^2$

**using** 2 **by** (*intro power-mono*) *auto*

**also have**  $\dots = \text{real } n + \text{sqrt } (4 * \text{real } n + 0) + 1$

**using** *real-sqrt-pow2* **by** (*simp add:power2-eq-square algebra-simps real-sqrt-mult*)

**also have**  $\dots \leq \text{real } n + \text{sqrt } (4 * \text{real } n + (\text{real } n * (\text{real } n - 6) + 1)) + 1$   
**using**  $d$  **by**  $(\text{intro add-mono iffD2[OF real-sqrt-le-iff]}) \text{ auto}$   
**also have**  $\dots = \text{real } n + \text{sqrt } ((\text{real } n - 1)^2) + 1$   
**by**  $(\text{intro-cong } [\sigma_2 (+), \sigma_1 \text{sqrt}]) (\text{auto simp add:power2-eq-square algebra-simps})$   
**also have**  $\dots = 2 * \text{real } n$   
**using**  $d$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$   
**qed**

**have**  $\text{nat } \lceil \text{sqrt } (\text{real } n) \rceil^2 \in \{n..2*n\}$   
**by**  $(\text{simp add: approximation-preproc-nat(13) sqrt-le-D } 1)$   
**hence**  $\text{see-size } (\text{see-mgg } (\text{nat } \lceil \text{sqrt } (\text{real } n) \rceil)) \in \{n..2*n\}$   
**by**  $(\text{simp add:see-mgg-def})$   
**moreover have**  $\text{sqrt } (\text{real } n) > 0$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**hence**  $0 < \text{nat } \lceil \text{sqrt } (\text{real } n) \rceil$  **by**  $\text{simp}$   
**ultimately have**  $\text{is-expander } (\text{see-standard-aux } n) ((5 * \text{sqrt } 2 / 8) / 2 + 1 / 2)$   
**unfolding**  $\text{see-standard-aux-def}$  **by**  $(\text{intro see-compress see-mgg}) \text{ auto}$   
**thus**  $?A$   
**by**  $(\text{auto simp add:field-simps})$   
**show**  $?B$   
**unfolding**  $\text{see-standard-aux-def}$  **by**  $(\text{simp add:see-mgg-def})$   
**show**  $?C$   
**unfolding**  $\text{see-standard-aux-def}$  **by**  $\text{simp}$   
**qed**

**lemma**  $\text{nth-root-nat-eq}$ :

**assumes**  $k > 0$   
**shows**  $\text{nth-root-nat } k \ n = \text{nat } \lfloor \text{root } k \ n \rfloor$   
**proof**  $(\text{rule nth-root-nat-unique})$   
**have**  $\text{real } (\text{nat } \lfloor \text{root } k \ (\text{real } n) \rfloor^k) = \text{of-int } (\lfloor \text{root } k \ (\text{real } n) \rfloor)^k$   
**unfolding**  $\text{of-nat-power}$  **using**  $\text{real-root-ge-zero}$   
**by**  $(\text{subst of-nat-nat}) \text{ simp-all}$   
**also have**  $\dots \leq \text{root } k \ n^k$   
**using**  $\text{real-root-ge-zero}$  **by**  $(\text{intro power-mono}) \text{ auto}$   
**also have**  $\dots = n$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**finally show**  $\text{nat } \lfloor \text{root } k \ (\text{real } n) \rfloor^k \leq n$  **by**  $\text{simp}$

**next**

**have**  $\text{real } n = (\text{root } k \ (\text{real } n))^k$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**also have**  $\dots < (\text{of-int } \lfloor \text{root } k \ (\text{real } n) \rfloor + 1)^k$   
**using**  $\text{real-root-ge-zero}$   $\text{assms}$  **by**  $(\text{intro power-strict-mono}) \text{ auto}$   
**also have**  $\dots = \text{real } ((\text{nat } \lfloor \text{root } k \ (\text{real } n) \rfloor + 1))^k$   
**unfolding**  $\text{of-nat-power}$  **using**  $\text{real-root-ge-zero}$   
**by**  $(\text{subst of-nat-nat}) \text{ auto}$   
**also have**  $\dots = \text{real } ((\text{nat } \lfloor \text{root } k \ (\text{real } n) \rfloor + 1)^k)$   
**by**  $(\text{simp add: assms nat-add-distrib})$   
**finally show**  $n < (\text{nat } \lfloor \text{root } k \ (\text{real } n) \rfloor + 1)^k$  **by**  $\text{linarith}$

**qed**

**lemma**  $\text{nat-sqrt-code}$ :  $\text{nth-root-nat } 2 \ n + \text{of-bool } (\neg(\text{is-nth-power-nat } 2 \ n)) = \text{nat } \lceil \text{sqrt } n \rceil$   
 $(\text{is } ?L = ?R)$

**proof**  $(\text{cases is-nth-power-nat } 2 \ n)$

**case**  $\text{True}$

**then obtain**  $m$  **where**  $n\text{-def: } n = m^2$  **using**  $\text{is-nth-power-nat-def is-nth-powerE}$  **by**  $\text{metis}$   
**have**  $?L = \text{nth-root-nat } 2 \ n$  **using**  $\text{True}$  **by**  $\text{simp}$   
**also have**  $\dots = m$  **unfolding**  $n\text{-def}$  **by**  $(\text{intro nth-root-nat-nth-power}) \text{ auto}$   
**also have**  $\dots = ?R$  **unfolding**  $n\text{-def}$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$

**next**

```

case False
have 0: sqrt n ≥ 0 by simp

have False if sqrt n = of-int [sqrt n]
proof -
  define m where m = nat (of-int [sqrt n])
  have int m = of-int [sqrt n]
  unfolding m-def using 0 that by (subst of-nat-nat) (simp-all add:ceiling-altdef)
  hence sqrt n = real m using that by simp
  hence n = real m^2 by (metis of-nat-0-le-iff real-sqrt-pow2)
  hence n = m^2 by simp
  thus False using False unfolding is-nth-power-nat-def is-nth-power-def by metis
qed
hence f:sqrt n ≠ of-int [sqrt n] by blast

have ?L = nat [sqrt n] + 1 using False by (subst nth-root-nat-eq) (simp-all add:sqrt-def)
also have ... = ?R using f unfolding ceiling-altdef by (simp add: nat-add-distrib)
finally show ?thesis by simp
qed

lemma see-standard-aux-code [code]:
  see-standard-aux n = see-compress n (see-mgg (nth-root-nat 2 n + of-bool (¬is-nth-power-nat 2
n)))
  unfolding nat-sqrt-code see-standard-aux-def by simp

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 = 0.95nat [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]:

```

$see\_standard\_power\ x = (if\ x \leq 0 \vee x \geq 1\ then\ 0\ else\ (1 + see\_standard\_power\ (x/0.95)))$   
**proof** (cases  $x \leq 0 \vee x \geq 1$ )  
**case** *True*  
**have**  $\ln x / \ln (19 / 20) \leq 0$  **if**  $x > 0$   
**proof** –  
**have**  $x \geq 1$  **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**  $\dots = -\ln 0.95$   
**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**  $see\_standard\_power\ x = nat\ \lceil \ln x / \ln 0.95 \rceil$   
**using** *x-range unfolding see-standard-power-def* **by** *simp*  
**also have**  $\dots = nat\ \lceil \ln (x/0.95) / \ln 0.95 + 1 \rceil$   
**by** (*subst ln-divide-pos[OF x-range(1)]*) (*simp-all add:field-simps*)  
**also have**  $\dots = nat\ (\lceil \ln (x/0.95) / \ln 0.95 \rceil + 1)$   
**by** (*intro arg-cong[where f=nat]*) *simp*  
**also have**  $\dots = 1 + nat\ \lceil \ln (x/0.95) / \ln 0.95 \rceil$   
**using**  $0$  **by** (*subst nat-add-distrib*) *auto*  
**also have**  $\dots = (if\ x \leq 0 \vee 1 \leq x\ then\ 0\ else\ 1 + see\_standard\_power\ (x/0.95))$   
**unfolding** *see-standard-power-def* **using** *x-range* **by** *auto*  
**finally show** *?thesis* **by** *simp*  
**qed**  
  
**definition** *see-standard* ::  $nat \Rightarrow real \Rightarrow strongly\_explicit\_expander$   
**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*  $n\ \Lambda_a$ ) =  $16^{\wedge}(nat\ \lceil \ln \Lambda_a / \ln 0.95 \rceil)$  (**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*  $n\ \Lambda_a$ ) =  $16^{\wedge}(see\_standard\_power\ \Lambda_a)$   
**unfolding** *see-standard-def* **using** *see-standard-aux[OF assms(1)]* **by** *simp*

also have ... =  $16^{\wedge}(\text{nat } \lceil \ln \Lambda_a / \ln 0.95 \rceil)$   
 unfolding *see-standard-power-def* using *assms(2)* by *simp*  
 finally show ?C by *simp*  
 qed

**fun** *see-sample-walk* :: *strongly-explicit-expander*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *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*  $\equiv$  *see-size* *e* \* *see-degree* *e*  $\wedge$   
**shows** {# *see-sample-walk* *e* *l* *k*. *k*  $\in$  # *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*  $\wedge$

**have** 1: {*i* \* ?*d*..*(i + 1)* \* ?*d*}  $\cap$  {*j* \* ?*d*..*(j + 1)* \* ?*d*} = {} **if** *i*  $\neq$  *j* **for** *i j*  
**using** *that index-div-eq* by *blast*

**have** 2: *vertices-from* ?*G* *x* = {# *see-step* *e* *i* *x*. *i*  $\in$  # *mset-set* {..*?d*} #} (**is** ?*L* = ?*R*)  
**if** *x*  $\in$  *verts* ?*G* **for** *x*

**proof** –  
**have** *x* < ?*n*  
**using** *that* **unfolding** *graph-of-def* by *simp*  
**hence** 1: *out-arcs* ?*G* *x* = ( $\lambda$ *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*  $\in$  # *mset-set* (*out-arcs* ?*G* *x*) #}  
**unfolding** *verts-from-alt* **by** (*simp* *add:graph-of-def*)  
**also have** ... = {# *arc-head* *a*. *a*  $\in$  # {# *Arc* *x* (*see-step* *e* *i* *x*) *i*. *i*  $\in$  # *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* ( $\bigcup_{w < r. \{w * ?d .. (w + 1) * ?d\}}$ ) = ( $\sum_{w < r. \text{card } \{w * ?d .. (w + 1) * ?d\}}$ )  
**using** 1 **by** (*intro* *card-UN-disjoint*) *auto*  
**also have** ... = *r* \* ?*d* **by** *simp*  
**finally have** *card* ( $\bigcup_{w < r. \{w * ?d .. (w + 1) * ?d\}}$ ) = *card* {..*?d* \* *r*} **by** *simp*  
**moreover have** ?*d* + *z* \* ?*d*  $\leq$  ?*d* \* *r* **if** *z* < *r* **for** *z*

```

proof –
  have  $?d + z * ?d = ?d * (z + 1)$  by simp
  also have  $\dots \leq ?d * r$ 
    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  $\{\# ?w (l+1) k. k \in \# \text{mset-set } \{..<?n * ?d^{(l+1)}\} \# \} = \{\# ?w (l+1) k. k \in \# \text{mset-set } \{..<?d * r\} \# \}$ 
  unfolding r-def by (simp add:ac-simps)
also have  $\dots = \{\# ?w (l+1) x. x \in \# \text{mset-set } (\bigcup w < r. \{w * ?d..<(w + 1) * ?d\}) \# \}$ 
  unfolding 0 by simp
also have  $\dots = \text{image-mset } (?w (l+1)) (\text{concat-mset } (\text{image-mset } (\text{mset-set } \circ (\lambda w. \{w * ?d..<(w + 1) * ?d\})) (\text{mset-set } \{..<r\})))$ 
  by (intro arg-cong2 [where  $f = \text{image-mset}$ ] concat-disjoint-union-mset refl 1) auto
also have  $\dots = \text{concat-mset } \{\# \{ \# ?w (l+1) i. i \in \# \text{mset-set } \{w * ?d..<(w+1) * ?d\} \# \}. w \in \# \text{mset-set } \{..<r\} \# \}$ 
  by (simp add:image-concat-mset image-mset.compositionality comp-def del:see-sample-walk.simps)
also have  $\dots = \text{concat-mset } \{\# \{ \# ?w (l+1) i. i \in \# \text{mset-set } ((+)(w * ?d) \{..<?d\}) \# \}. w \in \# \text{mset-set } \{..<r\} \# \}$ 
  by (intro-cong [ $\sigma_1$  concat-mset,  $\sigma_2$  image-mset,  $\sigma_1$  mset-set] more:ext)
  (simp add: atLeast0LessThan[symmetric])
also have  $\dots = \text{concat-mset } \{\# \{ \# ?w (l+1) i. i \in \# \text{image-mset } ((+)(w * ?d)) (\text{mset-set } \{..<?d\}) \# \}. w \in \# \text{mset-set } \{..<r\} \# \}$ 
  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  $\dots = \text{concat-mset } \{\# \{ \# ?w (l+1) (w * ?d + i). i \in \# \text{mset-set } \{..<?d\} \# \}. w \in \# \text{mset-set } \{..<r\} \# \}$ 
  by (simp add:image-mset.compositionality comp-def del:see-sample-walk.simps)
also have  $\dots = \text{concat-mset } \{\# \{ \# ?w l w @ [\text{see-step } e \ i \ (\text{last } (?w \ l \ w))]. i \in \# \text{mset-set } \{..<?d\} \# \}. w \in \# \text{mset-set } \{..<r\} \# \}$ 
  by (intro-cong [ $\sigma_1$  concat-mset] more:image-mset-cong) (simp add:Let-def)
also have  $\dots = \text{concat-mset } \{\# \{ \# w @ [\text{see-step } e \ i \ (\text{last } w)]. i \in \# \text{mset-set } \{..<?d\} \# \}. w \in \# \text{walks}' \ ?G \ l \# \}$ 
  unfolding r-def Suc[symmetric] image-mset.compositionality comp-def by simp
also have  $\dots = \text{concat-mset } \{\# \{ \# w @ [x]. x \in \# \{ \# \text{see-step } e \ i \ (\text{last } w). i \in \# \text{mset-set } \{..<?d\} \# \} \# \}. w \in \# \text{walks}' \ ?G \ l \# \}$ 
  unfolding image-mset.compositionality comp-def by simp
also have  $\dots = \text{concat-mset } \{\# \{ \# w @ [x]. x \in \# \text{vertices-from } ?G \ (\text{last } w) \# \}. w \in \# \text{walks}' \ ?G \ l \# \}$ 
  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  $\dots = \text{walks}' (\text{graph-of } e) (l+1)$ 
  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

definition expander-pro ::
  nat  $\Rightarrow$  real  $\Rightarrow$  ('a,'b) pseudorandom-object-scheme  $\Rightarrow$  (nat  $\Rightarrow$  'a) pseudorandom-object
  where expander-pro l  $\Lambda$  S = (
    let e = see-standard (pro-size S)  $\Lambda$  in
    (| pro-last = see-size e * see-degree e  $\wedge^{(l-1)} - 1$ ,
      pro-select = ( $\lambda i j$ . pro-select S (see-sample-walk e (l-1) i ! j mod pro-size S)) |)
  )

open-bundle expander-pseudorandom-object-syntax
begin
notation expander-pro ( $\langle \mathcal{E} \rangle$ )
end

unbundle expander-pseudorandom-object-syntax

context
  fixes l :: nat
  fixes  $\Lambda$  :: real
  fixes P :: 'a pseudorandom-object
  assumes l-gt-0: l > 0
  assumes  $\Lambda$ -gt-0:  $\Lambda$  > 0
begin

private definition e where e = see-standard (pro-size P)  $\Lambda$ 

private lemma expander-pro-alt:
  expander-pro l  $\Lambda$  P =
    (| pro-last = see-size e * see-degree e  $\wedge^{(l-1)} - 1$ ,
      pro-select = ( $\lambda i j$ . pro-select P (see-sample-walk e (l-1) i ! j mod pro-size P)) |)
  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=P]  $\Lambda$ -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 ( $\mathcal{E}$  l  $\Lambda$  P) = see-size e * see-degree e  $\wedge^{(l-1)}$ 
  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  $\equiv$  map-pmf ( $\lambda xs$  i. pro-select P (xs ! i mod pro-size P))
    (pmf-of-multiset (walks (graph-of e) l))
  shows sample-pro (expander-pro l  $\Lambda$  P) = R
proof -
  let ?S = {..see-size e * see-degree e  $\wedge^{(l-1)}$ }

```

**let**  $?T = (\text{map-pmf } (\text{see-sample-walk } e \ (l-1)) \ (\text{pmf-of-set } ?S))$   
**have**  $0 \in ?S$   
**using**  $e\text{-size-gt-0 } e\text{-deg-gt-0}$  **by**  $\text{auto}$   
**hence**  $?S \neq \{\}$   
**by**  $\text{blast}$   
**hence**  $?T = \text{pmf-of-multiset } \{\#\text{see-sample-walk } e \ (l-1) \ i. \ i \in \#\ \text{mset-set } ?S\#\}$   
**by**  $(\text{subst map-pmf-of-set}) \ \text{simp-all}$   
**also have**  $\dots = \text{pmf-of-multiset } (\text{walks}' \ (\text{graph-of } e) \ (l-1))$   
**by**  $(\text{subst see-sample-walk}) \ \text{auto}$   
**also have**  $\dots = \text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) \ l)$   
**unfolding**  $\text{walks-def}$  **using**  $l\text{-gt-0}$  **by**  $(\text{cases } l, \ \text{simp-all})$   
**finally have**  $0: ?T = \text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) \ l)$   
**by**  $\text{simp}$

**have**  $\text{sample-pro } (\text{expander-pro } l \ \Lambda \ P) = \text{map-pmf } (\lambda x s \ j. \ \text{pro-select } P \ (x s ! j \ \text{mod } \text{pro-size } P))$   
 $?T$   
**unfolding**  $\text{expander-sample-size } \text{sample-pro-alt}$  **unfolding**  $\text{map-pmf-comp } \text{expander-pro-alt}$  **by**  $\text{simp}$   
**also have**  $\dots = R$  **unfolding**  $0 \ R\text{-def}$  **by**  $\text{simp}$   
**finally show**  $?thesis$  **by**  $\text{simp}$   
**qed**

**lemma**  $\text{expander-pro-range: pro-select } (\text{expander-pro } l \ \Lambda \ P) \ i \ j \in \text{pro-set } P$   
**unfolding**  $\text{expander-pro-alt}$  **by**  $(\text{simp add:pro-select-in-set})$

**lemma**  $\text{expander-uniform-property:}$   
**assumes**  $i < l$   
**shows**  $\text{map-pmf } (\lambda w. \ w \ i) \ (\text{sample-pro } (\text{expander-pro } l \ \Lambda \ P)) = \text{sample-pro } P \ (\text{is } ?L = ?R)$   
**proof**  $-$   
**have**  $?L = \text{map-pmf } (\lambda x. \ \text{pro-select } P \ (x \ \text{mod } \text{pro-size } P)) \ (\text{map-pmf } (\lambda x s. \ (x s ! i)) \ (\text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) \ l)))$   
**unfolding**  $\text{sample-pro-expander-walks}$  **by**  $(\text{simp add: map-pmf-comp})$   
**also have**  $\dots = \text{map-pmf } (\lambda x. \ \text{pro-select } P \ (x \ \text{mod } \text{pro-size } P)) \ (\text{pmf-of-set } (\text{verts } (\text{graph-of } e)))$   
**unfolding**  $E.\text{uniform-property}[OF \ \text{assms}]$  **by**  $\text{simp}$   
**also have**  $\dots = ?R$   
**using**  $\text{pro-size-gt-0}$  **unfolding**  $\text{sample-pro-alt}$   
**by**  $(\text{intro map-pmf-cong}) \ (\text{simp-all add:e-def graph-of-def see-standard select-def})$   
**finally show**  $?thesis$   
**by**  $\text{simp}$   
**qed**

**lemma**  $\text{expander-kl-chernoff-bound:}$   
**assumes**  $\text{measure } (\text{sample-pro } P) \ \{w. \ T \ w\} \leq \mu$   
**assumes**  $c \leq 1 \ \mu + \Lambda * (1-\mu) \leq c \ \mu \leq 1$   
**shows**  $\text{measure } (\text{sample-pro } (\text{expander-pro } l \ \Lambda \ P)) \ \{w. \ \text{real } (\text{card } \{i \in \{..\langle l\}. \ T \ (w \ i)\}) \geq c * l\} \leq \exp \ (- \ \text{real } l * \text{KL-div } c \ (\mu + \Lambda * (1-\mu))) \ (\text{is } ?L \leq ?R)$   
**proof**  $(\text{cases } \text{measure } (\text{sample-pro } P) \ \{w. \ T \ w\} > 0)$   
**case**  $\text{True}$   
**let**  $?w = \text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) \ l)$   
**define**  $V$  **where**  $V = \{v \in \text{verts } (\text{graph-of } e). \ T \ (\text{pro-select } P \ v)\}$   
**define**  $\nu$  **where**  $\nu = \text{measure } (\text{sample-pro } P) \ \{w. \ T \ w\}$

**have**  $\nu\text{-gt-0: } \nu > 0$  **unfolding**  $\nu\text{-def}$  **using**  $\text{True}$  **by**  $\text{simp}$   
**have**  $\nu\text{-le-1: } \nu \leq 1$  **unfolding**  $\nu\text{-def}$  **by**  $\text{simp}$   
**have**  $\nu\text{-le-}\mu: \nu \leq \mu$  **unfolding**  $\nu\text{-def}$  **using**  $\text{assms}(1)$  **by**  $\text{simp}$

**have**  $0: \text{card } \{i \in \{..\langle l\}. \ T \ (\text{pro-select } P \ (w ! i \ \text{mod } \text{pro-size } P))\} = \text{card } \{i \in \{..\langle l\}. \ w ! i \in$

$V\}$   
**if**  $w \in \text{set-pmf } (\text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) l))$  **for**  $w$   
**proof** –  
**have**  $a0: w \in \# \text{ walks } (\text{graph-of } e) l$  **using** *that*  $E.\text{walks-nonempty}$  **by** *simp*  
**have**  $a1: w ! i \in \text{verts } (\text{graph-of } e)$  **if**  $i < l$  **for**  $i$   
**using** *that*  $E.\text{set-walks-3}[OF a0]$  **by** *auto*  
**moreover** **have**  $w ! i \text{ mod } \text{pro-size } P = w ! i$  **if**  $i < l$  **for**  $i$   
**using**  $a1[OF \text{ that}] \text{ see-standard}(2)$   $e\text{-def}$  **by**  $(\text{simp add:graph-of-def})$   
**ultimately show** *?thesis*  
**unfolding**  $V\text{-def}$   
**by**  $(\text{intro arg-cong}[\text{where } f=\text{card}] \text{ restr-Collect-cong})$  *auto*  
**qed**

**have**  $1: E.\Lambda_a \leq \Lambda$   
**using**  $\text{see-standard}(1)$  **unfolding**  $\text{is-expander-def } e\text{-def}$  **by** *simp*

**have**  $2: V \subseteq \text{verts } (\text{graph-of } e)$   
**unfolding**  $V\text{-def}$  **by** *simp*

**have**  $\nu = \text{measure } (\text{pmf-of-set } \{..<\text{pro-size } P\}) (\{v. T (\text{pro-select } P v)\})$   
**unfolding**  $\nu\text{-def sample-pro-alt}$  **by** *simp*  
**also** **have**  $\dots = \text{real } (\text{card } (\{v \in \{..<\text{pro-size } P\}. T (\text{pro-select } P v)\})) / \text{real } (\text{pro-size } P)$   
**using**  $\text{pro-size-gt-0}$  **by**  $(\text{subst measure-pmf-of-set})$   $(\text{auto simp add:Int-def})$   
**also** **have**  $\dots = \text{real } (\text{card } V) / \text{card } (\text{verts } (\text{graph-of } e))$   
**unfolding**  $V\text{-def graph-of-def } e\text{-def}$  **using**  $\text{see-standard}$  **by**  $(\text{simp add:Int-commute})$   
**finally** **have**  $\nu\text{-eq: } \nu = \text{real } (\text{card } V) / \text{card } (\text{verts } (\text{graph-of } e))$   
**by** *simp*

**have**  $3: 0 < \nu + E.\Lambda_a * (1 - \nu)$   
**using**  $\nu\text{-le-1}$  **by**  $(\text{intro add-pos-nonneg } \nu\text{-gt-0 mult-nonneg-nonneg } E.\Lambda\text{-ge-0})$  *auto*

**have**  $\nu + E.\Lambda_a * (1 - \nu) = \nu * (1 - E.\Lambda_a) + E.\Lambda_a$  **by**  $(\text{simp add:algebra-simps})$   
**also** **have**  $\dots \leq \mu * (1 - E.\Lambda_a) + E.\Lambda_a$  **using**  $E.\Lambda\text{-le-1}$   
**by**  $(\text{intro add-mono mult-right-mono } \nu\text{-le-}\mu)$  *auto*  
**also** **have**  $\dots = \mu + E.\Lambda_a * (1 - \mu)$  **by**  $(\text{simp add:algebra-simps})$   
**also** **have**  $\dots \leq \mu + \Lambda * (1 - \mu)$  **using**  $\text{assms}(4)$  **by**  $(\text{intro add-mono mult-right-mono } 1)$  *auto*  
**finally** **have**  $4: \nu + E.\Lambda_a * (1 - \nu) \leq \mu + \Lambda * (1 - \mu)$  **by** *simp*

**have**  $5: \nu + E.\Lambda_a * (1 - \nu) \leq c$  **using**  $4 \text{ assms}(3)$  **by** *simp*

**have**  $?L = \text{measure } ?w \{y. c * \text{real } l \leq \text{real } (\text{card } \{i \in \{..<l\}. T (\text{pro-select } P (y ! i \text{ mod } \text{pro-size } P)\}))\}$   
**unfolding**  $\text{sample-pro-expander-walks}$  **by** *simp*  
**also** **have**  $\dots = \text{measure } ?w \{y. c * \text{real } l \leq \text{real } (\text{card } \{i \in \{..<l\}. y ! i \in V\})\}$   
**using**  $0$  **by**  $(\text{intro measure-pmf-cong})$   $(\text{simp})$   
**also** **have**  $\dots \leq \exp (- \text{real } l * \text{KL-div } c (\nu + E.\Lambda_a * (1 - \nu)))$   
**using**  $\text{assms}(2)$   $3$   $5$  **unfolding**  $\nu\text{-eq}$  **by**  $(\text{intro } E.\text{kl-bernoff-property } l\text{-gt-0 } 2)$  *auto*  
**also** **have**  $\dots \leq \exp (- \text{real } l * \text{KL-div } c (\mu + \Lambda * (1 - \mu)))$   
**using**  $l\text{-gt-0}$  **by**  $(\text{intro iffD2}[OF \text{ exp-le-cancel-iff}] \text{ iffD2}[OF \text{ mult-le-cancel-left-neg}] \text{KL-div-mono-right}[OF \text{ disjI2}] \text{ conjI } 3$   $4 \text{ assms}(2,3))$  *auto*  
**finally** **show** *?thesis* **by** *simp*

**next**  
**case** *False*  
**hence**  $0: \text{measure } (\text{sample-pro } P) \{w. T w\} = 0$  **using**  $\text{zero-less-measure-iff}$  **by** *blast*  
**hence**  $1: T w = \text{False}$  **if**  $w \in \text{pro-set } P$  **for**  $w$  **using** *that*  $\text{measure-pmf-posI}$  **by** *force*

**have**  $\mu + \Lambda * (1 - \mu) > 0$   
**proof**  $(\text{cases } \mu = 0)$

**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**  $c > 0$  **using** *assms(3)* **by** *auto*  
**hence**  $2:c*\text{real } l > 0$  **using** *l-gt-0* **by** *simp*  
  
**let**  $?w = \text{pmf-of-multiset } (\text{walks } (\text{graph-of } e) l)$   
  
**have**  $?L = \text{measure } ?w \{y. c*\text{real } l \leq \text{card } \{i \in \{..<l\}. T (\text{pro-select } P (y ! i \text{ mod } \text{pro-size } P))\}\}$   
**unfolding** *sample-pro-expander-walks* **by** *simp*  
**also have**  $\dots = 0$  **using** *pro-select-in-set 2* **by** (*subst 1*) *auto*  
**also have**  $\dots \leq ?R$  **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *expander-pro-size:*

$\text{pro-size } (\text{expander-pro } l \ \Lambda \ P) = \text{pro-size } P * (16 \wedge ((l-1) * \text{nat } \lceil \ln \ \Lambda / \ln (19 / 20) \rceil))$   
(is  $?L = ?R$ )

**proof** –

**have**  $?L = \text{see-size } e * \text{see-degree } e \wedge (l - 1)$   
**unfolding** *expander-sample-size* **by** *simp*  
**also have**  $\dots = \text{pro-size } P * (16 \wedge \text{nat } \lceil \ln \ \Lambda / \ln (19 / 20) \rceil) \wedge (l - 1)$   
**using** *see-standard* **unfolding** *e-def* **by** *simp*  
**also have**  $\dots = \text{pro-size } P * (16 \wedge ((l-1) * \text{nat } \lceil \ln \ \Lambda / \ln (19 / 20) \rceil))$   
**unfolding** *power-mult[symmetric]* **by** (*simp add:ac-simps*)  
**finally show** *?thesis*  
**by** *simp*

**qed**

**context**

**fixes**  $\gamma :: \text{real}$   
**assumes**  $\gamma$ -ge-0:  $\gamma \geq 0$

**begin**

**lemma** *expander-chernoff-bound-one-sided:*

**assumes** *AE*  $x$  *in* *sample-pro*  $P$ .  $f \ x \in \{0,1::\text{real}\}$   
**assumes**  $(\int x. f \ x \ \partial \text{sample-pro } P) \leq \mu$   
**shows**  $\text{measure } (\text{expander-pro } l \ \Lambda \ P) \{w. (\sum i < l. f (w \ i)) / l - \mu \geq \gamma + \Lambda\} \leq \exp (- 2 * \text{real } l * \gamma \wedge 2)$   
(is  $?L \leq ?R$ )

**proof** –

**let**  $?w = \text{sample-pro } (\text{expander-pro } l \ \Lambda \ P)$   
**define**  $T$  **where**  $T \ x = (f \ x = 1)$  **for**  $x$

**have**  $1: \text{indicator } \{w. T \ w\} \ x = f \ x$  **if**  $x \in \text{pro-set } P$  **for**  $x$

**proof** –

**have**  $f \ x \in \{0,1\}$  **using** *assms(1)* **that** **unfolding** *AE-measure-pmf-iff* **by** *simp*  
**thus** *?thesis* **unfolding**  $T$ -def **by** *auto*

**qed**

**have**  $\text{measure } P \{w. T \ w\} = (\int x. \text{indicator } \{w. T \ w\} \ x \ \partial P)$  **by** *simp*  
**also have**  $\dots = (\int x. f \ x \ \partial P)$  **using** 1 **by** (*intro integral-cong-AE AE-pmfI*) *auto*  
**also have**  $\dots \leq \mu$  **using** *assms(2)* **by** *simp*  
**finally have** 0:  $\text{measure } P \{w. T \ w\} \leq \mu$  **by** *simp*

hence  $\mu$ -ge-0:  $\mu \geq 0$  **using** *measure-nonneg order.trans* **by** *blast*

have *cases*:  $(\gamma=0 \implies p) \implies (\gamma+\Lambda+\mu > 1 \implies p) \implies (\gamma+\Lambda+\mu \leq 1 \wedge \gamma > 0 \implies p) \implies p$   
for  $p$

**using**  $\gamma$ -ge-0 **by** *argo*

have  $?L = \text{measure } ?w \{w. (\gamma+\Lambda+\mu)*l \leq (\sum i<l. f (w i))\}$

**using**  $l$ -gt-0 **by** (*intro measure-pmf-cong*) (*auto simp:field-simps*)

also have  $\dots = \text{measure } ?w \{w. (\gamma+\Lambda+\mu)*l \leq \text{card } \{i \in \{..<l\}. T (w i)\}\}$

**proof** (*rule measure-pmf-cong*)

**fix**  $\omega$

**assume**  $\omega \in \text{pro-set } (\text{expander-pro } l \ \Lambda \ P)$

**hence**  $\omega x \in \text{pro-set } P$  **for**  $x$  **using** *expander-pro-range set-sample-pro* **by** (*metis image-iff*)

**hence**  $(\sum i<l. f (\omega i)) = (\sum i<l. \text{indicator } \{w. T w\} (\omega i))$  **using** 1 **by** (*intro sum.cong*)

*auto*

also have  $\dots = \text{card } \{i \in \{..<l\}. T (\omega i)\}$  **unfolding** *indicator-def* **by** (*auto simp:Int-def*)

**finally** have  $(\sum i<l. f (\omega i)) = (\text{card } \{i \in \{..<l\}. T (\omega i)\})$  **by** *simp*

**thus**  $(\omega \in \{w. (\gamma+\Lambda+\mu)*l \leq (\sum i<l. f (w i))\}) = (\omega \in \{w. (\gamma+\Lambda+\mu)*l \leq \text{card } \{i \in \{..<l\}. T (w i)\}\})$

**by** *simp*

**qed**

also have  $\dots \leq ?R$  (**is**  $?L1 \leq -$ )

**proof** (*rule cases*)

**assume**  $\gamma = 0$  **thus** *?thesis* **by** *simp*

**next**

**assume**  $a: \gamma + \Lambda + \mu \leq 1 \wedge 0 < \gamma$

**hence**  $\mu$ -lt-1:  $\mu < 1$  **using**  $\gamma$ -ge-0  $\Lambda$ -gt-0 **by** *simp*

**hence**  $\mu$ -le-1:  $\mu \leq 1$  **by** *simp*

**have**  $\mu + \Lambda * (1 - \mu) \leq \mu + \Lambda * 1$  **using**  $\mu$ -ge-0  $\Lambda$ -gt-0 **by** (*intro add-mono mult-left-mono*)

*auto*

also have  $\dots < \gamma + \Lambda + \mu$  **using**  $\gamma$ -ge-0  $a$  **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**  $\mu$ -lt-1

**by** (*intro add-nonneg-pos*)  $\mu$ -ge-0 *mult-pos-pos*  $\Lambda$ -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**  $?L1 \leq \text{exp } (- \text{real } l * \text{KL-div } (\gamma+\Lambda+\mu) (\mu + \Lambda*(1-\mu)))$

**using**  $a$   $b$  **by** (*intro expander-kl-bernoff-bound*)  $\mu$ -le-1 0) *auto*

also have  $\dots \leq \text{exp } (- \text{real } l * (2 * ((\gamma+\Lambda+\mu) - (\mu + \Lambda*(1-\mu)))^2))$

**by** (*intro iffD2[OF exp-le-cancel-iff]*) *mult-left-mono-neg* *KL-div-lower-bound*  $c$   $d$ ) *simp*

also have  $\dots \leq \text{exp } (- \text{real } l * (2 * (\gamma^2)))$

**using**  $\gamma$ -ge-0  $\mu$ -lt-1  $\Lambda$ -gt-0  $\mu$ -ge-0

**by** (*intro iffD2[OF exp-le-cancel-iff]*) *mult-left-mono-neg*[**where**  $c=-\text{real } l$ ] *mult-left-mono* *power-mono*) *simp-all*

also have  $\dots = ?R$  **by** *simp*

**finally** show  $?L1 \leq ?R$  **by** *simp*

**next**

**assume**  $a: 1 < \gamma + \Lambda + \mu$

**have**  $(\gamma+\Lambda+\mu)* \text{real } l > \text{real } (\text{card } \{i \in \{..<l\}. (x i)\})$  **for**  $x$

**proof** -

**have**  $\text{real } (\text{card } \{i \in \{..<l\}. (x i)\}) \leq \text{card } \{..<l\}$  **by** (*intro of-nat-mono card-mono*) *auto*

**also** have  $\dots = \text{real } l$  **by** *simp*

**also** have  $\dots < (\gamma+\Lambda+\mu)* \text{real } l$  **using**  $l$ -gt-0  $a$  **by** *simp*

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

**qed**

**hence**  $?L1 = 0$  **unfolding** *not-le[symmetric]* **by** *auto*

also have  $\dots \leq ?R$  **by** *simp*

finally show  $?L1 \leq ?R$  by *simp*  
 qed  
 finally show *?thesis* by *simp*  
 qed

lemma *expander-chernoff-bound-1*:

assumes *AE x in sample-pro P. f x ∈ {0,1::real}*  
 defines  $\mu \equiv (\int x. f x \partial \text{sample-pro } P)$   
 shows *measure (E l Λ P) {w. |(\sum i<l. f (w i))/l-μ|≥γ+Λ} ≤ 2\*exp (- 2 \* real l \* γ^2)*  
 (is  $?L \leq ?R$ )

proof –

let  $?w = \text{sample-pro } (\text{expander-pro } l \ \Lambda \ P)$   
 have  $?L \leq \text{measure } ?w \{w. (\sum i<l. f (w i))/l-\mu \geq \gamma+\Lambda\} + \text{measure } ?w \{w. (\sum i<l. f (w i))/l-\mu \leq -(\gamma+\Lambda)\}$   
 by (*intro pmf-add*) *auto*  
 also have  $\dots \leq \text{exp } (-2*\text{real } l*\gamma^2) + \text{measure } ?w \{w. -((\sum i<l. f (w i))/l-\mu) \geq (\gamma+\Lambda)\}$   
 using *assms* by (*intro add-mono expander-chernoff-bound-one-sided*) (*auto simp: algebra-simps*)  
 also have  $\dots \leq \text{exp } (-2*\text{real } l*\gamma^2) + \text{measure } ?w \{w. ((\sum i<l. 1-f (w i))/l-(1-\mu)) \geq (\gamma+\Lambda)\}$   
 using *l-gt-0* by (*auto simp: sum-subtractf field-simps*)  
 also have  $\dots \leq \text{exp } (-2*\text{real } l*\gamma^2) + \text{exp } (-2*\text{real } l*\gamma^2)$   
 using *assms* by (*intro add-mono expander-chernoff-bound-one-sided*) *auto*  
 also have  $\dots = ?R$  by *simp*  
 finally show *?thesis* by *simp*

qed

lemma *expander-chernoff-bound*:

assumes *AE x in sample-pro P. f x ∈ {0,1::real}*  
 defines  $\mu \equiv \text{measure-pmf.expectation } P \ f$   
 shows *P(ω in E l Λ P. |(\sum i<l. f (ω i))/l-μ|≥γ+Λ) ≤ 2\*exp (- 2 \* real l \* γ^2)*  
 (is  $?L \leq ?R$ )

proof –

have  $?L = \text{measure } (\text{expander-pro } l \ \Lambda \ P) \{w. |(\sum i<l. f (w i))/l-\mu| \geq \gamma+\Lambda\}$  by *simp*  
 also have  $\dots \leq ?R$  **unfolding**  $\mu\text{-def}$  by (*intro expander-chernoff-bound-1 assms(1)*)  
 finally show *?thesis* by *simp*

qed

lemma *expander-chernoff-bound-2*:

assumes *AE x in sample-pro P. f x ∈ {0,1::real}*  
 defines  $\mu \equiv \text{measure-pmf.expectation } P \ f$   
 shows *P(ω in sample-pro (E l Λ P). |(\sum i<l. f (ω i))-real l\*μ|≥real l\*(γ+Λ)) ≤ 2\*exp (- 2 \* real l \* γ^2)*  
 (is  $?L \leq ?R$ )

proof –

have  $?L = \text{measure } (\text{expander-pro } l \ \Lambda \ P) \{w. |(\sum i<l. f (w i))/l-\mu| \geq \gamma+\Lambda\}$   
 using *l-gt-0* by (*intro arg-cong2[where f=measure] refl Collect-cong*) (*simp add: field-simps*)  
 also have  $\dots \leq ?R$  **unfolding**  $\mu\text{-def}$  by (*intro expander-chernoff-bound-1 assms(1)*)  
 finally show *?thesis* by *simp*

qed

end

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](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](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](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.