Expander Graphs

Emin Karayel

March 17, 2025

Abstract

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

Contents

1 Introduction

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

- One-sided spectral expansion – an upper-bound on the second largest eigenvalue λ_2 of the adjacency matrix,
- Two-sided spectral expansion – an upper-bound on the absolute value of both λ_2 and λ_n the smallest eigenvalue,
- Edge expansion – a lower-bound on the relative count of edges between any subset and its complement.

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

This entry formalizes

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

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

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

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

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

2 Preliminary Results

2.1 Constructive Chernoff Bound

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

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

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

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

assumes $a \le b$ **and** $x > 0$ $x \le 1$
shows x *powr* $b \le x$ *powr* a
⟨*proof*⟩

lemma *exp-powr*: $(exp\ x)\ powr\ y = exp\ (x*y)$ **for** $x :: real$
⟨*proof*⟩

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

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

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

definition *KL-div* :: $real \Rightarrow real \Rightarrow real$
where $KL\text{-}div\ p\ q = p * ln\ (p/q) + (1-p) * ln\ ((1-p)/(1-q))$

theorem *impagliazzo-kabanets-pmf*:
fixes $Y :: nat \Rightarrow 'a \Rightarrow bool$
fixes $p :: 'a\ pmf$
assumes $n > 0$
assumes $\bigwedge i.\ i \in \{..<n\} \Longrightarrow \delta\ i \in \{0..1\}$
assumes $\bigwedge S.\ S \subseteq \{..<n\} \Longrightarrow measure\ p\ \{\omega.\ (\forall\, i \in S.\ Y\ i\ \omega)\} \le (\prod i \in S.\ \delta\ i)$
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 $measure\ p\ \{\omega.\ real\ (card\ \{i \in \{..<n\}.\ Y\ i\ \omega\}) \ge \gamma * n\} \le exp\ (-real\ n * KL\text{-}div\ \gamma$
$\delta\text{-}avg)$
(**is** $?L \le ?R$)
⟨*proof*⟩

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

lemma (**in** *prob-space*) *establish-pmf*:
fixes $f :: 'a \Rightarrow 'b$
assumes *rv*: *random-variable discrete f*
assumes *countable* (f ‘ *space M*)
shows $distr\ M\ discrete\ f \in \{M.\ prob\text{-}space\ M \wedge sets\ M = UNIV \wedge (AE\ x\ in\ M.\ measure\ M$
$\{x\} \neq 0)\}$
⟨*proof*⟩

lemma *singletons-image-eq*:
$(\lambda x.\ \{x\})$ ‘ $T \subseteq Pow\ T$
⟨*proof*⟩

theorem (**in** *prob-space*) *impagliazzo-kabanets*:
fixes $Y :: nat \Rightarrow 'a \Rightarrow bool$
assumes $n > 0$

assumes $\bigwedge i.\ i \in \{..<n\} \Longrightarrow$ *random-variable discrete* $(Y\ i)$
assumes $\bigwedge i.\ i \in \{..<n\} \Longrightarrow \delta\ i \in \{0..1\}$
assumes $\bigwedge S.\ S \subseteq \{..<n\} \Longrightarrow \mathcal{P}(\omega\ in\ M.\ (\forall i \in S.\ Y\ i\ \omega)) \leq (\prod i \in S.\ \delta\ i)$
defines $\delta\text{-}avg \equiv (\sum i \in \{..<n\}.\ \delta\ i)/n$
assumes $\gamma \in \{\delta\text{-}avg..1\}$ $\delta\text{-}avg > 0$
shows $\mathcal{P}(\omega\ in\ M.\ real\ (card\ \{i \in \{..<n\}.\ Y\ i\ \omega\}) \geq \gamma * n) \leq exp\ (-real\ n * KL\text{-}div\ \gamma\ \delta\text{-}avg)$
(**is** *?L* $\leq$ *?R*)
⟨*proof*⟩

Bounds and properties of *KL-div*

lemma *KL-div-mono-right-aux-1*:
assumes $0 \leq p$ $p \leq q$ $q \leq q'$ $q' < 1$
shows $KL\text{-}div\ p\ q - 2*(p-q)\hat{}2 \leq KL\text{-}div\ p\ q' - 2*(p-q')\hat{}2$
⟨*proof*⟩

lemma *KL-div-swap*: $KL\text{-}div\ (1-p)\ (1-q) = KL\text{-}div\ p\ q$
⟨*proof*⟩

lemma *KL-div-mono-right-aux-2*:
assumes $0 < q'$ $q' \leq q$ $q \leq p$ $p \leq 1$
shows $KL\text{-}div\ p\ q - 2*(p-q)\hat{}2 \leq KL\text{-}div\ p\ q' - 2*(p-q')\hat{}2$
⟨*proof*⟩

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 $KL\text{-}div\ p\ q - 2*(p-q)\hat{}2 \leq KL\text{-}div\ p\ q' - 2*(p-q')\hat{}2$
⟨*proof*⟩

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*⟩

lemma *KL-div-lower-bound*:
assumes $p \in \{0..1\}$ $q \in \{0<..<1\}$
shows $2*(p-q)\hat{}2 \leq KL\text{-}div\ p\ q$
⟨*proof*⟩

end

2.2 Congruence Method

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

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

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

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

where *cong-tag-3 x = CongTag*

lemma *arg-cong3*:
assumes *x1 = x2 y1 = y2 z1 = z2*
shows *f x1 y1 z1 = f x2 y2 z2*
⟨*proof*⟩

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

bundle *intro-cong-syntax*
begin
notation *cong-tag-1* (‹σ_1›)
notation *cong-tag-2* (‹σ_2›)
notation *cong-tag-3* (‹σ_3›)
end

lemma *restr-Collect-cong*:
assumes $\bigwedge x.\ x \in A \Longrightarrow P\ x = Q\ x$
shows $\{x \in A.\ P\ x\} = \{x \in A.\ Q\ x\}$
⟨*proof*⟩

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 ⋀*n M x. P M* ⟹ ¬(*x* ∈# *M*) ⟹ *n* > *0* ⟹ *P* (*M* + *replicate-mset n x*)
shows *P M*
⟨*proof*⟩

lemma *sum-mset-conv*:
fixes *f* :: *'a ⇒ 'b*::{*semiring-1*}
shows *sum-mset* (*image-mset f A*) = *sum* (λ*x. of-nat* (*count A x*) ∗ *f x*) (*set-mset A*)
⟨*proof*⟩

lemma *sum-mset-conv-2*:

fixes *f* :: *'a* ⇒ *'b*::{*semiring-1*}
assumes *set-mset A* ⊆ *B finite B*
shows *sum-mset* (*image-mset f A*) = *sum* (λ*x*. *of-nat* (*count A x*) ∗ *f x*) *B* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *count-mset-exp*: *count A x* = *size* (*filter-mset* (λ*y*. *y* = *x*) *A*)
⟨*proof*⟩

lemma *mset-repl*: *mset* (*replicate k x*) = *replicate-mset k x*
⟨*proof*⟩

lemma *count-image-mset-inj*:
assumes *inj f*
shows *count* (*image-mset f A*) (*f x*) = *count A x*
⟨*proof*⟩

lemma *count-image-mset-0-triv*:
assumes *x* ∉ *range f*
shows *count* (*image-mset f A*) *x* = *0*
⟨*proof*⟩

lemma *filter-mset-ex-predicates*:
assumes ⋀*x*. ¬ *P x* ∨ ¬ *Q x*
shows *filter-mset P M* + *filter-mset Q M* = *filter-mset* (λ*x*. *P x* ∨ *Q x*) *M*
⟨*proof*⟩

lemma *sum-count-2*:
assumes *finite F*
shows *sum* (*count M*) *F* = *size* (*filter-mset* (λ*x*. *x* ∈ *F*) *M*)
⟨*proof*⟩

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

lemma *image-concat-mset*:
image-mset f (*concat-mset xss*) = *concat-mset* (*image-mset* (*image-mset f*) *xss*)
⟨*proof*⟩

lemma *concat-add-mset*:
concat-mset (*image-mset* (λ*x*. *f x* + *g x*) *xs*) = *concat-mset* (*image-mset f xs*) + *concat-mset* (*image-mset g xs*)
⟨*proof*⟩

lemma *concat-add-mset-2*:
concat-mset (*xs* + *ys*) = *concat-mset xs* + *concat-mset ys*
⟨*proof*⟩

lemma *size-concat-mset*:
size (*concat-mset xss*) = *sum-mset* (*image-mset size xss*)
⟨*proof*⟩

lemma *filter-concat-mset*:
filter-mset P (*concat-mset xss*) = *concat-mset* (*image-mset* (*filter-mset P*) *xss*)
⟨*proof*⟩

lemma *count-concat-mset*:
count (*concat-mset xss*) *xs* = *sum-mset* (*image-mset* (λ*x*. *count x xs*) *xss*)
⟨*proof*⟩

lemma *set-mset-concat-mset*:
set-mset (*concat-mset xss*) = ⋃ (*set-mset* ‘ (*set-mset xss*))
⟨*proof*⟩

lemma *concat-mset-empty*: *concat-mset* {#} = {#}
⟨*proof*⟩

lemma *concat-mset-single*: *concat-mset* {#*x*#} = *x*
⟨*proof*⟩

lemma *concat-disjoint-union-mset*:
assumes *finite I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *finite* (*A i*)
assumes ⋀*i j*. *i* ∈ *I* ⟹ *j* ∈ *I* ⟹ *i* ≠ *j* ⟹ *A i* ∩ *A j* = {}
shows *mset-set* (⋃ (*A* ‘ *I*)) = *concat-mset* (*image-mset* (*mset-set* ∘ *A*) (*mset-set I*))
⟨*proof*⟩

lemma *size-filter-mset-conv*:
size (*filter-mset f A*) = *sum-mset* (*image-mset* (λ*x*. *of-bool* (*f x*) :: *nat*) *A*)
⟨*proof*⟩

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

lemma *repeat-image-concat-mset*:
repeat-mset n (*image-mset f A*) = *concat-mset* (*image-mset* (λ*x*. *replicate-mset n* (*f x*)) *A*)
⟨*proof*⟩

lemma *mset-prod-eq*:
assumes *finite A finite B*
shows
mset-set (*A* × *B*) = *concat-mset* {# {# (*x*,*y*). *y* ∈# *mset-set B* #} .*x* ∈# *mset-set A* #}
⟨*proof*⟩

lemma *sum-mset-repeat*:
fixes *f* :: *'a* ⇒ *'b* :: {*comm-monoid-add*,*semiring-1*}
shows *sum-mset* (*image-mset f* (*repeat-mset n A*)) = *of-nat n* ∗ *sum-mset* (*image-mset f A*)
⟨*proof*⟩

unbundle *no intro-cong-syntax*

end

3 Definitions

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

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

unbundle *intro-cong-syntax*

definition *arcs-betw* **where** *arcs-betw G u v* = {*a*. *a* ∈ *arcs G* ∧ *head G a* = *v* ∧ *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 ∈ *verts G*.

definition *symmetric-multi-graph* **where** *symmetric-multi-graph G* =
  (*fin-digraph G* ∧ (∀ *v w*. {*v*, *w*} ⊆ *verts G* ⟶ *card* (*arcs-betw G w v*) = *card* (*arcs-betw G v w*)))

lemma *symmetric-multi-graphI*:
  assumes *fin-digraph G*
  assumes *bij-betw f* (*arcs G*) (*arcs G*)
  assumes ⋀*e*. *e* ∈ *arcs G* ⟹ *head G* (*f e*) = *tail G e* ∧ *tail G* (*f e*) = *head G e*
  shows *symmetric-multi-graph G*
⟨*proof*⟩

lemma *symmetric-multi-graphD2*:
  assumes *symmetric-multi-graph G*
  shows *fin-digraph G*
  ⟨*proof*⟩

lemma *symmetric-multi-graphD*:
  assumes *symmetric-multi-graph G*
  shows *card* {*e* ∈ *arcs G*. *head G e*=*v* ∧ *tail G e*=*w*} = *card* {*e* ∈ *arcs G*. *head G e*=*w* ∧ *tail G e*=*v*}
    (**is** *card ?L* = *card ?R*)
⟨*proof*⟩

lemma *symmetric-multi-graphD3*:
  assumes *symmetric-multi-graph G*
  shows
    card {*e*∈*arcs G*. *tail G e*=*v* ∧ *head G e*=*w*}=*card* {*e*∈*arcs G*. *tail G e*=*w*∧*head G e*=*v*}
  ⟨*proof*⟩

lemma *symmetric-multi-graphD4*:
  assumes *symmetric-multi-graph G*
  shows *card* (*arcs-betw G v w*) = *card* (*arcs-betw G w v*)
  ⟨*proof*⟩

lemma *symmetric-degree-eq*:
  assumes *symmetric-multi-graph G*
  assumes *v* ∈ *verts G*
  shows *out-degree G v* = *in-degree G v* (**is** *?L* = *?R*)
⟨*proof*⟩

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

lemma (**in** *fin-digraph*) *count-edges*:
  count (*edges G*) (*u*,*v*) = *card* (*arcs-betw G u v*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *count-edges-sym*:
  assumes *symmetric-multi-graph G*
  shows *count* (*edges G*) (*v*, *w*) = *count* (*edges G*) (*w*, *v*)
  ⟨*proof*⟩

lemma (**in** *fin-digraph*) *edges-sym*:
  assumes *symmetric-multi-graph G*
  shows {# (*y*,*x*). (*x*,*y*) ∈# (*edges G*) #} = *edges G*
⟨*proof*⟩

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

context *fin-digraph*
begin

lemma *edge-set*:
  assumes *x* ∈# *edges G*
  shows *fst x* ∈ *verts G snd x* ∈ *verts G*
  ⟨*proof*⟩

lemma *verts-from-alt*:
  vertices-from G v = *image-mset* (*head G*) (*mset-set* (*out-arcs G v*))
⟨*proof*⟩

lemma *verts-to-alt*:
  vertices-to G v = *image-mset* (*tail G*) (*mset-set* (*in-arcs G v*))
⟨*proof*⟩

lemma *set-mset-vertices-from*:
  set-mset (*vertices-from G x*) ⊆ *verts G*
  ⟨*proof*⟩

lemma *set-mset-vertices-to*:
  set-mset (*vertices-to G x*) ⊆ *verts G*
  ⟨*proof*⟩

end

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

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

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

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

lemma *reg*:
  assumes *v* ∈ *verts G*
  shows *out-degree G v* = *d in-degree G v* = *d*
⟨*proof*⟩

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

lemma *n-gt-0*: *n* > *0*
  ⟨*proof*⟩

lemma *d-gt-0*: $d > 0$
⟨*proof*⟩

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

lemma *conjugate-divide*[*simp*]:
  fixes $x\ y :: 'c :: conjugatable\text{-}field$
  shows $conjugate\ (x\ /\ y) = conjugate\ x\ /\ conjugate\ y$
⟨*proof*⟩

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

definition $g\text{-}norm\ f = sqrt\ (g\text{-}inner\ f\ f)$

lemma *g-norm-eq*: $g\text{-}norm\ f = L2\text{-}set\ f\ (verts\ G)$
  ⟨*proof*⟩

lemma *g-inner-cauchy-schwartz*:
  fixes $f\ g :: 'a \Rightarrow real$
  shows $|g\text{-}inner\ f\ g| \le g\text{-}norm\ f * g\text{-}norm\ g$
⟨*proof*⟩

lemma *g-inner-cong*:
  assumes $\bigwedge x.\ x \in verts\ G \Longrightarrow f1\ x = f2\ x$
  assumes $\bigwedge x.\ x \in verts\ G \Longrightarrow g1\ x = g2\ x$
  shows $g\text{-}inner\ f1\ g1 = g\text{-}inner\ f2\ g2$
  ⟨*proof*⟩

lemma *g-norm-cong*:
  assumes $\bigwedge x.\ x \in verts\ G \Longrightarrow f\ x = g\ x$
  shows $g\text{-}norm\ f = g\text{-}norm\ g$
  ⟨*proof*⟩

lemma *g-norm-nonneg*: $g\text{-}norm\ f \ge 0$
  ⟨*proof*⟩

lemma *g-norm-sq*:
  $g\text{-}norm\ f\hat{\ }2 = g\text{-}inner\ f\ f$
  ⟨*proof*⟩

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

lemma *g-step-simps*:
  $g\text{-}step\ (\lambda x.\ f\ x + g\ x)\ y = g\text{-}step\ f\ y + g\text{-}step\ g\ y$
  $g\text{-}step\ (\lambda x.\ f\ x\ /\ c)\ y = g\text{-}step\ f\ y\ /\ c$
  ⟨*proof*⟩

lemma *g-inner-step-eq*:
  g-inner f (*g-step f*) = ($\sum$ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*)) / *d* (**is** *?L* = *?R*)
⟨*proof*⟩

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

lemma Λ-*test-ne*:
  assumes *n* > *1*
  shows Λ-*test* ≠ {}
⟨*proof*⟩

lemma Λ-*test-empty*:
  assumes *n* = *1*
  shows Λ-*test* = {}
⟨*proof*⟩

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

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

definition Λ_2 :: *real*
  where Λ_2 = (*if n* > *1 then* (*SUP f* ∈ Λ-*test*. *g-inner f* (*g-step f*)/*g-inner f f*) *else 0*)

definition Λ_a :: *real*
  where Λ_a = (*if n* > *1 then* (*SUP f* ∈ Λ-*test*. |*g-inner f* (*g-step f*)|/*g-inner f f*) *else 0*)

lemma *sum-arcs-tail*:
  fixes *f* :: *'a* ⇒ (*'c* :: *semiring-1*)
  shows ($\sum$ *a* ∈ *arcs G*. *f* (*tail G a*)) = *of-nat d* ∗ ($\sum$ *v* ∈ *verts G*. *f v*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *sum-arcs-head*:
  fixes *f* :: *'a* ⇒ (*'c* :: *semiring-1*)
  shows ($\sum$ *a* ∈ *arcs G*. *f* (*head G a*)) = *of-nat d* ∗ ($\sum$ *v* ∈ *verts G*. *f v*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *bdd-above-aux*:
  |$\sum$ *a*∈*arcs G*. *f*(*head G a*)∗*f*(*tail G a*)| ≤ *d*∗ *g-norm f*^*2* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *bdd-above-aux-2*:
  assumes *f* ∈ Λ-*test*
  shows |*g-inner f* (*g-step f*)| / *g-inner f f* ≤ *1*
⟨*proof*⟩

lemma *bdd-above-aux-3*:
  assumes *f* ∈ Λ-*test*
  shows *g-inner f* (*g-step f*) / *g-inner f f* ≤ *1* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

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

lemma *bdd-above-*Λ_2: *bdd-above* $((\lambda f.\ g\text{-}inner\ f\ (g\text{-}step\ f)\ /\ g\text{-}inner\ f\ f)$ ‘ *Λ-test*)
⟨*proof*⟩

lemma *Λ-le-1*: $\Lambda_a \le 1$
⟨*proof*⟩

lemma Λ_2*-le-1*: $\Lambda_2 \le 1$
⟨*proof*⟩

lemma *Λ-ge-0*: $\Lambda_a \ge 0$
⟨*proof*⟩

lemma *os-expanderI*:
assumes $n > 1$
assumes $\bigwedge f.\ g\text{-}inner\ f\ (\lambda\text{-}.\ 1){=}0 \Longrightarrow g\text{-}inner\ f\ (g\text{-}step\ f) \le C{*}g\text{-}norm\ f\hat{\ }2$
shows $\Lambda_2 \le C$
⟨*proof*⟩

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

lemma *expander-intro-1*:
assumes $C \ge 0$
assumes $\bigwedge f.\ g\text{-}inner\ f\ (\lambda\text{-}.\ 1){=}0 \Longrightarrow |g\text{-}inner\ f\ (g\text{-}step\ f)| \le C{*}g\text{-}norm\ f\hat{\ }2$
shows $\Lambda_a \le C$
⟨*proof*⟩

lemma *expander-intro*:
assumes $C \ge 0$
assumes $\bigwedge f.\ g\text{-}inner\ f\ (\lambda\text{-}.\ 1){=}0 \Longrightarrow |\sum a \in arcs\ G.\ f(head\ G\ a) * f(tail\ G\ a)| \le C{*}g\text{-}norm$ $f\hat{\ }2$
shows $\Lambda_a \le C/d$
⟨*proof*⟩

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

lemma *expansionD*:
assumes $g\text{-}inner\ f\ (\lambda\text{-}.\ 1) = 0$
shows $|\sum a \in arcs\ G.\ f\ (head\ G\ a) * f\ (tail\ G\ a)| \le d * \Lambda_a * g\text{-}norm\ f\hat{\ }2$ (**is** $?L \le ?R$)
⟨*proof*⟩

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

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

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

lemma *edge-expansionD*:

assumes $S \subseteq$ *verts G 2∗card S* $\leq n$
shows $\Lambda_e * $ *card S* $\leq$ *real* (*card* (*edges-betw S* ($-S$)))
⟨*proof*⟩

lemma *edge-expansionI*:
fixes α :: *real*
assumes $n > 1$
assumes $\bigwedge S.\ S \subseteq$ *verts G* $\Longrightarrow$ *2∗card S* $\leq n \Longrightarrow S \neq \{\} \Longrightarrow$ *card* (*edges-betw S* ($-S$)) $\geq \alpha *$ *card S*
shows $\Lambda_e \geq \alpha$
⟨*proof*⟩

end

lemma *regular-graphI*:
assumes *symmetric-multi-graph G*
assumes *verts G* $\neq \{\}$ $d > 0$
assumes $\bigwedge v.\ v \in$ *verts G* $\Longrightarrow$ *out-degree G v* $= d$
shows *regular-graph G*
⟨*proof*⟩

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*⟩

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.*Λ_a $H \leq \Lambda_a$
and *regular-graph-iso-os-expansion-le*: *regular-graph.*Λ_2 $H \leq \Lambda_2$
and *regular-graph-iso-edge-expansion-ge*: *regular-graph.*Λ_e $H \geq \Lambda_e$
⟨*proof*⟩

lemma (**in** *regular-graph*)
assumes *digraph-iso G H*
shows *regular-graph-iso-expansion*: *regular-graph.*Λ_a $H = \Lambda_a$
and *regular-graph-iso-os-expansion*: *regular-graph.*Λ_2 $H = \Lambda_2$
and *regular-graph-iso-edge-expansion*: *regular-graph.*Λ_e $H = \Lambda_e$
⟨*proof*⟩

unbundle *no intro-cong-syntax*

end

4 Setup for Types to Sets

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

begin

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

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

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

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

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

sublocale *type-definition enum-verts enum-verts-inv verts G*
⟨*proof*⟩

lemma *enum-verts*: *bij-betw enum-verts UNIV* (*verts G*)
  ⟨*proof*⟩

The stochastic matrix associated to the graph.

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

lemma *card-n*: *CARD*('*n*) = *n*
  ⟨*proof*⟩

lemma *symmetric-A*: *transpose A* = *A*
⟨*proof*⟩

lemma *g-step-conv*:
  (χ *i*. *g-step f* (*enum-verts i*)) = *A* *v (χ *i*. *f* (*enum-verts i*))
⟨*proof*⟩

lemma *g-inner-conv*:
  g-inner f g = (χ *i*. *f* (*enum-verts i*)) · (χ *i*. *g* (*enum-verts i*))
  ⟨*proof*⟩

lemma *g-norm-conv*:
  g-norm f = *norm* (χ *i*. *f* (*enum-verts i*))
⟨*proof*⟩

end

lemma *eg-tts-1*:
  assumes *regular-graph G*
  assumes ∃(*f*::('*n*::*finite*) ⇒ '*a*) *g*. *type-definition f g* (*verts G*)
  shows *regular-graph-tts TYPE*('*n*) *G*
  ⟨*proof*⟩

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*⟩

lemma *remove-finite-premise*:
  (*class.finite TYPE*(*'n*) $\Longrightarrow$ $\exists$(*Rep* :: *'n* $\Rightarrow$ *'a*) *Abs. type-definition Rep Abs* (*verts G*) $\Longrightarrow$ *PROP Q*)
  $\equiv$ ($\exists$(*Rep* :: *'n* $\Rightarrow$ *'a*) *Abs. type-definition Rep Abs* (*verts G*) $\Longrightarrow$ *PROP Q*)
  (**is** *?L* $\equiv$ *?R*)
⟨*proof*⟩

end

end

5 Algebra-only Theorems

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

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

lemma *pythagoras*:
  fixes *v w* :: *'a::real-inner*
  assumes *v* · *w* = *0*
  shows *norm* (*v*+*w*)^*2* = *norm v*^*2* + *norm w*^*2*
  ⟨*proof*⟩

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

definition *ind-vec* :: *'n set* $\Rightarrow$ *real*^*'n*
  where *ind-vec S* = (χ *i. of-bool*(*i* $\in$ *S*))

lemma *diag-mult-eq*: *diag x* ** *diag y* = *diag* (*x* * *y*)
  ⟨*proof*⟩

lemma *diag-vec-mult-eq*: *diag x* *v *y* = *x* * *y*
  ⟨*proof*⟩

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

lemma *matrix-norm-boundI*:
  assumes $\bigwedge$*x. norm* (*A* *v *x*) $\le$ *l* * *norm x*
  shows *matrix-norm-bound A l*
  ⟨*proof*⟩

lemma *matrix-norm-boundD*:
  assumes *matrix-norm-bound A l*
  shows *norm* (*A* *v *x*) $\le$ *l* * *norm x*
  ⟨*proof*⟩

lemma *matrix-norm-bound-nonneg*:
  fixes A :: *real^'n^'m*
  assumes *matrix-norm-bound* A l
  shows $l \geq 0$
⟨*proof*⟩

lemma *matrix-norm-bound-0*:
  assumes *matrix-norm-bound* A *0*
  shows $A = (0::real^{\frown\prime}n^{\frown\prime}m)$
⟨*proof*⟩

lemma *matrix-norm-bound-diag*:
  fixes x :: *real^'n*
  assumes $\bigwedge i.\ |x\ \$\ i| \leq l$
  shows *matrix-norm-bound* (*diag* x) l
⟨*proof*⟩

lemma *vector-scaleR-matrix-ac-2*: $b *_R (A::real^{\frown\prime}n^{\frown\prime}m) *v\ x = b *_R (A *v\ x)$
  ⟨*proof*⟩

lemma *matrix-norm-bound-scale*:
  assumes *matrix-norm-bound* A l
  shows *matrix-norm-bound* $(b *_R A)$ $(|b| * l)$
⟨*proof*⟩

definition *nonneg-mat* :: *real^'n^'m* ⇒ *bool*
  where *nonneg-mat* $A = (\forall\, i\ j.\ A\ \$\ i\ \$\ j \geq 0)$

lemma *nonneg-mat-1*:
  shows *nonneg-mat* (*mat 1*)
  ⟨*proof*⟩

lemma *nonneg-mat-prod*:
  assumes *nonneg-mat* A *nonneg-mat* B
  shows *nonneg-mat* $(A ** B)$
  ⟨*proof*⟩

lemma *nonneg-mat-transpose*:
  nonneg-mat (*transpose* A) = *nonneg-mat* A
  ⟨*proof*⟩

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

lemma *spec-boundD1*:
  assumes *spec-bound* M l
  shows $0 \leq l$
  ⟨*proof*⟩

lemma *spec-boundD2*:
  assumes *spec-bound* M l
  assumes $v \cdot 1 = 0$
  shows $norm\ (M *v\ v) \leq l * norm\ v$
  ⟨*proof*⟩

lemma *spec-bound-mono*:
  assumes *spec-bound* M α $\alpha \leq \beta$
  shows *spec-bound* M β

⟨*proof*⟩

definition *markov* :: *real^'n^'n ⇒ bool*
  where *markov M* = (*nonneg-mat M* ∧ *M* ∗*v* *1* = *1* ∧ *1* *v*∗ *M* = *1*)

lemma *markov-symI*:
  assumes *nonneg-mat A transpose A* = *A A* ∗*v* *1* = *1*
  shows *markov A*
⟨*proof*⟩

lemma *markov-apply*:
  assumes *markov M*
  shows *M* ∗*v* *1* = *1 1* *v*∗ *M* = *1*
  ⟨*proof*⟩

lemma *markov-transpose*:
  markov A = *markov* (*transpose A*)
  ⟨*proof*⟩
fun *matrix-pow* **where**
  matrix-pow M 0 = *mat 1* |
  matrix-pow M (*Suc n*) = *M* ∗∗ (*matrix-pow M n*)

lemma *markov-orth-inv*:
  assumes *markov A*
  shows *inner* (*A* ∗*v* *x*) *1* = *inner x 1*
⟨*proof*⟩

lemma *markov-id*:
  markov (*mat 1*)
  ⟨*proof*⟩

lemma *markov-mult*:
  assumes *markov A markov B*
  shows *markov* (*A* ∗∗ *B*)
⟨*proof*⟩

lemma *markov-matrix-pow*:
  assumes *markov A*
  shows *markov* (*matrix-pow A k*)
  ⟨*proof*⟩

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*⟩

lemma *spec-bound-pow*:
  assumes *markov A*
  assumes *spec-bound A l*
  shows *spec-bound* (*matrix-pow A k*) (*l^k*)
⟨*proof*⟩

fun *intersperse* :: *'a ⇒ 'a list ⇒ '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 $intersperse\ z\ (xs@[y]) = intersperse\ z\ xs@[z,y]$
  ⟨*proof*⟩

lemma *foldl-intersperse*:
  assumes $xs \neq []$
  shows $foldl\ f\ a\ ((intersperse\ x\ xs)@[x]) = foldl\ (\lambda y\ z.\ f\ (f\ y\ z)\ x)\ a\ xs$
  ⟨*proof*⟩

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*⟩

context *regular-graph-tts*
begin

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

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

lemma *inner-1-1*: $1 \cdot (1::real^{\frown}'n) = CARD('n)$
  ⟨*proof*⟩

definition $proj\text{-}unit :: real^{\frown}'n \Rightarrow real^{\frown}'n$
  where $proj\text{-}unit\ v = (1 \cdot v) *_R stat$

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

lemma *proj-rem-orth*: $1 \cdot (proj\text{-}rem\ v) = 0$
  ⟨*proof*⟩

lemma *split-vec*: $v = proj\text{-}unit\ v + proj\text{-}rem\ v$
  ⟨*proof*⟩

lemma *apply-J*: $J *v\ x = proj\text{-}unit\ x$
⟨*proof*⟩

lemma *spec-bound-J*: $spec\text{-}bound\ (J :: real^{\frown}'n^{\frown}'n)\ 0$
⟨*proof*⟩

lemma *matrix-decomposition-lemma-aux*:
  fixes $A :: real^{\frown}'n^{\frown}'n$
  assumes *markov A*
  shows $spec\text{-}bound\ A\ l \longleftrightarrow matrix\text{-}norm\text{-}bound\ (A - (1-l) *_R J)\ l$ (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

lemma *matrix-decomposition-lemma*:
  fixes $A :: real^{\frown}'n^{\frown}'n$
  assumes *markov A*
  shows $spec\text{-}bound\ A\ l \longleftrightarrow (\exists E.\ A = (1-l) *_R J + l *_R E \wedge matrix\text{-}norm\text{-}bound\ E\ 1 \wedge l \geq 0)$
    (**is** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

lemma *hitting-property-alg*:
  fixes *S* :: (*'n* :: *finite*) *set*
  assumes *l-range*: *l* ∈ {*0..1*}
  defines *P* ≡ *diag* (*ind-vec S*)
  defines *μ* ≡ *card S* / *CARD*(*'n*)
  assumes ⋀*M*. *M* ∈ *set Ms* ⟹ *spec-bound M l* ∧ *markov M*
  shows *foldl* (λ*x M*. *P* ∗*v* (*M* ∗*v x*)) (*P* ∗*v stat*) *Ms* · *1* ≤ (*μ* + *l* ∗ (*1*−*μ*))^(*length Ms*+*1*)
⟨*proof*⟩

lemma *upto-append*:
  assumes *i* ≤ *j j* ≤ *k*
  shows [*i*..<*j*]@[*j*..<*k*] = [*i*..<*k*]
  ⟨*proof*⟩

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

lemma *bool-list-split*:
  assumes *bool-list-split xs* = (*ys*,*z*)
  shows *xs* = *concat* (*map* (λ*k*. *replicate k False*@[*True*]) *ys*)@*replicate z False*
  ⟨*proof*⟩

lemma *bool-list-split-count*:
  assumes *bool-list-split xs* = (*ys*,*z*)
  shows *length* (*filter id xs*) = *length ys*
  ⟨*proof*⟩

lemma *foldl-concat*:
  foldl f a (*concat xss*) = *foldl* (λ*y xs*. *foldl f y xs*) *a xss*
  ⟨*proof*⟩

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

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

end

lemma *foldl-matrix-mult-expand*:
  fixes *Ms* :: ((*'r*::{*semiring-1*,*comm-monoid-mult*})^*'a*^*'a*) *list*

shows (*foldl* (λ*x M*. *M* ∗*v x*) *a Ms*) $ *k* = (∑ *x* | *length x* = *length Ms*+*1* ∧ *x*! *length Ms* = *k*.
(∏ *i*< *length Ms*. (*Ms* ! *i*) $ (*x* ! (*i*+*1*)) $ (*x* ! *i*)) ∗ *a* $ (*x* ! *0*))
⟨*proof*⟩

lemma *foldl-matrix-mult-expand-2*:
fixes *Ms* :: (*real*^'*a*^'*a*) *list*
shows (*foldl* (λ*x M*. *M* ∗*v x*) *a Ms*) · *1* = (∑ *x* | *length x* = *length Ms*+*1*.
(∏ *i*< *length Ms*. (*Ms* ! *i*) $ (*x* ! (*i*+*1*)) $ (*x* ! *i*)) ∗ *a* $ (*x* ! *0*))
(**is** *?L* = *?R*)
⟨*proof*⟩

end

6 Spectral Theory

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

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

unbundle *intro-cong-syntax*

hide-const *Matrix-Legacy.transpose*
hide-const *Matrix-Legacy.row*
hide-const *Matrix-Legacy.mat*
hide-const *Matrix.mat*
hide-const *Matrix.row*
hide-fact *Matrix-Legacy.row-def*
hide-fact *Matrix-Legacy.mat-def*
hide-fact *Matrix.vec-eq-iff*
hide-fact *Matrix.mat-def*
hide-fact *Matrix.row-def*
no-notation *Matrix.scalar-prod* (**infix** ‹·› *70*)
no-notation *Ordered-Semiring.max* (‹*Max*₁›)

lemma *mult-right-mono'*: *y* ≥ (*0*::*real*) ⟹ *x* ≤ *z* ∨ *y* = *0* ⟹ *x* ∗ *y* ≤ *z* ∗ *y*
⟨*proof*⟩

lemma *poly-prod-zero*:
fixes *x* :: '*a* :: *idom*
assumes *poly* (∏ *a*∈#*xs*. [:− *a*, *1*:]) *x* = *0*
shows *x* ∈# *xs*
⟨*proof*⟩

lemma *poly-prod-inj-aux-1*:
fixes *xs ys* :: ('*a* :: *idom*) *multiset*
assumes *x* ∈# *xs*
assumes (∏ *a*∈#*xs*. [:− *a*, *1*:]) = (∏ *a*∈#*ys*. [:− *a*, *1*:])
shows *x* ∈# *ys*
⟨*proof*⟩

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*⟩

lemma *poly-prod-inj*:
  fixes *xs ys* :: (*'a* :: *idom*) *multiset*
  assumes $(\prod a{\in}\#xs.\ [:- a,\ 1:]) = (\prod a{\in}\#ys.\ [:- a,\ 1:])$
  shows $xs = ys$
  ⟨*proof*⟩

definition *eigenvalues* :: (*'a*::*comm-ring-1*)^*'n*^*'n* ⇒ *'a multiset*
  where
    eigenvalues A = (*SOME as. charpoly A* = $(\prod a{\in}\#as.\ [:- a,\ 1:])$ ∧ *size as* = *CARD* (*'n*))

lemma *char-poly-factorized-hma*:
  fixes *A* :: *complex*^*'n*^*'n*
  shows ∃ *as. charpoly A* = $(\prod a{\leftarrow}as.\ [:- a,\ 1:])$ ∧ *length as* = *CARD* (*'n*)
  ⟨*proof*⟩

lemma *eigvals-poly-length*:
  fixes *A* :: *complex*^*'n*^*'n*
  shows
    charpoly A = $(\prod a{\in}\#eigenvalues\ A.\ [:- a,\ 1:])$ (**is** *?A*)
    size (*eigenvalues A*) = *CARD* (*'n*) (**is** *?B*)
⟨*proof*⟩

lemma *similar-matrix-eigvals*:
  fixes *A B* :: *complex*^*'n*^*'n*
  assumes *similar-matrix A B*
  shows *eigenvalues A* = *eigenvalues B*
⟨*proof*⟩

definition *upper-triangular-hma* :: *'a*::*zero*^*'n*^*'n* ⇒ *bool*
  where *upper-triangular-hma A* ≡
    ∀ *i*. ∀ *j*. (*to-nat j* < *Bij-Nat.to-nat i* ⟶ *A* \$*h i* \$*h j* = *0*)

lemma *for-all-reindex2*:
  assumes *range f* = *A*
  shows (∀ *x* ∈ *A*. ∀ *y* ∈ *A*. *P x y*) ⟷ (∀ *x y*. *P* (*f x*) (*f y*))
  ⟨*proof*⟩

lemma *upper-triangular-hma*:
  fixes *A* :: (*'a*::*zero*)^*'n*^*'n*
  shows *upper-triangular* ($from\text{-}hma_m$ *A*) = *upper-triangular-hma A* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *from-hma-carrier*:
  fixes *A* :: *'a*^(*'n*::*finite*)^(*'m*::*finite*)
  shows $from\text{-}hma_m$ *A* ∈ *carrier-mat* (*CARD* (*'m*)) (*CARD* (*'n*))
  ⟨*proof*⟩

definition *diag-mat-hma* :: *'a*^*'n*^*'n* ⇒ *'a multiset*
  where *diag-mat-hma A* = *image-mset* (λ*i*. *A* \$*h i* \$*h i*) (*mset-set UNIV*)

lemma *diag-mat-hma*:

fixes A :: $'a^{\frown\prime}n^{\frown\prime}n$
shows *mset* (*diag-mat* (*from-hma*$_m$ *A*)) = *diag-mat-hma A* (**is** *?L* = *?R*)
⟨*proof*⟩

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

lemma *adjoint-hma-eq*: *adjoint-hma A* \$*h i* \$*h j* = *cnj* (*A* \$*h j* \$*h i*)
⟨*proof*⟩

lemma *adjoint-hma*:
fixes *A* :: *complex*⌢('*n*::*finite*)⌢('*m*::*finite*)
shows *mat-adjoint* (*from-hma*$_m$ *A*) = *from-hma*$_m$ (*adjoint-hma A*)
⟨*proof*⟩

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

context
includes *lifting-syntax*
begin

lemma *cinner-hma*:
fixes *x y* :: $complex^{\frown\prime}n$
shows *cinner x y* = (*from-hma*$_v$ *x*) ·*c* (*from-hma*$_v$ *y*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *cinner-hma-transfer*[*transfer-rule*]:
(*HMA-V* ===> *HMA-V* ===> (=)) (·*c*) *cinner*
⟨*proof*⟩

lemma *adjoint-hma-transfer*[*transfer-rule*]:
(*HMA-M* ===> *HMA-M*) (*mat-adjoint*) *adjoint-hma*
⟨*proof*⟩

end

lemma *adjoint-adjoint-id*[*simp*]: *adjoint-hma* (*adjoint-hma A*) = *A*
⟨*proof*⟩

lemma *adjoint-def-alter-hma*:
cinner (*A* ∗*v v*) *w* = *cinner v* (*adjoint-hma A* ∗*v w*)
⟨*proof*⟩

lemma *cinner-0*: *cinner 0 0* = *0*
⟨*proof*⟩

lemma *cinner-scale-left*: *cinner* (*a* ∗*s v*) *w* = *a* ∗ *cinner v w*
⟨*proof*⟩

lemma *cinner-scale-right*: *cinner v* (*a* ∗*s w*) = *cnj a* ∗ *cinner v w*
⟨*proof*⟩

lemma *norm-of-real*:
shows *norm* (*map-vector complex-of-real v*) = *norm v*
⟨*proof*⟩

definition *unitary-hma* :: $complex^{\frown\prime}n^{\frown\prime}n \Rightarrow bool$
where *unitary-hma A* ⟷ *A* ∗∗ *adjoint-hma A* = *Finite-Cartesian-Product.mat 1*

definition *unitarily-equiv-hma* **where**
unitarily-equiv-hma A B U ≡ (*unitary-hma U* ∧ *similar-matrix-wit A B U* (*adjoint-hma U*))

definition *diagonal-mat* :: ('*a*::*zero*)^('*n*::*finite*)^'*n* ⇒ *bool* **where**
diagonal-mat A ≡ (∀ *i*. ∀ *j*. *i* ≠ *j* ⟶ *A* \$*h i* \$*h j* = *0*)

lemma *diagonal-mat-ex*:
assumes *diagonal-mat A*
shows *A* = *diag* (χ *i*. *A* \$*h i* \$*h i*)
⟨*proof*⟩

lemma *diag-diagonal-mat*[*simp*]: *diagonal-mat* (*diag x*)
⟨*proof*⟩

lemma *diag-imp-upper-tri*: *diagonal-mat A* ⟹ *upper-triangular-hma A*
⟨*proof*⟩

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

definition *real-diag-decomp-hma* **where**
real-diag-decomp-hma A d U ≡ *unitary-diag A d U* ∧
(∀ *i*. *d* \$*h i* ∈ *Reals*)

definition *hermitian-hma* :: *complex*^'*n*^'*n* ⇒ *bool* **where**
hermitian-hma A = (*adjoint-hma A* = *A*)

lemma *from-hma-one*:
$\mathit{from\text{-}hma}_m$ (*mat 1* :: (('*a*::{*one*,*zero*})^'*n*^'*n*)) = $\mathit{1}_m$ *CARD*('*n*)
⟨*proof*⟩

lemma *from-hma-mult*:
fixes *A* :: ('*a* :: *semiring-1*)^'*m*^'*n*
fixes *B* :: '*a*^'*k*^'*m*::*finite*
shows $\mathit{from\text{-}hma}_m$ *A* ∗ $\mathit{from\text{-}hma}_m$ *B* = $\mathit{from\text{-}hma}_m$ (*A* ∗∗ *B*)
⟨*proof*⟩

lemma *hermitian-hma*:
hermitian-hma A = *hermitian* ($\mathit{from\text{-}hma}_m$ *A*)
⟨*proof*⟩

lemma *unitary-hma*:
fixes *A* :: *complex*^'*n*^'*n*
shows *unitary-hma A* = *unitary* ($\mathit{from\text{-}hma}_m$ *A*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *unitary-hmaD*:
fixes *A* :: *complex*^'*n*^'*n*
assumes *unitary-hma A*
shows *adjoint-hma A* ∗∗ *A* = *mat 1* (**is** *?A*) *A* ∗∗ *adjoint-hma A* = *mat 1* (**is** *?B*)
⟨*proof*⟩

lemma *unitary-hma-adjoint*:
assumes *unitary-hma A*
shows *unitary-hma* (*adjoint-hma A*)
⟨*proof*⟩

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

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

lemma *diagonal-mat-hma*:
fixes *A* :: (*'a* :: *zero*)$^{\frown}$(*'n* :: *finite*)$^{\frown}$*'n*
shows *diagonal-mat A* = *Matrix.diagonal-mat* (*from-hma*$_m$ *A*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *unitary-diag-hma*:
fixes *A* :: *complex*$^{\frown}$*'n*$^{\frown}$*'n*
shows *unitary-diag A d U* =
Spectral-Theory-Complements.unitary-diag (*from-hma*$_m$ *A*) (*from-hma*$_m$ (*diag d*)) (*from-hma*$_m$
U)
⟨*proof*⟩

lemma *real-diag-decomp-hma*:
fixes *A* :: *complex*$^{\frown}$*'n*$^{\frown}$*'n*
shows *real-diag-decomp-hma A d U* =
real-diag-decomp (*from-hma*$_m$ *A*) (*from-hma*$_m$ (*diag d*)) (*from-hma*$_m$ *U*)
⟨*proof*⟩

lemma *diagonal-mat-diag-ex-hma*:
assumes *Matrix.diagonal-mat A A* ∈ *carrier-mat CARD*(*'n*) *CARD* (*'n* :: *finite*)
shows *from-hma*$_m$ (*diag* (χ (*i*::*'n*). *A* \$\$ (*to-nat i*,*to-nat i*))) = *A*
⟨*proof*⟩

theorem *commuting-hermitian-family-diag-hma*:
fixes *Af* :: (*complex*$^{\frown}$*'n*$^{\frown}$*'n*) *set*
assumes *finite Af*
and $Af \neq \{\}$
and $\bigwedge A.\ A \in Af \Longrightarrow$ *hermitian-hma A*
and $\bigwedge A\ B.\ A \in Af \Longrightarrow B \in Af \Longrightarrow A ** B = B ** A$
shows ∃ *U*. ∀ *A*∈ *Af*. ∃ *B*. *real-diag-decomp-hma A B U*
⟨*proof*⟩

lemma *char-poly-upper-triangular*:
fixes *A* :: *complex*$^{\frown}$*'n*$^{\frown}$*'n*
assumes *upper-triangular-hma A*
shows *charpoly A* = ($\prod$ *a* ∈# *diag-mat-hma A*. [:− *a*, *1*:])
⟨*proof*⟩

lemma *upper-tri-eigvals*:
fixes *A* :: *complex*$^{\frown}$*'n*$^{\frown}$*'n*
assumes *upper-triangular-hma A*
shows *eigenvalues A* = *diag-mat-hma A*
⟨*proof*⟩

lemma *cinner-self*:
  fixes *v* :: *complex*^'*n*
  shows *cinner v v* = *norm v*^*2*
⟨*proof*⟩

lemma *unitary-iso*:
  assumes *unitary-hma U*
  shows *norm* (*U* ∗*v v*) = *norm v*
⟨*proof*⟩

lemma (**in** *semiring-hom*) *mult-mat-vec-hma*:
  map-vector hom (*A* ∗*v v*) = *map-matrix hom A* ∗*v map-vector hom v*
  ⟨*proof*⟩

lemma (**in** *semiring-hom*) *mat-hom-mult-hma*:
  map-matrix hom (*A* ∗∗ *B*) = *map-matrix hom A* ∗∗ *map-matrix hom B*
  ⟨*proof*⟩

context *regular-graph-tts*
begin

lemma *to-nat-less-n*: *to-nat* (*x*::'*n*) < *n*
  ⟨*proof*⟩

lemma *to-nat-from-nat*: *x* < *n* ⟹ *to-nat* (*from-nat x* :: '*n*) = *x*
  ⟨*proof*⟩

lemma *hermitian-A*: *hermitian-hma A*
  ⟨*proof*⟩

lemma *nonneg-A*: *nonneg-mat A*
  ⟨*proof*⟩

lemma *g-step-1*:
  assumes *v* ∈ *verts G*
  shows *g-step* (λ-. *1*) *v* = *1* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *markov*: *markov* (*A* :: *real*^'*n*^'*n*)
⟨*proof*⟩

lemma *nonneg-J*: *nonneg-mat J*
  ⟨*proof*⟩

lemma *J-eigvals*: *eigenvalues J* = {#*1*::*complex*#} + *replicate-mset* (*n* − *1*) *0*
⟨*proof*⟩

lemma *J-markov*: *markov J*
⟨*proof*⟩

lemma *markov-complex-apply*:
  assumes *markov M*
  shows (*map-matrix complex-of-real M*) ∗*v* (*1* :: *complex*^'*n*) = *1* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *J-A-comm-real*: *J* ∗∗ *A* = *A* ∗∗ (*J* :: *real*^'*n*^'*n*)
⟨*proof*⟩

lemma *J-A-comm*: $J ** A = A ** (J :: complex^{\frown\prime}n^{\frown\prime}n)$ (**is** *?L = ?R*)
⟨*proof*⟩

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

definition γ_2 :: *'n itself* ⇒ *real* **where**
γ_2 *- = (if n > 1 then Max-mset {# Re x. x ∈# (eigenvalues A − {#1#})#} else 0)*

lemma *J-sym*: *hermitian-hma J*
⟨*proof*⟩

lemma
shows *evs-real*: *set-mset (eigenvalues A::complex multiset)* ⊆ ℝ (**is** *?R1*)
and *ev-1*: *(1::complex) ∈# eigenvalues A*
and γ_a*-ge-0*: γ_a *TYPE ('n)* ≥ *0*
and *find-any-ev*:
∀ α ∈# *eigenvalues A − {#1#}*. ∃ *v. cinner v 1 = 0* ∧ *v ≠ 0* ∧ *A *v v = α *s v*
and γ_a*-bound*: ∀ *v. cinner v 1 = 0* ⟶ *norm (A *v v)* ≤ γ_a *TYPE('n) * norm v*
and γ_2*-bound*: ∀ *(v::real^'n). v · 1 = 0* ⟶ *v · (A *v v)* ≤ γ_2 *TYPE ('n) * norm v^2*
⟨*proof*⟩

lemma *find-any-real-ev*:
assumes *complex-of-real α ∈# eigenvalues A − {#1#}*
shows ∃ *v. v · 1 = 0* ∧ *v ≠ 0* ∧ *A *v v = α *s v*
⟨*proof*⟩

lemma *size-evs*:
size (eigenvalues A − {#1::complex#}) = n−1
⟨*proof*⟩

lemma *find-*γ_2:
assumes *n > 1*
shows γ_a *TYPE('n) ∈# image-mset cmod (eigenvalues A − {#1::complex#})*
⟨*proof*⟩

lemma γ_2*-real-ev*:
assumes *n > 1*
shows ∃ *v.* (∃ *α. abs α=*γ_a *TYPE('n)* ∧ *v · 1=0* ∧ *v ≠ 0* ∧ *A *v v = α *s v)*
⟨*proof*⟩

lemma γ_a*-real-bound*:
fixes *v :: real^'n*
assumes *v · 1 = 0*
shows *norm (A *v v)* ≤ γ_a *TYPE('n) * norm v*
⟨*proof*⟩

lemma Λ_e*-eq-*Λ: $\Lambda_a = \gamma_a$ *TYPE('n)*
⟨*proof*⟩

lemma γ_2*-ev*:
assumes *n > 1*
shows ∃ *v. v · 1 = 0* ∧ *v ≠ 0* ∧ *A *v v =* γ_2 *TYPE('n) *s v*
⟨*proof*⟩

lemma Λ_2*-eq-*γ_2: $\Lambda_2 = \gamma_2$ *TYPE ('n)*
⟨*proof*⟩

lemma *expansionD2*:
  assumes *g-inner f* (λ-. *1*) = *0*
  shows *g-norm* (*g-step f*) ≤ Λ_a ∗ *g-norm f* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

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

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

lemma Λ_2*-range*: $|\Lambda_2| \le \Lambda_a$
⟨*proof*⟩

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*) Λ_2*-range* =
  regular-graph-tts.Λ_2*-range*[*OF eg-tts-1*,
    internalize-sort 'n :: *finite*, *OF - regular-graph-axioms*,
    unfolded remove-finite-premise, *cancel-type-definition*, *OF verts-non-empty*]

unbundle *no intro-cong-syntax*

end

7 Cheeger Inequality

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

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

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

context *regular-graph*
begin

lemma *edge-expansionD2*:
  assumes *m* = *card* (*S* ∩ *verts G*) *2*∗*m* ≤ *n*
  shows Λ_e ∗ *m* ≤ *real* (*card* (*edges-betw S* (−*S*)))
⟨*proof*⟩

lemma *edges-betw-sym*:
  card (*edges-betw S T*) = *card* (*edges-betw T S*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *edges-betw-reg*:
  assumes *S* ⊆ *verts G*
  shows *card* (*edges-betw S UNIV*) = *card S* ∗ *d* (**is** *?L* = *?R*)
⟨*proof*⟩

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*⟩

end

lemma *surj-onI*:
  assumes $\bigwedge x.\ x \in B \Longrightarrow g\ x \in A \wedge f\ (g\ x) = x$
  shows $B \subseteq f\ `\ A$
  ⟨*proof*⟩

lemma *find-sorted-bij-1*:
  fixes $g :: 'a \Rightarrow ('b :: linorder)$
  assumes *finite S*
  shows $\exists f.\ bij\text{-}betw\ f\ \{..<card\ S\}\ S \wedge mono\text{-}on\ \{..<card\ S\}\ (g \circ f)$
⟨*proof*⟩

lemma *find-sorted-bij-2*:
  fixes $g :: 'a \Rightarrow ('b :: linorder)$
  assumes *finite S*
  shows $\exists f.\ bij\text{-}betw\ f\ S\ \{..<card\ S\} \wedge (\forall x\ y.\ x \in S \wedge y \in S \wedge f\ x < f\ y \longrightarrow g\ x \leq g\ y)$
⟨*proof*⟩

context *regular-graph-tts*
begin

Normalized Laplacian of the graph

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

lemma *L-pos-semidefinite*:
  fixes $v :: real\ \hat{}'n$
  shows $v \cdot (L *v\ v) \geq 0$
⟨*proof*⟩

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

lemma *cheeger-aux-1*:
  assumes $n > 1$
  shows $\Lambda_e \leq d * sqrt\ (2 * (1-\Lambda_2))$
⟨*proof*⟩

end

context *regular-graph*
begin

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

theorem *cheeger-inequality*:
  assumes $n > 1$
  shows $\Lambda_e \in \{d * (1 - \Lambda_2)\ /\ 2..\ d * sqrt\ (2 * (1 - \Lambda_2))\}$
  ⟨*proof*⟩

unbundle *no intro-cong-syntax*

end

end

8 Margulis Gabber Galil Construction

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

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

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

fun mgg-graph-step :: nat ⇒ (int × int) ⇒ (nat × int) ⇒ (int × int)
  where mgg-graph-step n (i,j) (l,σ) =
  [ ((i+σ∗(2∗j+0)) mod int n, j), (i, (j+σ∗(2∗i+0)) mod int n)
  , ((i+σ∗(2∗j+1)) mod int n, j), (i, (j+σ∗(2∗i+1)) mod int n) ] ! l

definition mgg-graph :: nat ⇒ (int × int, (int × int, nat × int) arc) pre-digraph where
  mgg-graph n =
    (| verts = {0..<n} × {0..<n},
    arcs = (λ(t,l). (Arc t (mgg-graph-step n t l) l))`(({0..<int n}×{0..<int n})×({..<4}×{−1,1})),
      tail = arc-tail,
      head = arc-head |)

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

abbreviation G where G ≡ mgg-graph m

lemma wf-digraph: wf-digraph (mgg-graph m)
⟨proof⟩

lemma mgg-finite: fin-digraph (mgg-graph m)
⟨proof⟩

interpretation fin-digraph mgg-graph m
  ⟨proof⟩

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

lemma *arcs-sym*:
arcs G = *arcs-pos* ∪ *arcs-neg*
⟨*proof*⟩

lemma *sym*: *symmetric-multi-graph* (*mgg-graph m*)
⟨*proof*⟩

lemma *out-deg*:
assumes $v \in$ *verts G*
shows *out-degree G v* = *8*
⟨*proof*⟩

lemma *verts-ne*:
verts G ≠ {}
⟨*proof*⟩

sublocale *regular-graph mgg-graph m*
⟨*proof*⟩

lemma *d-eq-8*: *d* = *8*
⟨*proof*⟩

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

lemma *g-inner-sum-left*:
assumes *finite I*
shows *g-inner* (λ*x*. (∑ *i* ∈ *I*. *f i x*)) *g* = (∑ *i*∈ *I*. *g-inner* (*f i*) *g*)
⟨*proof*⟩

lemma *g-inner-sum-right*:
assumes *finite I*
shows *g-inner f* (λ*x*. (∑ *i* ∈ *I*. *g i x*)) = (∑ *i*∈ *I*. *g-inner f* (*g i*))
⟨*proof*⟩

lemma *g-inner-reindex*:
assumes *bij-betw h* (*verts G*) (*verts G*)
shows *g-inner f g* = *g-inner* (λ*x*. (*f* (*h x*))) (λ*x*. (*g* (*h x*)))
⟨*proof*⟩

definition ω_F :: *real* ⇒ *complex* **where** ω_F *x* = *cis* (*2*∗*pi*∗*x*/*m*)

lemma ω_F-*simps*:
ω_F (*x* + *y*) = ω_F *x* ∗ ω_F *y*
ω_F (*x* − *y*) = ω_F *x* ∗ ω_F (−*y*)
cnj (ω_F *x*) = ω_F (−*x*)
⟨*proof*⟩

lemma ω_F-*cong*:
fixes *x y* :: *int*
assumes *x mod m* = *y mod m*
shows ω_F (*of-int x*) = ω_F (*of-int y*)
⟨*proof*⟩

lemma *cis-eq-1-imp*:
assumes *cis* (*2* ∗ *pi* ∗ *x*) = *1*
shows $x \in \mathbb{Z}$
⟨*proof*⟩

lemma ω_F*-eq-1-iff*:
  fixes x :: *int*
  shows $\omega_F\ x = 1 \longleftrightarrow x\ mod\ m = 0$
⟨*proof*⟩

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

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

lemma *FT-add*: $FT\ (\lambda x.\ f\ x + g\ x)\ v = FT\ f\ v + FT\ g\ v$
  ⟨*proof*⟩

lemma *FT-zero*: $FT\ (\lambda x.\ 0)\ v = 0$
  ⟨*proof*⟩

lemma *FT-sum*:
  assumes *finite I*
  shows $FT\ (\lambda x.\ (\sum i \in I.\ f\ i\ x))\ v = (\sum i \in I.\ FT\ (f\ i)\ v)$
  ⟨*proof*⟩

lemma *FT-scale*: $FT\ (\lambda x.\ c * f\ x)\ v = c * FT\ f\ v$
  ⟨*proof*⟩

lemma *FT-cong*:
  assumes $\bigwedge x.\ x \in verts\ G \Longrightarrow f\ x = g\ x$
  shows $FT\ f = FT\ g$
  ⟨*proof*⟩

lemma *parseval*:
  $g\text{-}inner\ f\ g = g\text{-}inner\ (FT\ f)\ (FT\ g)/m\hat{\ }2$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *plancharel*:
  $(\sum v \in verts\ G.\ norm\ (f\ v)\hat{\ }2) = (\sum v \in verts\ G.\ norm\ (FT\ f\ v)\hat{\ }2)/m\hat{\ }2$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *FT-swap*:
  $FT\ (\lambda x.\ f\ (snd\ x,\ fst\ x))\ (u,v) = FT\ f\ (v,u)$
⟨*proof*⟩

lemma *mod-add-mult-eq*:
  fixes $a\ x\ y$ :: *int*
  shows $(a + x * (y\ mod\ m))\ mod\ m = (a+x*y)\ mod\ m$
  ⟨*proof*⟩

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 $f\ (x,y) = f\ (x\ mod\ m,\ y\ mod\ m)$
  ⟨*proof*⟩

lemma *periodic-comp*:
  assumes *periodic f*
  shows $periodic\ (\lambda x.\ g\ (f\ x))$

⟨*proof*⟩

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 $f\ (x,y) = f\ (u,\ v)$
⟨*proof*⟩

lemma *periodic-FT*: *periodic* (*FT f*)
⟨*proof*⟩

lemma *FT-sheer-aux*:
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*⟩

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*⟩

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 γ*-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 $compare\ x\ y = (fst\ x \le fst\ y \wedge snd\ x \le snd\ y \wedge x \ne y)$

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

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

lemma α*-inv*: $1/\alpha = \alpha/2$
⟨*proof*⟩

definition γ :: $int \times int \Rightarrow int \times int \Rightarrow real$
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 γ*-sym*: $\gamma\ x\ y * \gamma\ y\ x = 1$
⟨*proof*⟩

lemma γ*-nonneg*: $\gamma\ x\ y \ge 0$
⟨*proof*⟩

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

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

lemma γ'*-cases*:
 $abs\ (x-1/2) = abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = 1$
 $abs\ (x-1/2) > abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = 1/\alpha$
 $abs\ (x-1/2) < abs\ (y-1/2) \Longrightarrow \gamma'\ x\ y = \alpha$
 ⟨*proof*⟩

lemma *if-cong-direct*:
 assumes $a = b$
 assumes $c = d'$
 assumes $e = f$
 shows $(if\ a\ then\ c\ else\ e) = (if\ b\ then\ d'\ else\ f)$
 ⟨*proof*⟩

lemma γ'*-cong*:
 assumes $abs\ (x-1/2) = abs\ (u-1/2)$
 assumes $abs\ (y-1/2) = abs\ (v-1/2)$
 shows $\gamma'\ x\ y = \gamma'\ u\ v$
 ⟨*proof*⟩

lemma *add-swap-cong*:
 fixes $x\ y\ u\ v$ :: *'a* :: *ab-semigroup-add*
 assumes $x = y\ u = v$
 shows $x + u = v + y$
 ⟨*proof*⟩

lemma *frac-cong*:
 fixes $x\ y$ :: *real*
 assumes $x - y \in \mathbb{Z}$
 shows $frac\ x = frac\ y$
⟨*proof*⟩

lemma *frac-expand*:
 fixes x :: *real*
 shows $frac\ x = (if\ x < (-1)\ then\ (x-\lfloor x\rfloor)\ else\ (if\ x < 0\ then\ (x+1)\ else\ (if\ x < 1\ then\ x\ else\ (if\ x < 2\ then\ (x-1)\ else\ (x-\lfloor x\rfloor)))))$
⟨*proof*⟩

lemma *one-minus-frac*:
 fixes x :: *real*
 shows $1 - frac\ x = (if\ x \in \mathbb{Z}\ then\ 1\ else\ frac\ (-x))$
 ⟨*proof*⟩

lemma *abs-rev-cong*:
 fixes $x\ y$ :: *real*
 assumes $x = -\ y$
 shows $abs\ x = abs\ y$
 ⟨*proof*⟩

lemma *cos-pi-ge-0*:

assumes $x \in \{-1/2..\ 1/2\}$
shows $cos\ (pi * x) \geq 0$
⟨*proof*⟩

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*⟩

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

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

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

lemma *fun-bound-real-1*:
assumes $x \in \{0..<1\}\ y \in \{0..1/2\}\ (x,y) \neq (0,0)$
shows $|cos(pi*x)|*\varphi\ x\ y + |cos(pi*y)|*\varphi\ y\ x \leq 2.5 * sqrt\ 2$ (**is** $?L \leq ?R$)
⟨*proof*⟩

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 $|cos(pi*x)|*\varphi\ x\ y + |cos(pi*y)|*\varphi\ y\ x \leq 2.5 * sqrt\ 2$ (**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *mod-to-frac*:
fixes $x :: int$
shows $real\text{-}of\text{-}int\ (x\ mod\ m) = m * frac\ (x/m)$ (**is** $?L = ?R$)
⟨*proof*⟩

lemma *fun-bound*:
assumes $v \in verts\ G\ v \neq (0,0)$
shows $\tau(fst\ v)*(\gamma\ v\ (S_2\ v)+\gamma\ v\ (T_2\ v))+\tau(snd\ v)*(\gamma\ v\ (S_1\ v)+\gamma\ v\ (T_1\ v)) \leq 2.5 * sqrt\ 2$
(**is** $?L \leq ?R$)
⟨*proof*⟩

Equation 15 in Proof of Theorem 8.8

lemma *hoory-8-8*:
fixes $f :: int \times int \Rightarrow real$
assumes $\bigwedge x.\ f\ x \geq 0$
assumes $f\ (0,0) = 0$
assumes *periodic f*
shows $g\text{-}inner\ f\ (\lambda x.\ f(S_2\ x)*\tau\ (fst\ x)+f(S_1\ x)*\tau\ (snd\ x)) \leq 1.25 * sqrt\ 2 * g\text{-}norm\ f\hat{\ }2$
(**is** $?L \leq ?R$)
⟨*proof*⟩

lemma *hoory-8-7*:
fixes $f :: int \times int \Rightarrow complex$
assumes $f\ (0,0) = 0$
assumes *periodic f*
shows $norm(g\text{-}inner\ f\ (\lambda x.\ f\ (S_2\ x) * (1+\omega_F\ (fst\ x)) + f\ (S_1\ x) * (1+\omega_F\ (snd\ x))))$
$\leq (2.5 * sqrt\ 2) * (\sum v \in verts\ G.\ norm\ (f\ v)\hat{\ }2)$ (**is** $?L \leq ?R$)

⟨*proof*⟩

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

Inequality stated before Theorem 8.3 in Hoory.

lemma *mgg-numerical-radius-aux*:
  assumes *g-inner f* (λ-. *1*) = *0*
  shows |(∑ *a* ∈ *arcs G*. *f* (*head G a*) ∗ *f* (*tail G a*))| ≤ (*5* ∗ *sqrt 2*) ∗ *g-norm f*^*2* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

definition *MGG-bound* :: *real*
  where *MGG-bound* = *5* ∗ *sqrt 2* / *8*

Main result: Theorem 8.2 in Hoory.

lemma *mgg-numerical-radius*: Λ_a ≤ *MGG-bound*
⟨*proof*⟩

end

end

9 Random Walks

theory *Expander-Graphs-Walks*
  imports
    Expander-Graphs-Algebra
    Expander-Graphs-Eigenvalues
    Expander-Graphs-TTS
    Constructive-Chernoff-Bound
begin

unbundle *intro-cong-syntax*

no-notation *Matrix.vec-index* (**infixl** ‹$› *100*)
hide-const *Matrix.vec-index*
hide-const *Matrix.vec*
no-notation *Matrix.scalar-prod* (**infix** ‹·› *70*)

fun *walks'* :: (*'a*,*'b*) *pre-digraph* ⇒ *nat* ⇒ (*'a list*) *multiset*
  where
    walks' G 0 = *image-mset* (λ*x*. [*x*]) (*mset-set* (*verts G*)) |
    walks' G (*Suc n*) =
      concat-mset {#{#*w* @[*z*].*z*∈# *vertices-from G* (*last w*)#}. *w* ∈# *walks' G n*#}

definition *walks G l* = (*case l of 0* ⇒ {#[]#} | *Suc pl* ⇒ *walks' G pl*)

lemma *Union-image-mono*: (⋀*x*. *x* ∈ *A* ⟹ *f x* ⊆ *g x*) ⟹ ⋃ (*f* ‘ *A*) ⊆ ⋃ (*g* ‘ *A*)
  ⟨*proof*⟩

context *fin-digraph*
begin

lemma *count-walks'*:
assumes *set xs* ⊆ *verts G*
assumes *length xs* = *l+1*
shows *count* (*walks' G l*) *xs* = (∏ *i* ∈ {..<*l*}. *count* (*edges G*) (*xs* ! *i*, *xs* ! (*i+1*)))
⟨*proof*⟩

lemma *count-walks*:
assumes *set xs* ⊆ *verts G*
assumes *length xs* = *l l* > *0*
shows *count* (*walks G l*) *xs* = (∏ *i* ∈ {..<*l*−*1*}. *count* (*edges G*) (*xs* ! *i*, *xs* ! (*i+1*)))
⟨*proof*⟩

lemma *set-walks'*:
set-mset (*walks' G l*) ⊆ {*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = (*l+1*)}
⟨*proof*⟩

lemma *set-walks*:
set-mset (*walks G l*) ⊆ {*xs*. *set xs* ⊆ *verts G* ∧ *length xs* = *l*}
⟨*proof*⟩

lemma *set-walks-2*:
assumes *xs* ∈# *walks' G l*
shows *set xs* ⊆ *verts G xs* ≠ []
⟨*proof*⟩

lemma *set-walks-3*:
assumes *xs* ∈# *walks G l*
shows *set xs* ⊆ *verts G length xs* = *l*
⟨*proof*⟩
end

lemma *measure-pmf-of-multiset*:
assumes *A* ≠ {#}
shows *measure* (*pmf-of-multiset A*) *S* = *real* (*size* (*filter-mset* (λ*x*. *x* ∈ *S*) *A*)) / *size A*
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *pmf-of-multiset-image-mset*:
assumes *A* ≠ {#}
shows *pmf-of-multiset* (*image-mset f A*) = *map-pmf f* (*pmf-of-multiset A*)
⟨*proof*⟩

context *regular-graph*
begin

lemma *size-walks'*:
size (*walks' G l*) = *card* (*verts G*) ∗ *d*^*l*
⟨*proof*⟩

lemma *size-walks*:
size (*walks G l*) = (*if l* > *0 then n* ∗ *d*^(*l*−*1*) *else 1*)
⟨*proof*⟩

lemma *walks-nonempty*:
walks G l ≠ {#}
⟨*proof*⟩

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*⟩

lemma *spec-bound*:
spec-bound A Λ_a
⟨*proof*⟩

A spectral expansion rule that does not require orthogonality of the vector for the stationary distribution:

lemma *expansionD3*:
|*g-inner f* (*g-step f*)| ≤ Λ_a ∗ *g-norm f*^*2* + (*1*−Λ_a) ∗ *g-inner f* (λ-. *1*)^*2* / *n* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

definition *ind-mat* **where** *ind-mat S* = *diag* (*ind-vec* (*enum-verts* −‘ *S*))

lemma *walk-distr*:
measure (*pmf-of-multiset* (*walks G l*)) {ω. (∀ *i*<*l*. ω ! *i* ∈ *S i*)} =
foldl (λ*x M*. *M* ∗*v x*) *stat* (*intersperse A* (*map* (λ*i*. *ind-mat* (*S i*)) [*0*..<*l*]))·*1*
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *hitting-property*:
assumes *S* ⊆ *verts G*
assumes *I* ⊆ {..<*l*}
defines μ ≡ *real* (*card S*) / *card* (*verts G*)
shows *measure* (*pmf-of-multiset* (*walks G l*)) {*w*. *set* (*nths w I*) ⊆ *S*} ≤ (μ+Λ_a∗(*1*−μ))^*card I*
(**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *uniform-property*:
assumes *i* < *l x* ∈ *verts G*
shows *measure* (*pmf-of-multiset* (*walks G l*)) {*w*. *w* ! *i* = *x*} = *1*/*real* (*card* (*verts G*))
(**is** *?L* = *?R*)
⟨*proof*⟩

end

context *regular-graph*
begin

lemmas *expansionD3* =
regular-graph-tts.expansionD3[*OF eg-tts-1*,
internalize-sort 'n :: finite, *OF - regular-graph-axioms*,
unfolded remove-finite-premise, *cancel-type-definition*, *OF verts-non-empty*]

lemmas *g-step-remains-orth* =
regular-graph-tts.g-step-remains-orth[*OF eg-tts-1*,
internalize-sort 'n :: finite, *OF - regular-graph-axioms*,
unfolded remove-finite-premise, *cancel-type-definition*, *OF verts-non-empty*]

lemmas *hitting-property* =

regular-graph-tts.hitting-property[*OF eg-tts-1*,
internalize-sort 'n :: finite, *OF - regular-graph-axioms*,
unfolded remove-finite-premise, *cancel-type-definition*, *OF verts-non-empty*]

lemmas *uniform-property-2* =
regular-graph-tts.uniform-property[*OF eg-tts-1*,
internalize-sort 'n :: finite, *OF - regular-graph-axioms*,
unfolded remove-finite-premise, *cancel-type-definition*, *OF verts-non-empty*]

theorem *uniform-property*:
assumes $i < l$
shows *map-pmf* ($\lambda w.\ w\ !\ i$) (*pmf-of-multiset* (*walks G l*)) = *pmf-of-set* (*verts G*) (**is** *?L = ?R*)
⟨*proof*⟩

lemma *uniform-property-gen*:
fixes *S :: 'a set*
assumes $S \subseteq$ *verts G* $i < l$
defines $\mu \equiv$ *real* (*card S*) / *card* (*verts G*)
shows *measure* (*pmf-of-multiset* (*walks G l*)) $\{w.\ w\ !\ i \in S\} = \mu$ (**is** *?L = ?R*)
⟨*proof*⟩

theorem *kl-chernoff-property*:
assumes $l > 0$
assumes $S \subseteq$ *verts G*
defines $\mu \equiv$ *real* (*card S*) / *card* (*verts G*)
assumes $\gamma \le 1$ $\mu + \Lambda_a * (1-\mu) \in \{0<..\gamma\}$
shows *measure* (*pmf-of-multiset* (*walks G l*)) {*w. real* (*card* $\{i \in \{..<l\}.\ w\ !\ i \in S\}) \ge \gamma * l\}$
$\le$ *exp* ($-$ *real l* $*$ *KL-div* γ ($\mu+\Lambda_a*(1-\mu)$)) (**is** *?L* $\le$ *?R*)
⟨*proof*⟩

end

unbundle *no intro-cong-syntax*

end

10 Graph Powers

theory *Expander-Graphs-Power-Construction*
imports
Expander-Graphs-Walks
Graph-Theory.Arc-Walk
begin

unbundle *intro-cong-syntax*

fun *is-arc-walk* :: (*'a*, *'b*) *pre-digraph* ⇒ *'a* ⇒ *'b list* ⇒ *bool*
where
is-arc-walk G - [] = *True* |
is-arc-walk G y (*x#xs*) = (*is-arc-walk G* (*head G x*) *xs* ∧ *tail G x* = *y* ∧ *x* ∈ *arcs G*)

definition *arc-walk-head* :: (*'a*, *'b*) *pre-digraph* ⇒ (*'a* × *'b list*) ⇒ *'a*
where
arc-walk-head G x = (*if snd x* = [] *then fst x else head G* (*last* (*snd x*)))

lemma *is-arc-walk-snoc*:
is-arc-walk G y (*xs*@[*x*]) ⟷ *is-arc-walk G y xs* ∧ *x* ∈ *out-arcs G* (*arc-walk-head G* (*y*,*xs*))

⟨*proof*⟩

lemma *is-arc-walk-set*:
assumes *is-arc-walk G u w*
shows *set w* ⊆ *arcs G*
⟨*proof*⟩

lemma (**in** *wf-digraph*) *awalk-is-arc-walk*:
assumes *u* ∈ *verts G*
shows *is-arc-walk G u w* ⟷ *awalk u w* (*awlast u w*)
⟨*proof*⟩

definition *arc-walks* :: (*'a*, *'b*) *pre-digraph* ⇒ *nat* ⇒ (*'a* × *'b list*) *set*
where
arc-walks G l = {(*u*,*w*). *u* ∈ *verts G* ∧ *is-arc-walk G u w* ∧ *length w* = *l*}

lemma *arc-walks-len*:
assumes *x* ∈ *arc-walks G l*
shows *length* (*snd x*) = *l*
⟨*proof*⟩

lemma (**in** *wf-digraph*) *awhd-of-arc-walk*:
assumes *w* ∈ *arc-walks G l*
shows *awhd* (*fst w*) (*snd w*) = *fst w*
⟨*proof*⟩

lemma (**in** *wf-digraph*) *awlast-of-arc-walk*:
assumes *w* ∈ *arc-walks G l*
shows *awlast* (*fst w*) (*snd w*) = *arc-walk-head G w*
⟨*proof*⟩

lemma (**in** *wf-digraph*) *arc-walk-head-wellformed*:
assumes *w* ∈ *arc-walks G l*
shows *arc-walk-head G w* ∈ *verts G*
⟨*proof*⟩

lemma (**in** *wf-digraph*) *arc-walk-tail-wellformed*:
assumes *w* ∈ *arc-walks G l*
shows *fst w* ∈ *verts G*
⟨*proof*⟩

lemma (**in** *fin-digraph*) *arc-walks-fin*:
finite (*arc-walks G l*)
⟨*proof*⟩

lemma (**in** *wf-digraph*) *awalk-verts-unfold*:
assumes *w* ∈ *arc-walks G l*
shows *awalk-verts* (*fst w*) (*snd w*) = *fst w*#*map* (*head G*) (*snd w*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *arc-walks-map-walks'*:
walks' G l = *image-mset* (*case-prod awalk-verts*) (*mset-set* (*arc-walks G l*))
⟨*proof*⟩

lemma (**in** *fin-digraph*) *arc-walks-map-walks*:
walks G (*l*+*1*) = *image-mset* (*case-prod awalk-verts*) (*mset-set* (*arc-walks G l*))
⟨*proof*⟩

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*⟩

definition *graph-power* :: ('*a*, '*b*) *pre-digraph* ⇒ *nat* ⇒ ('*a*, ('*a* × '*b list*)) *pre-digraph*
  where *graph-power G l* =
    (| *verts* = *verts G*, *arcs* = *arc-walks G l*, *tail* = *fst*, *head* = *arc-walk-head G* |)

lemma (**in** *wf-digraph*) *graph-power-wf*:
  wf-digraph (*graph-power G l*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *graph-power-fin*:
  fin-digraph (*graph-power G l*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *graph-power-count-edges*:
  fixes *l v w*
  defines $S \equiv \{x.\ length\ x{=}l{+}1 \wedge set\ x{\subseteq}verts\ G \wedge hd\ x{=}v \wedge last\ x{=}w\}$
  shows $count\ (edges\ (graph\text{-}power\ G\ l))\ (v,w) = (\sum x \in S.(\prod i{<}l.\ count(edges\ G)(x!i,x!(i{+}1))))$
    (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *graph-power-sym-aux*:
  assumes *symmetric-multi-graph G*
  assumes *v* ∈ *verts* (*graph-power G l*) *w* ∈ *verts* (*graph-power G l*)
  shows *card* (*arcs-betw* (*graph-power G l*) *v w*) = *card* (*arcs-betw* (*graph-power G l*) *w v*)
    (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *graph-power-sym*:
  assumes *symmetric-multi-graph G*
  shows *symmetric-multi-graph* (*graph-power G l*)
⟨*proof*⟩

lemma (**in** *fin-digraph*) *graph-power-out-degree'*:
  assumes *reg*: ⋀*v*. *v* ∈ *verts G* ⟹ *out-degree G v* = *d*
  assumes *v* ∈ *verts* (*graph-power G l*)
  shows *out-degree* (*graph-power G l*) *v* = *d* ^ *l* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *regular-graph*) *graph-power-out-degree*:
  assumes *v* ∈ *verts* (*graph-power G l*)
  shows *out-degree* (*graph-power G l*) *v* = *d* ^ *l* (**is** *?L* = *?R*)
  ⟨*proof*⟩

lemma (**in** *regular-graph*) *graph-power-regular*:
  regular-graph (*graph-power G l*)
⟨*proof*⟩

lemma (**in** *regular-graph*) *graph-power-degree*:
  regular-graph.d (*graph-power G l*) = *d*^*l* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma (**in** *regular-graph*) *graph-power-step*:
  assumes *x* ∈ *verts G*
  shows *regular-graph.g-step* (*graph-power G l*) *f x* = (*g-step*^^*l*) *f x*

⟨*proof*⟩

lemma (**in** *regular-graph*) *graph-power-expansion*:
  regular-graph.Λ_a (*graph-power G l*) ≤ Λ_a ^ *l*
⟨*proof*⟩

unbundle *no intro-cong-syntax*

end

11 Strongly Explicit Expander Graphs

In some applications, representing an expander graph using a data structure (for example as an adjacency lists) would be prohibitive. For such cases strongly explicit expander graphs (SEE) are relevant. These are expander graphs, which can be represented implicitly using a function that computes for each vertex its neighbors in space and time logarithmic w.r.t. to the size of the graph. An application can for example sample a random walk, from a SEE using such a function efficiently. An example of such a graph is the Margulis construction from Section 8. This section presents the latter as a SEE but also shows that two graph operations that preserve the SEE property, in particular the graph power construction from Section 10 and a compression scheme introduced by Murtagh et al. [9, Theorem 20]. Combining all of the above it is possible to construct strongly explicit expander graphs of *every size* and spectral gap.

theory *Expander-Graphs-Strongly-Explicit*
  imports *Expander-Graphs-Power-Construction Expander-Graphs-MGG*
begin

unbundle *intro-cong-syntax*
no-notation *Digraph.dominates* (‹- →ı -› [*100*,*100*] *40*)

record *strongly-explicit-expander* =
  see-size :: *nat*
  see-degree :: *nat*
  see-step :: *nat* ⇒ *nat* ⇒ *nat*

definition *graph-of* :: *strongly-explicit-expander* ⇒ (*nat*, (*nat*,*nat*) *arc*) *pre-digraph*
  where *graph-of e* =
   ⦇ *verts* = {..<*see-size e*},
    arcs = (λ(*v*, *i*). *Arc v* (*see-step e i v*) *i*) ‘ ({..<*see-size e*} × {..<*see-degree e*}),
    tail = *arc-tail*,
    head = *arc-head* ⦈

definition *is-expander e* Λ_a ⟷
  regular-graph (*graph-of e*) ∧ *regular-graph*.Λ_a (*graph-of e*) ≤ Λ_a

lemma *is-expander-mono*:
  assumes *is-expander e a a* ≤ *b*
  shows *is-expander e b*
  ⟨*proof*⟩

lemma *graph-of-finI*:
  assumes *see-step e* ∈ ({..<*see-degree e*} → ({..<*see-size e*} → {..<*see-size e*}))
  shows *fin-digraph* (*graph-of e*)
⟨*proof*⟩

lemma *edges-graph-of*:
edges(*graph-of e*)={#(*v*,*see-step e i v*). (*v*,*i*)∈#*mset-set* ({..<*see-size e*}×{..<*see-degree e*})#}
⟨*proof*⟩

lemma *out-degree-see*:
assumes *v* ∈ *verts* (*graph-of e*)
shows *out-degree* (*graph-of e*) *v* = *see-degree e* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *card-arc-walks-see*:
assumes *fin-digraph* (*graph-of e*)
shows *card* (*arc-walks* (*graph-of e*) *n*) = *see-degree e*^*n* ∗ *see-size e* (**is** *?L* = *?R*)
⟨*proof*⟩

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*⟩

The following introduces the compression scheme, described in [9, Theorem 20].

fun *see-compress* :: *nat* ⇒ *strongly-explicit-expander* ⇒ *strongly-explicit-expander*
where *see-compress m e* =
(| *see-size* = *m*, *see-degree* = *see-degree e* ∗ *2*
, *see-step* = (λ*k v*.
if k < *see-degree e*
then (*see-step e k v*) *mod m*
else (*if v*+*m* < *see-size e then* (*see-step e* (*k*−*see-degree e*) (*v*+*m*)) *mod m else v*)) |)

lemma *edges-of-compress*:
fixes *e m*
assumes *2*∗*m* ≥ *see-size e m* ≤ *see-size e*
defines *A* ≡ {# (*x mod m*, *y mod m*). (*x*,*y*) ∈# *edges* (*graph-of e*)#}
defines *B* ≡ *repeat-mset* (*see-degree e*) {# (*x*,*x*). *x* ∈# (*mset-set* {*see-size e* − *m*..<*m*})#}
shows *edges* (*graph-of* (*see-compress m e*)) = *A* + *B* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *see-compress-sym*:
assumes *2*∗*m* ≥ *see-size e m* ≤ *see-size e*
assumes *symmetric-multi-graph* (*graph-of e*)
shows *symmetric-multi-graph* (*graph-of* (*see-compress m e*))
⟨*proof*⟩

lemma *see-compress*:
assumes *is-expander e* Λ_a
assumes *2*∗*m* ≥ *see-size e m* ≤ *see-size e*
shows *is-expander* (*see-compress m e*) (Λ_a/*2* + *1*/*2*)
⟨*proof*⟩

The graph power of a strongly explicit expander graph is itself a strongly explicit expander graph.

fun *to-digits* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat list*
where
to-digits - *0* - = [] |
to-digits b (*Suc l*) *k* = (*k mod b*)# *to-digits b l* (*k div b*)

fun *from-digits* :: *nat* ⇒ *nat list* ⇒ *nat*
where
from-digits b [] = *0* |

from-digits b (x#xs) = x + b ∗ from-digits b xs

lemma *to-from-digits*:
assumes *length xs = n set xs ⊆ {..<b}*
shows *to-digits b n (from-digits b xs) = xs*
⟨*proof*⟩

lemma *from-digits-range*:
assumes *length xs = n set xs ⊆ {..<b}*
shows *from-digits b xs < b^n*
⟨*proof*⟩

lemma *from-digits-inj*:
inj-on (from-digits b) {xs. set xs ⊆ {..<b} ∧ length xs = n}
⟨*proof*⟩

fun *see-power :: nat ⇒ strongly-explicit-expander ⇒ strongly-explicit-expander*
where *see-power l e =*
⦇ see-size = see-size e, see-degree = see-degree e^l
, see-step = (λk v. foldl (λy x. see-step e x y) v (to-digits (see-degree e) l k)) ⦈

lemma *graph-power-iso-see-power*:
assumes *fin-digraph (graph-of e)*
shows *digraph-iso (graph-power (graph-of e) n) (graph-of (see-power n e))*
⟨*proof*⟩

lemma *see-power*:
assumes *is-expander e* Λ_a
shows *is-expander (see-power n e)* (Λ_a*^n*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

definition *see-mgg :: nat ⇒ strongly-explicit-expander* **where**
see-mgg n = ⦇ see-size = n^2, 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*⟩

lemma *see-mgg*:
  assumes $n > 0$
  shows *is-expander* (*see-mgg n*) (*5∗ sqrt 2 / 8*)
⟨*proof*⟩

Using all of the above it is possible to construct strongly explicit expanders of every size and spectral gap with asymptotically optimal degree.

definition *see-standard-aux*
  where *see-standard-aux n* = *see-compress n* (*see-mgg* (*nat* ⌈*sqrt n*⌉))

lemma *see-standard-aux*:
  assumes $n > 0$
  shows
    is-expander (*see-standard-aux n*) ((*8+5 ∗ sqrt 2*) / *16*) (**is** *?A*)
    see-degree (*see-standard-aux n*) = *16* (**is** *?B*)
    see-size (*see-standard-aux n*) = *n* (**is** *?C*)
⟨*proof*⟩

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 $\Lambda_a > 0$
  shows *0.95^*(*see-standard-power* Λ_a) ≤ Λ_a (**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *see-standard-power-eval*[*code*]:
  see-standard-power x = (*if x* ≤ *0* ∨ *x* ≥ *1 then 0 else* (*1+see-standard-power* (*x/0.95*)))
⟨*proof*⟩

definition *see-standard* :: *nat* ⇒ *real* ⇒ *strongly-explicit-expander*
  where *see-standard n* Λ_a = *see-power* (*see-standard-power* Λ_a) (*see-standard-aux n*)

theorem *see-standard*:
  assumes $n > 0$ $\Lambda_a > 0$
  shows *is-expander* (*see-standard n* Λ_a) Λ_a
    and *see-size* (*see-standard n* Λ_a) = *n*
    and *see-degree* (*see-standard n* Λ_a) = *16* ^ (*nat* ⌈*ln* Λ_a / *ln 0.95*⌉) (**is** *?C*)
⟨*proof*⟩

fun *see-sample-walk* :: *strongly-explicit-expander* ⇒ *nat* ⇒ *nat* ⇒ *nat list*
  where
    see-sample-walk e 0 x = [*x*] |
    see-sample-walk e (*Suc l*) *x* = (*let w* = *see-sample-walk e l* (*x div* (*see-degree e*)) *in*
      w@[*see-step e* (*x mod* (*see-degree e*)) (*last w*)])

theorem *see-sample-walk*:
  fixes *e l*
  assumes *fin-digraph* (*graph-of e*)
  defines *r* ≡ *see-size e* ∗ *see-degree e ^l*
  shows {# *see-sample-walk e l k*. *k* ∈# *mset-set* {..<*r*} #} = *walks′* (*graph-of e*) *l*
  ⟨*proof*⟩

unbundle *no intro-cong-syntax*

end

12 Expander Walks as Pseudorandom Objects

theory *Pseudorandom-Objects-Expander-Walks*
  imports
    Universal-Hash-Families.Pseudorandom-Objects
    Expander-Graphs.Expander-Graphs-Strongly-Explicit
begin

unbundle *intro-cong-syntax*
hide-const (**open**) *Quantum.T*
hide-fact (**open**) *SN-Orders.of-nat-mono*
hide-fact *Missing-Ring.mult-pos-pos*

definition *expander-pro* ::
  nat ⇒ *real* ⇒ (′*a*,′*b*) *pseudorandom-object-scheme* ⇒ (*nat* ⇒ ′*a*) *pseudorandom-object*
  where *expander-pro l* Λ *S* = (
    let e = *see-standard* (*pro-size S*) Λ *in*
      (| *pro-last* = *see-size e* ∗ *see-degree e*^(*l*−*1*) − *1*,
        pro-select = (λ*i j*. *pro-select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j mod pro-size S*)) |)
)

context
  fixes *l* :: *nat*
  fixes Λ :: *real*
  fixes *S* :: (′*a*,′*b*) *pseudorandom-object-scheme*
  assumes *l-gt-0*: *l* > *0*
  assumes Λ*-gt-0*: Λ > *0*
begin

private definition *e* **where** *e* = *see-standard* (*pro-size S*) Λ

private lemma *expander-pro-alt*: *expander-pro l* Λ *S* = (| *pro-last* = *see-size e* ∗ *see-degree e*^(*l*−*1*) − *1*,
        pro-select = (λ*i j*. *pro-select S* (*see-sample-walk e* (*l*−*1*) *i* ! *j mod pro-size S*)) |)
  ⟨*proof*⟩ **lemmas** *see-standard* = *see-standard* [*OF pro-size-gt-0*[**where** *S*=*S*] Λ*-gt-0*]

interpretation *E*: *regular-graph graph-of e*
  ⟨*proof*⟩ **lemma** *e-deg-gt-0*: *see-degree e* > *0*
  ⟨*proof*⟩ **lemma** *e-size-gt-0*: *see-size e* > *0*
  ⟨*proof*⟩ **lemma** *expander-sample-size*: *pro-size* (*expander-pro l* Λ *S*) = *see-size e* ∗ *see-degree e*^(*l*−*1*)
  ⟨*proof*⟩ **lemma** *sample-pro-expander-walks*:
  defines *R* ≡ *map-pmf* (λ*xs i*. *pro-select S* (*xs* ! *i mod pro-size S*))
    (*pmf-of-multiset* (*walks* (*graph-of e*) *l*))
  shows *sample-pro* (*expander-pro l* Λ *S*) = *R*
⟨*proof*⟩

lemma *expander-pro-range*: *pro-select* (*expander-pro l* Λ *S*) *i j* ∈ *pro-set S*
  ⟨*proof*⟩

lemma *expander-uniform-property*:
  assumes *i* < *l*
  shows *map-pmf* (λ*w*. *w i*) (*sample-pro* (*expander-pro l* Λ *S*)) = *sample-pro S* (**is** *?L* = *?R*)

⟨*proof*⟩

lemma *expander-kl-chernoff-bound*:
assumes *measure* (*sample-pro S*) {*w. T w*} ≤ *μ*
assumes *γ* ≤ *1* *μ* + Λ ∗ (*1*−*μ*) ≤ *γ* *μ* ≤ *1*
shows *measure* (*sample-pro* (*expander-pro l* Λ *S*)) {*w. real* (*card* {*i* ∈ {..<*l*}. *T* (*w i*)}) ≥ *γ*∗*l*}
≤ *exp* (− *real l* ∗ *KL-div γ* (*μ* + Λ∗(*1*−*μ*))) (**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *expander-chernoff-bound-one-sided*:
assumes *AE x in sample-pro S*. *f x* ∈ {*0*,*1*::*real*}
assumes (∫ *x*. *f x* ∂*sample-pro S*) ≤ *μ* *l* > *0* *γ* ≥ *0*
shows *measure* (*expander-pro l* Λ *S*) {*w*. (∑ *i*<*l*. *f* (*w i*))/*l*−*μ*≥*γ*+Λ} ≤ *exp* (− *2* ∗ *real l* ∗ *γ*^*2*)
(**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *expander-chernoff-bound*:
assumes *AE x in sample-pro S*. *f x* ∈ {*0*,*1*::*real*} *l* > *0* *γ* ≥ *0*
defines *μ* ≡ (∫ *x*. *f x* ∂*sample-pro S*)
shows *measure* (*expander-pro l* Λ *S*) {*w*. |(∑ *i*<*l*. *f* (*w i*))/*l*−*μ*|≥*γ*+Λ} ≤ *2*∗*exp* (− *2* ∗ *real l* ∗ *γ*^*2*)
(**is** *?L* ≤ *?R*)
⟨*proof*⟩

lemma *expander-pro-size*:
pro-size (*expander-pro l* Λ *S*) = *pro-size S* ∗ (*16* ^((*l*−*1*) ∗ *nat* ⌈*ln* Λ / *ln* (*19* / *20*)⌉))
(**is** *?L* = *?R*)
⟨*proof*⟩

end

open-bundle *expander-pseudorandom-object-syntax*
begin
notation *expander-pro* (‹ℰ›)
end

unbundle *no intro-cong-syntax*

end

References

[1] J. Divasón, O. Kunar, R. Thiemann, and A. Yamada. Perron-frobenius theorem for spectral radius analysis. *Archive of Formal Proofs*, May 2016. https://isa-afp.org/entries/Perron_Frobenius.html, Formal proof development.

[2] M. Echenim. Simultaneous diagonalization of pairwise commuting hermitian matrices. *Archive of Formal Proofs*, July 2022. https://isa-afp.org/entries/Commuting_Hermitian.html, Formal proof development.

[3] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

[4] S. Hoory and N. Linial. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[5] R. Impagliazzo and V. Kabanets. Constructive proofs of concentration bounds. In M. Serna, R. Shaltiel, K. Jansen, and J. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 617–631, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4), 1987.

[7] O. Kuncar and A. Popescu. From types to sets by local type definition in higher-order logic. *Journal of Automated Reasoning*, 62:237 – 260, 2016.

[8] G. A. Margulis. Explicit construction of a concentrator. *Probl. Peredachi Inf.*, 9(4):71–80, 1973.

[9] J. Murtagh, O. Reingold, A. Sidford, and S. Vadhan. Deterministic Approximation of Random Walks in Small Space. In D. Achlioptas and L. A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:22, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] L. Noschinski. Graph theory. *Archive of Formal Proofs*, April 2013. https://isa-afp.org/entries/Graph_Theory.html, Formal proof development.

[11] S. P. Vadhan. Pseudorandomness. *Foundations and Trends(R) in Theoretical Computer Science*, 7(13):1–336, 2012.