Roth's Theorem on Arithmetic Progressions

Chelsea Edmonds, Angeliki Koutsoukou-Argyraki and Lawrence C. Paulson
Computer Laboratory, University of Cambridge CB3 0FD
{cle47,ak2110,lp15}@cam.ac.uk

January 3, 2022

Abstract

We formalise a proof of Roth's Theorem on Arithmetic Progressions, a major result in additive combinatorics on the existence of 3-term arithmetic progressions in subsets of natural numbers. To this end, we follow a proof using graph regularity. We employ our recent formalisation of Szemerédi's Regularity Lemma, a major result in extremal graph theory, which we use here to prove the Triangle Counting Lemma and the Triangle Removal Lemma. Our sources are Yufei Zhao's MIT lecture notes "Graph Theory and Additive Combinatorics"[1] and W.T. Gowers's Cambridge lecture notes "Topics in Combinatorics".[2] We also refer to the University of Georgia notes by Stephanie Bell and Will Grodzicki "Using Szemerédi's Regularity Lemma to Prove Roth's Theorem".[3]

Contents

Acknowledgements

The authors were supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council.

[1] https://ocw.mit.edu/courses/mathematics/18-217-graph-theory-and-additive-combinatorics-fall-2019/lecture-notes/MIT18_217F19_ch3.pdf and https://yufeizhao.com/gtac/gtac17.pdf

[2] https://www.dpmms.cam.ac.uk/~par31/notes/tic.pdf

[3] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.432.327

1 Roth's Theorem on Arithmetic Progressions

theory *Roth-Arithmetic-Progressions*
imports *Szemeredi-Regularity.Szemeredi*
Random-Graph-Subgraph-Threshold.Subgraph-Threshold
Ergodic-Theory.Asymptotic-Density
HOL−Library.Ramsey HOL−Library.Nat-Bijection

begin

1.1 For the Library

declare *prod-encode-eq* [*simp*]
declare *prod-decode-eq* [*simp*]

lemma *mult-mod-cancel-right*:
fixes m :: *'a*::{*euclidean-ring-cancel*,*semiring-gcd*}
assumes *eq*: $(a * n)$ *mod* $m = (b * n)$ *mod* m **and** *coprime* m n
shows a *mod* $m = b$ *mod* m
proof −
have m *dvd* $(a*n - b*n)$
using *eq mod-eq-dvd-iff* **by** *blast*
then have m *dvd* $a-b$
by (*metis* ‹*coprime m n*› *coprime-dvd-mult-left-iff left-diff-distrib'*)
then show *?thesis*
using *mod-eq-dvd-iff* **by** *blast*
qed

lemma *mult-mod-cancel-left*:
fixes m :: *'a*::{*euclidean-ring-cancel*,*semiring-gcd*}
assumes $(n * a)$ *mod* $m = (n * b)$ *mod* m **and** *coprime* m n
shows a *mod* $m = b$ *mod* m
by (*metis assms mult.commute mult-mod-cancel-right*)

lemma *edge-density-le1*: *edge-density X Y G* $\leq$ *1*
proof (*cases finite X* $\wedge$ *finite Y*)
case *True*
then show *?thesis*
using *of-nat-mono* [*OF max-all-edges-between*, *of X Y*]
by (*fastforce simp add*: *edge-density-def divide-simps*)
qed (*auto simp*: *edge-density-def*)

lemma *card-3-iff*: *card S* = *3* $\longleftrightarrow$ ($\exists$ *x y z*. *S* = {*x*,*y*,*z*} $\wedge$ *x* $\neq$ *y* $\wedge$ *y* $\neq$ *z* $\wedge$ *x* $\neq$ *z*)
by (*fastforce simp*: *card-Suc-eq numeral-eq-Suc*)

1.2 Miscellaneous Preliminaries

lemma *sum-prod-le-prod-sum*:

fixes $a :: 'a \Rightarrow 'b$::*linordered-idom*
assumes $\bigwedge i.\ i \in I \Longrightarrow a\ i \geq 0 \wedge b\ i \geq 0$
shows $(\sum i{\in}I.\ \sum j{\in}I.\ a\ i * b\ j) \leq (\sum i{\in}I.\ a\ i) * (\sum i{\in}I.\ b\ i)$
using *assms*
by (*induction I rule*: *infinite-finite-induct*) (*auto simp add*: *algebra-simps sum.distrib sum-distrib-left*)

lemma *real-mult-gt-cube*: $A \geq (X :: real) \Longrightarrow B \geq X \Longrightarrow C \geq X \Longrightarrow X \geq 0 \Longrightarrow A * B * C \geq X\hat{}3$
by (*simp add*: *mult-mono'* *power3-eq-cube*)

lemma *min-card-fin-X-elem*: *finite* $X \Longrightarrow x \in X \Longrightarrow card\ X \geq 1$
using *card.remove* **by** *fastforce*

lemma *card-or-filter-max*:
assumes *finite A*
shows *card* $\{a \in A\ .\ P\ a \vee Q\ a\} \leq card\ \{a \in A\ .\ P\ a\} + card\ \{a \in A\ .\ Q\ a\}$
proof −
have *fin*: *finite* $\{a \in A\ .\ P\ a\}$ *finite* $\{a \in A\ .\ Q\ a\}$
by(*simp-all add*: *assms*)
have *equiv*: $\{a \in A\ .\ P\ a \vee Q\ a\} = \{a \in A\ .\ P\ a\} \cup \{a \in A\ .\ Q\ a\}$ **by** *auto*
then have *card* $\{a \in A\ .\ P\ a\} + card\ \{a \in A\ .\ Q\ a\} = card\ (\{a \in A\ .\ P\ a\} \cup \{a \in A\ .\ Q\ a\}) + card\ (\{a \in A\ .\ P\ a\} \cap \{a \in A\ .\ Q\ a\})$
using *card-Un-Int fin* **by** *auto*
thus *?thesis* **using** *equiv*
by *presburger*
qed

lemma *triple-sigma-rewrite-card*:
assumes *finite X finite Y finite Z*
shows *card* $\{(x,\ y,\ z)\ .\ x \in X \wedge (y,\ z) \in Y \times Z \wedge P\ x\ y\ z\} = (\sum x{\in}\ X\ .\ card\ \{(y,z) \in Y \times Z.\ P\ x\ y\ z\})$
proof −
define W **where** $W \equiv \lambda x.\ \{(y,z) \in Y \times Z.\ P\ x\ y\ z\}$
have $W\ x \subseteq Y \times Z$ **for** x
by (*auto simp*: *W-def*)
then have [*simp*]: *finite* $(W\ x)$ **for** x
by (*meson assms finite-SigmaI infinite-super*)
have *eq*: $\{(x,\ y,\ z)\ .\ x \in X \wedge (y,\ z) \in Y \times Z \wedge P\ x\ y\ z\} = (\bigcup x{\in}X.\ \bigcup (y,\ z){\in}W\ x.\ \{(x,y,z)\})$
by (*auto simp*: *W-def*)
show *?thesis*
unfolding *eq* **by** (*simp add*: *disjoint-iff assms card-UN-disjoint*) (*simp add*: *W-def*)
qed

lemma *all-edges-between-Union1*:
all-edges-between (*Union* $\mathcal{X}$) $Y\ G = (\bigcup X{\in}\mathcal{X}.\ all\text{-}edges\text{-}between\ X\ Y\ G)$
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-Union2*:
all-edges-between X (Union 𝒴) G = (⋃ *Y*∈*𝒴*. *all-edges-between X Y G*)
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-disjoint1*:
assumes *disjoint R*
shows *disjoint* ((λ*X*. *all-edges-between X Y G*) ‘ *R*)
using *assms* **by** (*auto simp*: *all-edges-between-def disjoint-def*)

lemma *all-edges-between-disjoint2*:
assumes *disjoint R*
shows *disjoint* ((λ*Y*. *all-edges-between X Y G*) ‘ *R*)
using *assms* **by** (*auto simp*: *all-edges-between-def disjoint-def*)

lemma *all-edges-between-disjoint-family-on1*:
assumes *disjoint R*
shows *disjoint-family-on* (λ*X*. *all-edges-between X Y G*) *R*
by (*metis* (*no-types*, *lifting*) *all-edges-between-disjnt1 assms disjnt-def disjoint-family-on-def pairwiseD*)

lemma *all-edges-between-disjoint-family-on2*:
assumes *disjoint R*
shows *disjoint-family-on* (λ*Y*. *all-edges-between X Y G*) *R*
by (*metis* (*no-types*, *lifting*) *all-edges-between-disjnt2 assms disjnt-def disjoint-family-on-def pairwiseD*)

lemma *all-edges-between-mono1*:
Y ⊆ *Z* ⟹ *all-edges-between Y X G* ⊆ *all-edges-between Z X G*
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-mono2*:
Y ⊆ *Z* ⟹ *all-edges-between X Y G* ⊆ *all-edges-between X Z G*
by (*auto simp*: *all-edges-between-def*)

lemma *inj-on-mk-uedge*: *X* ∩ *Y* = {} ⟹ *inj-on mk-uedge* (*all-edges-between X Y G*)
by (*auto simp*: *inj-on-def doubleton-eq-iff all-edges-between-def*)

lemma *uwellformed-alt*:
assumes *uwellformed G* {*x*, *y*} ∈ *uedges G*
shows {*x*, *y*} ⊆ *uverts G*
using *uwellformed-def assms* **by** *auto*

lemma *uwellformed-alt-fst*:
assumes *uwellformed G* {*x*, *y*} ∈ *uedges G*
shows *x* ∈ *uverts G*
using *uwellformed-alt assms* **by** *simp*

lemma *uwellformed-alt-snd*:
assumes *uwellformed G* $\{x, y\} \in$ *uedges G*
shows $y \in$ *uverts G*
using *uwellformed-alt assms* **by** *simp*

lemma *all-edges-between-subset-times*: *all-edges-between X Y G* $\subseteq$ (*X* $\cap \bigcup$(*uedges G*)) $\times$ (*Y* $\cap \bigcup$(*uedges G*))
by (*auto simp*: *all-edges-between-def*)

lemma *finite-all-edges-between′*:
assumes *finite* (*uverts G*) *uwellformed G*
shows *finite* (*all-edges-between X Y G*)
proof −
have *finite* ($\bigcup$(*uedges G*))
by (*meson Pow-iff all-edges-subset-Pow assms finite-Sup subsetD wellformed-all-edges*)
with *all-edges-between-subset-times* **show** *?thesis*
by (*metis finite-Int finite-SigmaI finite-subset*)
qed

lemma *card-all-edges-between*:
assumes *finite Y finite* (*uverts G*) *uwellformed G*
shows *card* (*all-edges-between X Y G*) = ($\sum$ *y*$\in$*Y*. *card* (*all-edges-between X* $\{y\}$ *G*))
proof −
have *all-edges-between X Y G* = ($\bigcup$ *y*$\in$*Y*. *all-edges-between X* $\{y\}$ *G*)
by (*auto simp*: *all-edges-between-def*)
moreover have *disjoint-family-on* (λ*y*. *all-edges-between X* $\{y\}$ *G*) *Y*
unfolding *disjoint-family-on-def*
by (*auto simp*: *disjoint-family-on-def all-edges-between-def*)
ultimately show *?thesis*
by (*simp add*: *card-UN-disjoint′ assms finite-all-edges-between′*)
qed

lemma *max-edges-graph*:
assumes *uwellformed G finite* (*uverts G*)
shows *card* (*uedges G*) $\leq$ (*card* (*uverts G*))^2
proof −
have *card* (*uedges G*) $\leq$ *card* (*uverts G*) *choose 2*
by (*metis all-edges-finite assms card-all-edges card-mono wellformed-all-edges*)
thus *?thesis*
by (*metis binomial-le-pow le0 neq0-conv order.trans zero-less-binomial-iff*)
qed

lemma *all-edges-between-ss-uedges*: *mk-uedge* ‘ (*all-edges-between X Y G*) $\subseteq$ *uedges G*
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-betw-D1*: (*x*, *y*) $\in$ *all-edges-between X Y G* $\Longrightarrow$ *x* $\in$ *X*
by (*simp add*: *all-edges-between-def*)

lemma *all-edges-betw-D2*: $(x,\ y) \in$ *all-edges-between* $X\ Y\ G \Longrightarrow y \in Y$
by (*simp add*: *all-edges-between-def*)

lemma *all-edges-betw-D3*: $(x,\ y) \in$ *all-edges-between* $X\ Y\ G \Longrightarrow \{x,\ y\} \in$ *uedges* G
by (*simp add*: *all-edges-between-def*)

lemma *all-edges-betw-I*: $x \in X \Longrightarrow y \in Y \Longrightarrow \{x,\ y\} \in$ *uedges* $G \Longrightarrow (x,\ y) \in$ *all-edges-between* $X\ Y\ G$
by (*simp add*: *all-edges-between-def*)

lemma *all-edges-between-E-diff*:
all-edges-between $X\ Y\ (V,E-E') =$ *all-edges-between* $X\ Y\ (V,E) -$ *all-edges-between* $X\ Y\ (V,E')$
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-E-Un*:
all-edges-between $X\ Y\ (V,E \cup E') =$ *all-edges-between* $X\ Y\ (V,E) \cup$ *all-edges-between* $X\ Y\ (V,E')$
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-E-UN*:
all-edges-between $X\ Y\ (V,\ \bigcup i \in I.\ E\ i) = (\bigcup i \in I.$ *all-edges-between* $X\ Y\ (V,E\ i))$
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-betw-prod-def*: *all-edges-between* $X\ Y\ G = \{(x,\ y) \in X \times Y\ .\ \{x,\ y\} \in$ *uedges* $G\}$
by (*simp add*: *all-edges-between-def*)

thm *in-mk-uedge-img*
lemma *in-mk-uedge-img-iff*: $\{a,b\} \in$ *mk-uedge* ‘ $A \longleftrightarrow (a,b) \in A \vee (b,a) \in A$
by (*auto simp*: *doubleton-eq-iff intro*: *rev-image-eqI*)

lemma *all-edges-preserved*: $[\![$*all-edges-between* $A\ B\ G' =$ *all-edges-between* $A\ B\ G$; $X \subseteq A$; $Y \subseteq B]\!]$
$\Longrightarrow$ *all-edges-between* $X\ Y\ G' =$ *all-edges-between* $X\ Y\ G$
by (*auto simp*: *all-edges-between-def*)

lemma *subgraph-edge-wf*:
assumes *uwellformed G uverts H* = *uverts G uedges H* $\subseteq$ *uedges G*
shows *uwellformed H*
by (*metis assms subsetD uwellformed-def*)

1.3 Preliminaries on Neighbors in Graphs

definition *neighbor-in-graph*:: *uvert* $\Rightarrow$ *uvert* $\Rightarrow$ *ugraph* $\Rightarrow$ *bool*
where *neighbor-in-graph x y G* $\equiv (x \in$ (*uverts G*) $\wedge\ y \in$ (*uverts G*) $\wedge\ \{x,y\} \in$

(*uedges G*))

definition *neighbors* :: *uvert* ⇒ *ugraph* ⇒ *uvert set* **where**
neighbors x G ≡ {*y* ∈ *uverts G* . *neighbor-in-graph x y G*}

definition *neighbors-ss*:: *uvert* ⇒ *uvert set* ⇒ *ugraph* ⇒ *uvert set* **where**
neighbors-ss x Y G ≡ {*y* ∈ *Y* . *neighbor-in-graph x y G*}

lemma *all-edges-betw-prod-def-neighbors*: *uwellformed G* ⟹
all-edges-between X Y G = {(*x*, *y*) ∈ *X* × *Y* . *neighbor-in-graph x y G*}
by (*auto simp*: *neighbor-in-graph-def uwellformed-alt-fst uwellformed-alt-snd all-edges-between-def*)

lemma *all-edges-betw-sigma-neighbor*:
uwellformed G ⟹ *all-edges-between X Y G* = (*SIGMA x*:*X*. *neighbors-ss x Y G*)
by (*auto simp add*: *all-edges-between-def neighbors-ss-def neighbor-in-graph-def*
uwellformed-alt-fst uwellformed-alt-snd)

lemma *card-all-edges-betw-neighbor*:
assumes *finite X finite Y uwellformed G*
shows *card* (*all-edges-between X Y G*) = (∑ *x*∈*X*. *card* (*neighbors-ss x Y G*))
using *all-edges-betw-sigma-neighbor assms* **by** (*simp add*: *neighbors-ss-def*)

1.4 Preliminaries on Triangles in Graphs

definition *triangle-in-graph*:: *uvert* ⇒ *uvert* ⇒ *uvert* ⇒ *ugraph* ⇒ *bool*
where *triangle-in-graph x y z G*
≡ ({*x*,*y*} ∈ *uedges G*) ∧ ({*y*,*z*} ∈ *uedges G*) ∧ ({*x*,*z*} ∈ *uedges G*)

definition *triangle-triples*
where *triangle-triples X Y Z G* ≡ {(*x*,*y*,*z*) ∈ *X* × *Y* × *Z*. *triangle-in-graph x y z G*}

lemma *card-triangle-triples-rotate*: *card* (*triangle-triples X Y Z G*) = *card* (*triangle-triples Y Z X G*)
proof −
have *triangle-triples Y Z X G* = (λ(*x*,*y*,*z*). (*y*,*z*,*x*)) ‘ *triangle-triples X Y Z G*
by (*auto simp*: *triangle-triples-def case-prod-unfold image-iff insert-commute triangle-in-graph-def*)
moreover have *inj-on* (λ(*x*, *y*, *z*). (*y*, *z*, *x*)) (*triangle-triples X Y Z G*)
by (*auto simp*: *inj-on-def*)
ultimately show *?thesis*
by (*simp add*: *card-image*)
qed

lemma *triangle-commu1*:
assumes *triangle-in-graph x y z G*
shows *triangle-in-graph y x z G*
using *assms triangle-in-graph-def* **by** (*auto simp add*: *insert-commute*)

lemma *triangle-vertices-distinct1*:
assumes *wf*: *uwellformed G*
assumes *tri*: *triangle-in-graph x y z G*
shows $x \neq y$
proof (*rule ccontr*)
assume *a*: $\neg\ x \neq y$
have *card* $\{x, y\} = 2$ **using** *tri wf triangle-in-graph-def*
using *uwellformed-def* **by** *blast*
thus *False* **using** *a* **by** *simp*
qed

lemma *triangle-vertices-distinct2*:
assumes *uwellformed G triangle-in-graph x y z G*
shows $y \neq z$
by (*metis assms triangle-vertices-distinct1 triangle-in-graph-def*)

lemma *triangle-vertices-distinct3*:
assumes *uwellformed G triangle-in-graph x y z G*
shows $z \neq x$
by (*metis assms triangle-vertices-distinct1 triangle-in-graph-def*)

lemma *triangle-in-graph-edge-point*:
assumes *uwellformed G*
shows *triangle-in-graph x y z G* $\longleftrightarrow$ $\{y, z\} \in$ *uedges G* $\wedge$ *neighbor-in-graph x y G* $\wedge$ *neighbor-in-graph x z G*
by (*auto simp add*: *triangle-in-graph-def neighbor-in-graph-def assms uwellformed-alt-fst uwellformed-alt-snd*)

definition
unique-triangles G
$\equiv \forall\, e \in$ *uedges G*. $\exists!\, T.\ \exists\, x\ y\ z.\ T = \{x,y,z\} \wedge$ *triangle-in-graph x y z G* $\wedge\ e \subseteq T$

definition *triangle-free-graph*:: *ugraph* $\Rightarrow$ *bool*
where *triangle-free-graph G* $\equiv \neg(\exists\ x\ y\ z.$ *triangle-in-graph x y z G* $)$

lemma *triangle-free-graph-empty*: *uedges G* $= \{\} \Longrightarrow$ *triangle-free-graph G*
by (*simp add*: *triangle-free-graph-def triangle-in-graph-def*)

lemma *edge-vertices-not-equal*:
assumes *uwellformed G* $\{x,y\} \in$ *uedges G*
shows $x \neq y$
using *assms triangle-in-graph-def triangle-vertices-distinct1* **by** *blast*

lemma *edge-btw-vertices-not-equal*:
assumes *uwellformed G* $(x, y) \in$ *all-edges-between X Y G*
shows $x \neq y$
using *edge-vertices-not-equal all-edges-between-def*

by (*metis all-edges-betw-D3 assms*)

lemma *mk-triangle-from-ss-edges*:
assumes $(x, y) \in$ *all-edges-between X Y G* **and** $(x, z) \in$ *all-edges-between X Z G*
and $(y, z) \in$ *all-edges-between Y Z G*
shows (*triangle-in-graph x y z G*)
by (*meson all-edges-betw-D3 assms triangle-in-graph-def*)

lemma *triangle-in-graph-verts*:
assumes *uwellformed G*
assumes *triangle-in-graph x y z G*
shows $x \in$ *uverts G* $y \in$ *uverts G* $z \in$ *uverts G*
proof −
have *1*: $\{x, y\} \in$ *uedges G* **using** *triangle-in-graph-def*
using *assms*(*2*) **by** *auto*
then show $x \in$ *uverts G* **using** *uwellformed-alt-fst assms* **by** *blast*
then show $y \in$ *uverts G* **using** *1 uwellformed-alt-snd assms* **by** *blast*
have $\{x, z\} \in$ *uedges G* **using** *triangle-in-graph-def assms*(*2*) **by** *auto*
then show $z \in$ *uverts G* **using** *uwellformed-alt-snd assms* **by** *blast*
qed

definition *triangle-set* :: *ugraph* ⇒ *uvert set set*
where *triangle-set G* ≡ { $\{x,y,z\}$ | *x y z. triangle-in-graph x y z G*}

fun *mk-triangle-set* :: (*uvert* × *uvert* × *uvert*) ⇒ *uvert set*
where *mk-triangle-set* $(x, y, z) = \{x,y,z\}$

lemma *convert-triangle-rep-ss*:
fixes *G* :: *ugraph*
assumes $X \subseteq$ *uverts G* **and** $Y \subseteq$ *uverts G* **and** $Z \subseteq$ *uverts G*
shows *mk-triangle-set* ‘ $\{(x, y, z) \in X \times Y \times Z$. (*triangle-in-graph x y z G*)$\}$
$\subseteq$ *triangle-set G*
by (*auto simp add*: *subsetI triangle-set-def*) (*auto*)

lemma *finite-triangle-set*:
fixes *G* :: *ugraph*
assumes *fin*: *finite* (*uverts G*) **and** *wf*: *uwellformed G*
shows *finite* (*triangle-set G*)
proof −
have *triangle-set G* ⊆ *Pow* (*uverts G*)
using *insert-iff local.wf triangle-in-graph-def triangle-set-def uwellformed-def* **by**
auto
then show *?thesis*
by (*meson fin finite-Pow-iff infinite-super*)
qed

lemma *card-triangle-3*:

fixes G :: *ugraph*
assumes $t \in$ *triangle-set* G *uwellformed* G
shows *card* $t = 3$
using *assms* **by** (*auto simp*: *triangle-set-def edge-vertices-not-equal triangle-in-graph-def*)

lemma *triangle-set-power-set-ss*: *uwellformed* $G \Longrightarrow$ *triangle-set* $G \subseteq$ *Pow* (*uverts* G)
by (*auto simp add*: *triangle-set-def triangle-in-graph-def uwellformed-alt-fst uwellformed-alt-snd*)

lemma *triangle-set-finite*:
assumes *finite* (*uverts* G)
assumes *uwellformed* G
shows *finite* (*triangle-set* G)
using *triangle-set-power-set-ss assms*
by (*meson finite-Pow-iff rev-finite-subset*)

lemma *triangle-in-graph-ss*:
fixes G :: *ugraph* **and** *Gnew* :: *ugraph*
assumes *uedges Gnew* $\subseteq$ *uedges G*
assumes *triangle-in-graph x y z Gnew*
shows *triangle-in-graph x y z G*
proof −
have $\{x, y\} \in$ *uedges G* **using** *assms triangle-in-graph-def* **by** *auto*
have $\{y, z\} \in$ *uedges G* **using** *assms triangle-in-graph-def* **by** *auto*
have $\{x, z\} \in$ *uedges G* **using** *assms triangle-in-graph-def* **by** *auto*
thus *?thesis*
by (*simp add*: ‹$\{x, y\} \in$ *uedges G*› ‹$\{y, z\} \in$ *uedges G*› *triangle-in-graph-def*)
qed

lemma *triangle-set-graph-edge-ss*:
fixes G :: *ugraph* **and** *Gnew* :: *ugraph*
assumes *uwellformed G*
assumes *uedges Gnew* $\subseteq$ *uedges G*
assumes *uverts Gnew* = *uverts G*
shows (*triangle-set Gnew*) $\subseteq$ (*triangle-set G*)
proof (*intro subsetI*)
fix t **assume** $t \in$ *triangle-set Gnew*
then obtain x y z **where** $t = \{x,y,z\}$ **and** *triangle-in-graph x y z Gnew*
using *triangle-set-def assms mem-Collect-eq* **by** *auto*
then have *triangle-in-graph x y z G* **using** *assms triangle-in-graph-ss* **by** *simp*
thus $t \in$ *triangle-set G* **using** *triangle-set-def assms*
using ‹$t = \{x,y,z\}$› **by** *auto*
qed

lemma *triangle-set-graph-edge-ss-bound*:
fixes G :: *ugraph* **and** *Gnew* :: *ugraph*
assumes *uwellformed G*
assumes *finite* (*uverts G*)

assumes *uedges Gnew* $\subseteq$ *uedges G*
assumes *uverts Gnew* = *uverts G*
shows *card* (*triangle-set G*) $\geq$ *card* (*triangle-set Gnew*)
using *triangle-set-graph-edge-ss triangle-set-finite*
by (*simp add*: *assms card-mono*)

1.5 The Triangle Counting Lemma and the Triangle Removal Lemma

We begin with some more auxiliary material to be used in the main lemmas.

lemma *regular-pairI*:
fixes ε :: *real* **and** *G* :: *ugraph* **and** *X* :: *uvert set* **and** *Y* ::*uvert set*
assumes $\varepsilon > 0$ **and** *regular-pair X Y G* ε **and** *xss*: $X' \subseteq X$ **and** *yss*: $Y' \subseteq Y$ **and** *card* $X' \geq \varepsilon *$ *card X* **and** (*card* $Y' \geq \varepsilon *$ *card Y*)
shows | *edge-density X' Y' G − edge-density X Y G* | $\leq \varepsilon$
using *regular-pair-def assms* **by** *meson*

lemma *edge-density-zero*: $Y = \{\} \Longrightarrow$ *edge-density X Y G* = *0*
by (*simp add*: *edge-density-def*)

lemma *regular-pair-neighbor-bound*:
fixes ε::*real*
assumes *finG*: *finite* (*uverts G*)
assumes *xss*: *X* $\subseteq$ *uverts G* **and** *yss*: *Y* $\subseteq$ *uverts G* **and** *card X* > *0*
and *wf*: *uwellformed G*
and *eg0*: $\varepsilon > 0$ **and** *regular-pair X Y G* ε **and** *ed*: *edge-density X Y G* $\geq 2*\varepsilon$
shows *card*{*x* $\in$ *X*. *card* (*neighbors-ss x Y G*) < (*edge-density X Y G* − ε)∗ *card* (*Y*)} < ε ∗ *card X*
(**is** *card* (*?X'*) < ε ∗ -)
proof (*cases ?X'* = {})
case *True*
then show *?thesis*
by (*simp add*: *True* ‹*card X* > *0*› *eg0*)
next
case *False*
show *?thesis*
proof (*rule ccontr*)
assume ¬ (*card* (*?X'*) < ε ∗ *card X*)
then have *a*: (*card*(*?X'*) $\geq$ ε ∗ *card X*) **by** *simp*
have *fin*: *finite X finite Y* **using** *assms finite-subset* **by** *auto*
have *ebound*: $\varepsilon \leq 1/2$
by (*metis ed edge-density-le1 le-divide-eq-numeral1*(*1*) *mult.commute order-trans*)
have *finx*: *finite ?X'* **using** *fin* **by** *simp*
have $\bigwedge$ *x*. *x* $\in$ *?X'*$\Longrightarrow$ (*card* (*neighbors-ss x Y G*)) < (*edge-density X Y G* − ε) ∗ (*card Y*)
by *blast*
then have ($\sum$ *x*$\in$*?X'*. *card* (*neighbors-ss x Y G*)) < ($\sum$ *x*$\in$*?X'*. ((*edge-density X Y G* − ε)∗ (*card Y*)))

using *False sum-strict-mono*
by (*smt* (*verit*, *del-insts*) *finx of-nat-sum*)
then have *upper*: $(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G)) < (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y))$
by (*simp add*: *sum-bounded-above*)
have *sumge0*: $(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G)) \geq 0$
by *blast*
have *xge0*: $card\ X > 0$
using *fin*(1) *False* **by** *fastforce*
have *yge0*: $card\ Y > 0$
using *False* **by** *fastforce*
then have *xyge0*: $card\ X * card\ Y > 0$ **using** *xge0* **by** *simp*
then have *xyne0*: $card\ X * card\ Y \neq 0$ **by** *simp*
have *fracg0*:$(1/(card\ ?X' * card\ Y)) > 0$
using *card-0-eq finx False yge0* **by** *fastforce*
then have *upper2*: $(1/(card\ ?X' * card\ Y)) * (\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G)) < (1/(card\ ?X' * card\ Y)) * (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y))$
using *upper mult-less-cancel-left-pos*[*of* $(1/(card\ ?X' * card\ Y))$ $(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G))$ $(card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y))$]
by *linarith*
have *minuse*: $(1/(card\ ?X' * card\ Y)) * (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y)) = (edge\text{-}density\ X\ Y\ G - \varepsilon)$
proof −
have $(1/(card\ ?X' * card\ Y)) * (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y)) = (1/(card\ ?X' * card\ Y)) * ((card\ ?X')*(card\ Y))*(edge\text{-}density\ X\ Y\ G - \varepsilon)$
by (*smt* (*z3*) *divide-divide-eq-right of-nat-mult times-divide-eq-left*)
also have $\ldots = ((card\ ?X'* card\ Y)/(card\ ?X' * card\ Y)) * (edge\text{-}density\ X\ Y\ G - \varepsilon)$ **by** *simp*
also have $\ldots = 1 * (edge\text{-}density\ X\ Y\ G - \varepsilon)$
using *divide-eq-1-iff*[*of* $(card\ ?X'* card\ Y)$ $(card\ ?X'* card\ Y)$] *xyne0*
using *finx False* **by** *force*
finally have $(1/(card\ ?X' * card\ Y)) * (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y)) = (edge\text{-}density\ X\ Y\ G - \varepsilon)$ **by** *simp*
thus *?thesis* **by** *simp*
qed
then have *edlt1*: $(1/(card\ ?X' * card\ Y)) * (card\ ?X')*((edge\text{-}density\ X\ Y\ G - \varepsilon)*(card\ Y)) < edge\text{-}density\ X\ Y\ G$
using *eg0*
by *linarith*
then have *edlt2*: $(1/(card\ ?X' * card\ Y))*(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G)) < edge\text{-}density\ X\ Y\ G$
using *upper2* **by** *linarith*
then have $|edge\text{-}density\ X\ Y\ G - (1/(card\ ?X' * card\ Y))*(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G))| = edge\text{-}density\ X\ Y\ G - (1/(card\ ?X' * card\ Y))*(\sum x{\in}?X'.\ card\ (neighbors\text{-}ss\ x\ Y\ G))$
by *linarith*
have $(edge\text{-}density\ X\ Y\ G - (1/(card\ ?X' * card\ Y))*(\sum x{\in}?X'.\ card$

(*neighbors-ss x Y G*))) > (*edge-density X Y G* − (*1*/(*card ?X'* ∗ *card Y*)) ∗ (*card ?X'*)∗ ((*edge-density X Y G* − *ε*)∗ (*card Y*)))
using *edlt1 edlt2 upper2*
by *linarith*
then have *edge-density X Y G* − (*1*/(*card ?X'* ∗ *card Y*)) ∗ (∑ *x*∈*?X'*. *card* (*neighbors-ss x Y G*)) > *edge-density X Y G* − (*edge-density X Y G* − *ε*)
using *minuse* **by** *linarith*
then have *con*: *edge-density X Y G* − (*1*/(*card ?X'* ∗ *card Y*)) ∗ (∑ *x*∈*?X'*. *card* (*neighbors-ss x Y G*)) > *ε* **by** *simp*
have *ye*: *card Y* ≥ *ε* ∗ (*card Y*) **using** *ebound* **by** (*simp add*: *yge0*)
have *xe'*: *card ?X'* ≥ *ε* ∗ (*card X*) **using** *a* **by** *fastforce*
have *?X'* ⊆ *X* **by** *simp*
then have | *edge-density ?X' Y G* − *edge-density X Y G* | ≤ *ε*
using *regular-pairI*[*of ε X Y G ?X' Y*] *assms ye xe'* **by** *simp*
then have | (*card* (*all-edges-between ?X' Y G*))/ (*card ?X'* ∗ *card Y*) − *edge-density X Y G* | ≤ *ε*
by (*simp add*: *edge-density-def*)
then have | (*1*/(*card ?X'* ∗ *card Y*)) ∗ (*card* (*all-edges-between ?X' Y G*)) − *edge-density X Y G* | ≤ *ε*
by *simp*
then have |(*1*/(*card ?X'* ∗ *card Y*)) ∗ (∑ *x*∈*?X'*. *card* (*neighbors-ss x Y G*)) − *edge-density X Y G* | ≤ *ε*
using *card-all-edges-betw-neighbor fin wf* **by** *simp*
then have *lt*: |*edge-density X Y G* − (*1*/(*card ?X'* ∗ *card Y*)) ∗ (∑ *x*∈*?X'*. *card* (*neighbors-ss x Y G*)) | ≤ *ε*
by *simp*
thus *False* **using** *lt con* **by** *linarith*
qed
qed

lemma *neighbor-set-meets-e-reg-cond*:
fixes *ε*::*real*
assumes *X* ⊆ *uverts G* **and** *Y* ⊆ *uverts G* **and** *enot0*: *ε* > *0*
and *fin*: *finite X finite Y* **and** *uwellformed G*
and *rp1*: *regular-pair X Y G ε*
and *ed1*: *edge-density X Y G* ≥ *2*∗*ε*
and *card* (*neighbors-ss x Y G*) ≥ (*edge-density X Y G* − *ε*) ∗ *card Y*
shows *card* (*neighbors-ss x Y G*) ≥ *ε* ∗ *card* (*Y*)
proof −
have *card* (*neighbors-ss x Y G*) ≥ (*edge-density X Y G* − *ε*) ∗ *card Y* **using** *assms* **by** *simp*
thus *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *mult-right-mono of-nat-less-0-iff ed1 enot0*)
qed

lemma *all-edges-btwn-neighbour-sets-lower-bound*:
fixes *ε*::*real*
assumes *X* ⊆ *uverts G* **and** *Y* ⊆ *uverts G* **and** *Z* ⊆ *uverts G* **and** *ε* > *0*

and *finG*: *finite* (*uverts G*)
and *wf*: *uwellformed G* **and** *fin*: *finite X finite Y finite Z*
and *rp1*: *regular-pair X Y G ε* **and** *rp2*: *regular-pair Y Z G ε* **and** *rp3*: *regular-pair X Z G ε*
and *ed1*: *edge-density X Y G* ≥ *2∗ε* **and** *ed2*: *edge-density X Z G* ≥ *2∗ε* **and** *ed3*: *edge-density Y Z G* ≥ *2∗ε*
and *cond1*: *card* (*neighbors-ss x Y G*) ≥ (*edge-density X Y G* − *ε*) ∗ *card Y*
and *cond2*: *card* (*neighbors-ss x Z G*) ≥ (*edge-density X Z G* − *ε*) ∗ *card Z*
and *x* ∈ *X*
shows *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*)
≥ (*edge-density Y Z G* − *ε*) ∗ *card* (*neighbors-ss x Y G*) ∗ *card* (*neighbors-ss x Z G*)
(**is** *card* (*all-edges-between ?Y′ ?Z′ G*) ≥ (*edge-density Y Z G* − *ε*) ∗ *card ?Y′* ∗ *card ?Z′*)
proof −
have *yss′*: *?Y′* ⊆ *Y* **using** *neighbors-ss-def* **by** *simp*
have *zss′*: *?Z′* ⊆ *Z* **using** *neighbors-ss-def* **by** *simp*
have *min-sizeY*: *card ?Y′* ≥ *ε* ∗ *card Y* **using** *neighbor-set-meets-e-reg-cond cond1 assms fin* **by** *meson*
have *min-sizeZ*: *card ?Z′* ≥ *ε* ∗ *card Z* **using** *neighbor-set-meets-e-reg-cond cond2 assms fin* **by** *meson*
then have | *edge-density ?Y′ ?Z′ G* − *edge-density Y Z G* | ≤ *ε*
using *min-sizeY regular-pairI*[*of ε Y Z G ?Y′ ?Z′*] *yss′ zss′ assms* **by** *simp*
then have −*ε* ≤ (*edge-density ?Y′ ?Z′ G* − *edge-density Y Z G*)
by *linarith*
then have *edge-density Y Z G* − *ε* ≤ *edge-density ?Y′ ?Z′ G* **by** *linarith*
then have *edge-density Y Z G* − *ε* ≤ (*card* (*all-edges-between ?Y′ ?Z′ G*)/(*card ?Y′* ∗ *card ?Z′*)) **using** *edge-density-def* **by** *simp*
then have (*card ?Y′* ∗ *card ?Z′*) ∗ (*edge-density Y Z G* − *ε*) ≤ (*card* (*all-edges-between ?Y′ ?Z′ G*))
by (*metis abs-of-nat division-ring-divide-zero le-divide-eq mult-of-nat-commute of-nat-0-le-iff times-divide-eq-right zero-less-abs-iff*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *ab-semigroup-mult-class.mult-ac*(*1*) *mult-of-nat-commute of-nat-mult*)
qed

lemma *edge-density-implies-edge-exists*:
fixes *ε*::*real*
assumes *X* ⊆ *uverts G* **and** *Y* ⊆ *uverts G* **and** *ε* > *0* **and** *uwellformed G*
assumes *edge-density X Y G* ≥ *ε*
obtains *e* **where** *e* ∈ *all-edges-between X Y G*
proof −
have *edge-density X Y G* = *card* (*all-edges-between X Y G*) / (*card X* ∗ *card Y*) **by** (*simp add*: *edge-density-def*)
then have *card* (*all-edges-between X Y G*) ≠ *0*
using *assms divide-eq-0-iff* **by** *fastforce*
thus *?thesis*

by (*metis card.empty empty-subsetI subsetI subset-antisym that*)
qed

We are now ready to show the Triangle Counting Lemma (Theorem 3.13 in Zhao's notes):

theorem *triangle-counting-lemma*:
fixes ε::*real*
assumes *xss*: $X \subseteq$ *uverts G* **and** *yss*: $Y \subseteq$ *uverts G* **and** *zss*: $Z \subseteq$ *uverts G* **and** *en0*: $\varepsilon > 0$
and *finG*: *finite* (*uverts G*) **and** *wf*: *uwellformed G*
and *rp1*: *regular-pair X Y G* ε **and** *rp2*: *regular-pair Y Z G* ε **and** *rp3*: *regular-pair X Z G* ε
and *ed1*: *edge-density X Y G* $\geq 2*\varepsilon$ **and** *ed2*: *edge-density X Z G* $\geq 2*\varepsilon$ **and** *ed3*: *edge-density Y Z G* $\geq 2*\varepsilon$
shows *card* (*triangle-triples X Y Z G*)
$\geq (1-2*\varepsilon)*((\textit{edge-density } X\ Y\ G) - \varepsilon)*((\textit{edge-density } X\ Z\ G) - \varepsilon)*((\textit{edge-density } Y\ Z\ G) - \varepsilon)*$
(*card X*)$*$(*card Y*)$*$ (*card Z*)
proof −
let *?T-all* = $\{(x,y,z) \in X \times Y \times Z.$ (*triangle-in-graph x y z G*)$\}$
define *XF* **where** $XF \equiv \lambda Y.\ \{x \in X.\ \textit{card}(\textit{neighbors-ss } x\ Y\ G) < ((\textit{edge-density } X\ Y\ G) - \varepsilon) * \textit{card } Y\}$
have *fin*: *finite X finite Y finite Z* **using** *finG rev-finite-subset xss yss zss* **by** *auto*
have *card X* > 0
using *card-0-eq ed1 edge-density-def en0 fin*(*1*) **by** *fastforce*
have *ebound*: $\varepsilon \leq 1/2$
using *ed1 edge-density-le1 fin*
by (*metis le-divide-eq-numeral1*(*1*) *mult.commute order-trans*)
then have *ebound2*: $1 - 2*\varepsilon \geq 0$
by *linarith*

Obtain a subset of X where all elements meet minimum numbers for neighborhood size in Y and Z.

define *X2* **where** $X2 \equiv X - (XF\ Y \cup XF\ Z)$
have *xss*: $X2 \subseteq X$
by (*simp add*: *X2-def*)
have *finx2*: *finite X2*
by (*simp add*: *X2-def fin*)

Reasoning on the minimum size of *X2*:

have *part1*: $(XF\ Y \cup XF\ Z) \cup X2 = X$
by (*auto simp*: *XF-def X2-def*)
have *card-XFY*: *card* $(XF\ Y) <$ $\varepsilon *$ *card X*
using *regular-pair-neighbor-bound assms* ‹*card X* > 0› **by** (*simp add*: *XF-def*)

We now repeat the same argument as above to the regular pair X Z in G.

have *card-XFZ*: *card* $(XF\ Z) <$ $\varepsilon *$ *card X*

using *regular-pair-neighbor-bound assms* ‹*card X > 0*› **by** (*simp add*: *XF-def*)
have *card* (*XF Y* ∪ *XF Z*) ≤ *2* ∗ *ε* ∗ (*card X*)
by (*smt* (*verit*) *card-XFY card-XFZ card-Un-le comm-semiring-class.distrib of-nat-add of-nat-mono*)
then have *minx2help*: *card X2* ≥ *card X* − *2* ∗ *ε* ∗ *card X* **using** *part1*
by (*smt* (*verit*, *del-insts*) *card-Un-le of-nat-add of-nat-mono*)
then have *minx2*: *card X2* ≥ (*1* − *2* ∗ *ε*) ∗ *card X*
by (*metis mult.commute mult-cancel-left2 right-diff-distrib*)
have *edmultbound*: ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*) ≥ *0*
using *ed3 ed1 ed2 assms*(*4*) **by** *auto*

Reasoning on the minimum number of edges between neighborhoods of X in Y and Z.

have *edyzgt0*: ((*edge-density Y Z G*) − *ε*) > *0*
and *edxygt0*: ((*edge-density X Y G*) − *ε*) > *0* **using** *ed1 ed3* ‹*ε > 0*› **by** *linarith*+
have *cardnzgt0*: *card* (*neighbors-ss x Z G*) ≥ *0* **and** *cardnygt0*: *card* (*neighbors-ss x Y G*) ≥ *0*
if *x* ∈ *X2* **for** *x*
by *auto*
have *card-y-bound*: ⋀*x*. *x* ∈ *X2* ⟹ (*card* (*neighbors-ss x Y G*)) ≥ (*edge-density X Y G* − *ε*) ∗ (*card Y*)
by (*auto simp*: *XF-def X2-def*)
have *card-z-bound*: ⋀*x*. *x* ∈ *X2* ⟹ (*card* (*neighbors-ss x Z G*)) ≥ (*edge-density X Z G* − *ε*) ∗ (*card Z*)
by (*auto simp*: *XF-def X2-def*)
have *card-y-bound′*:
(∑ *x*∈ *X2*. ((*edge-density Y Z G*) − *ε*) ∗ (*card* (*neighbors-ss x Y G*)) ∗ (*card* (*neighbors-ss x Z G*))) ≥
(∑ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*card* (*neighbors-ss x Z G*)))
by (*rule sum-mono*) (*smt* (*verit*, *best*) *Groups.mult-ac*(*3*) *card-y-bound edyzgt0 mult.commute mult-right-mono of-nat-0-le-iff*)
have *x2-card*: ⋀*x*. *x* ∈ *X2* ⟹ ((*edge-density Y Z G*) − *ε*) ∗ (*card* (*neighbors-ss x Y G*)) ∗ (*card* (*neighbors-ss x Z G*))
≤ *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*)
by (*meson all-edges-btwn-neighbour-sets-lower-bound assms*(*1*) *card-y-bound card-z-bound ed1 ed2 ed3 en0 fin finG local.wf rp1 rp2 rp3 subsetD xss yss zss*)
have *card-z-bound′*:
(∑ *x*∈ *X2*. ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*card* (*neighbors-ss x Z G*))) ≥
(∑ *x*∈ *X2*. ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*))
using *card-z-bound mult-left-mono edxygt0 edyzgt0* **by** (*fastforce intro*!: *sum-mono*)
have *eq-set*: ⋀ *x*. *x* ∈ *X* ⟹ *card* {(*y*, *z*) . *y* ∈ *Y* ∧ *z* ∈ *Z* ∧ {*y*, *z*} ∈ *uedges G* ∧ *neighbor-in-graph x y G* ∧ *neighbor-in-graph x z G* } =
card {(*y*, *z*) . *y* ∈ (*neighbors-ss x Y G*) ∧ *z* ∈ (*neighbors-ss x Z G*) ∧ {*y*, *z*} ∈

uedges G }
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq neighbors-ss-def*)
have *card ?T-all* = ($\sum$ *x*∈ *X* . *card* {(*y*,*z*) ∈ *Y* × *Z*. *triangle-in-graph x y z G*})
using *triple-sigma-rewrite-card fin* **by** *force*
also have ... = ($\sum$ *x*∈ *X* . *card* {(*y*,*z*) ∈ *Y* × *Z*. {*y*,*z*} ∈ *uedges G* ∧ *neighbor-in-graph x y G* ∧ *neighbor-in-graph x z G* })
using *triangle-in-graph-edge-point assms* **by** *auto*
also have ... = ($\sum$ *x*∈ *X* . *card* {(*y*, *z*). *y* ∈ *Y* ∧ *z* ∈ *Z* ∧ {*y*, *z*} ∈ *uedges G* ∧ *neighbor-in-graph x y G* ∧ *neighbor-in-graph x z G* })
by *simp*
finally have *card ?T-all* = ($\sum$ *x*∈ *X* . *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*))
using *eq-set* **by** (*auto simp*: *all-edges-between-def*)
then have *l*: *card ?T-all* ≥ ($\sum$ *x*∈ *X2* . *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*))
by (*simp add*: *fin xss sum-mono2*)
have ($\sum$ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*card* (*neighbors-ss x Y G*)) ∗ (*card* (*neighbors-ss x Z G*))) ≤
($\sum$ *x*∈ *X2*. *real* (*card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*)))
by (*meson x2-card finx2 sum-mono*)
then have *card ?T-all* ≥ ($\sum$ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*card* (*neighbors-ss x Y G*)) ∗ (*card* (*neighbors-ss x Z G*)))
using *l of-nat-le-iff* [*symmetric*, **where** *'a=real*] **by** *force*
then have *card ?T-all* ≥ ($\sum$ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*card* (*neighbors-ss x Z G*)))
using *card-y-bound'* **by** *simp*
then have *tall-gt*: *card ?T-all* ≥ ($\sum$ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*))
using *card-z-bound'* **by** *simp*
have ($\sum$ *x*∈ *X2* . ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*)) =
card X2 ∗ ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*)
by *simp*
then have *of-real* (*card ?T-all*) ≥ *card X2* ∗ ((*edge-density Y Z G*) − *ε*) ∗ (*edge-density X Y G* − *ε*)∗
(*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*)
using *tall-gt*
by *force*
then have *of-real* (*card ?T-all*) ≥ ((*1* − *2* ∗ *ε*) ∗ *card X*) ∗ ((*edge-density Y Z G*) − *ε*) ∗
(*edge-density X Y G* − *ε*)∗ (*card Y*) ∗ (*edge-density X Z G* − *ε*)∗ (*card Z*)
using *minx2 edmultbound dual-order.trans mult.commute ordered-comm-semiring-class.comm-mult-left-mon*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*4*) *ed1 ed2 mult-cancel-right mult-less-cancel-right*

mult-pos-neg2 of-nat-0-eq-iff of-nat-le-0-iff)
then show *?thesis* **by** (*simp add*: *triangle-triples-def mult.commute mult.left-commute*)

qed

definition *regular-graph* :: *uvert set set* ⇒ *ugraph* ⇒ *real* ⇒ *bool*
where *regular-graph P G ε* ≡ ∀ *R S*. *R*∈*P* ⟶ *S*∈*P* ⟶ *regular-pair R S G ε*
for *ε*::*real*

A minimum density, but empty edge sets are excluded.

definition *edge-dense* :: *nat set* ⇒ *nat set* ⇒ *ugraph* ⇒ *real* ⇒ *bool*
where *edge-dense X Y G ε* ≡ *all-edges-between X Y G* = {} ∨ *edge-density X Y G* ≥ *ε*

definition *dense-graph* :: *uvert set set* ⇒ *ugraph* ⇒ *real* ⇒ *bool*
where *dense-graph P G ε* ≡ ∀ *R S*. *R*∈*P* ⟶ *S*∈*P* ⟶ *edge-dense R S G ε* **for** *ε*::*real*

definition *decent* :: *nat set* ⇒ *nat set* ⇒ *ugraph* ⇒ *real* ⇒ *bool*
where *decent X Y G η* ≡ *all-edges-between X Y G* = {} ∨ (*real* (*card X*) ≥ *η* ∧ *real* (*card Y*) ≥ *η*)

definition *decent-graph* :: *uvert set set* ⇒ *ugraph* ⇒ *real* ⇒ *bool*
where *decent-graph P G η* ≡ ∀ *R S*. *R*∈*P* ⟶ *S*∈*P* ⟶ *decent R S G η* **for** *ε*::*real*

The proof of the triangle counting lemma requires ordered triples. For each unordered triple there are six permutations, hence the factor of 1/6 here. This is mentioned briefly on pg 57 of Zhao's notes towards the end of the proof.

lemma *card-convert-triangle-rep*:
fixes *G* :: *ugraph*
assumes *X* ⊆ *uverts G* **and** *Y* ⊆ *uverts G* **and** *Z* ⊆ *uverts G* **and** *fin*: *finite* (*uverts G*)
and *wf*: *uwellformed G*
shows *card* (*triangle-set G*) ≥ *1/6* ∗ *card* {(*x*, *y*, *z*) ∈ *X* × *Y* × *Z* . (*triangle-in-graph x y z G*)}
(**is** - ≥ *1/6* ∗ *card ?TT*)
proof −
define *tofl* **where** *tofl* ≡ λ*l*::*nat list*. (*hd l*, *hd*(*tl l*), *hd*(*tl*(*tl l*)))
have *in-tofl*: (*x*, *y*, *z*) ∈ *tofl* ‘ *permutations-of-set* {*x*,*y*,*z*} **if** *x*≠*y y*≠*z x*≠*z* **for** *x y z*
proof −
have *distinct*[*x*,*y*,*z*]
using *that* **by** *simp*
then show *?thesis*
unfolding *tofl-def image-iff*
by (*smt* (*verit*, *best*) *list.sel*(*1*) *list.sel*(*3*) *list.simps*(*15*) *permutations-of-setI set-empty*)
qed
have *?TT* ⊆ {(*x*, *y*, *z*). (*triangle-in-graph x y z G*)}
by *auto*

also have ... $\subseteq$ ($\bigcup t \in$ *triangle-set G*. *tofl* ‘ *permutations-of-set t*)
proof (*clarsimp simp*: *triangle-set-def*)
fix *u v w*
assume *t*: *triangle-in-graph u v w G*
then have (*u*, *v*, *w*) $\in$ *tofl* ‘ *permutations-of-set* {*u*,*v*,*w*}
by (*metis in-tofl local.wf triangle-commu1 triangle-vertices-distinct1 triangle-vertices-distinct2*)
with *t* **show** $\exists t.$ ($\exists x\ y\ z.\ t = \{x, y, z\} \wedge$ *triangle-in-graph x y z G*) $\wedge$ (*u*, *v*, *w*) $\in$ *tofl* ‘ *permutations-of-set t*
by *blast*
qed
finally have *?TT* $\subseteq$ ($\bigcup t \in$ *triangle-set G*. *tofl* ‘ *permutations-of-set t*) **.**
then have *card ?TT* $\leq$ *card*($\bigcup t \in$ *triangle-set G*. *tofl* ‘ *permutations-of-set t*)
by (*intro card-mono finite-UN-I finite-triangle-set*) (*auto simp*: *assms*)
also have ... $\leq$ ($\sum t \in$ *triangle-set G*. *card* (*tofl* ‘ *permutations-of-set t*))
using *card-UN-le fin finite-triangle-set local.wf* **by** *blast*
also have ... $\leq$ ($\sum t \in$ *triangle-set G*. *card* (*permutations-of-set t*))
by (*meson card-image-le finite-permutations-of-set sum-mono*)
also have ... $\leq$ ($\sum t \in$ *triangle-set G*. *fact 3*)
apply (*rule sum-mono*)
by (*metis card.infinite card-permutations-of-set card-triangle-3 eq-refl local.wf nat.simps*(*3*) *numeral-3-eq-3*)
also have ... = *6* $*$ *card* (*triangle-set G*)
by (*simp add*: *eval-nat-numeral*)
finally have *card ?TT* $\leq$ *6* $*$ *card* (*triangle-set G*) **.**
then show *?thesis*
by (*simp add*: *divide-simps*)
qed

lemma *card-convert-triangle-rep-bound*:
fixes *G* :: *ugraph* **and** *t* :: *real*
assumes *card* {(*x*, *y*, *z*) $\in$ *X* $\times$ *Y* $\times$ *Z* . (*triangle-in-graph x y z G*)} $\geq$ *t*
assumes *X* $\subseteq$ *uverts G* **and** *Y* $\subseteq$ *uverts G* **and** *Z* $\subseteq$ *uverts G* **and** *fin*: *finite* (*uverts G*)
and *wf*: *uwellformed G*
shows *card* (*triangle-set G*) $\geq$ *1/6* $*t$
proof −
define *t′* **where** *t′* $\equiv$ *card* {(*x*, *y*, *z*) $\in$ *X* $\times$ *Y* $\times$ *Z* . (*triangle-in-graph x y z G*)}
have *t′* $\geq$ *t* **using** *assms t′-def* **by** *simp*
then have *tgt*: *1/6* $*$ *t′* $\geq$ *1/6* $*$ *t* **by** *simp*
have *card* (*triangle-set G*) $\geq$ *1/6* $*t′$ **using** *t′-def card-convert-triangle-rep assms* **by** *simp*
thus *?thesis* **using** *tgt* **by** *linarith*
qed

lemma *edge-density-eq0*:
assumes *all-edges-between A B G* = {} **and** *X* $\subseteq$ *A Y* $\subseteq$ *B*
shows *edge-density X Y G* = *0*

proof −
have *all-edges-between X Y G = {}*
by (*metis all-edges-between-mono1 all-edges-between-mono2 assms subset-empty*)
then show *?thesis*
by (*auto simp*: *edge-density-def*)
qed

The following is the Triangle Removal Lemma (Theorem 3.15 in Zhao's notes).

theorem *triangle-removal-lemma*:
fixes ε :: *real*
assumes *egt*: $\varepsilon > 0$
shows $\exists \delta$::*real* > 0. $\forall$ *G*. *card*(*uverts G*) $> 0 \longrightarrow$ *uwellformed G* $\longrightarrow$
card (*triangle-set G*) $\leq \delta *$ *card*(*uverts G*) ^ *3* $\longrightarrow$
($\exists$ *Gnew*. *triangle-free-graph Gnew* $\wedge$ *uverts Gnew* = *uverts G* $\wedge$ (*uedges Gnew* $\subseteq$ *uedges G*) $\wedge$
card (*uedges G* − *uedges Gnew*) $\leq \varepsilon *$ (*card* (*uverts G*))^2)
(**is** $\exists \delta$::*real* > 0. $\forall$ *G*. - $\longrightarrow$ - $\longrightarrow$ - $\longrightarrow$ ($\exists$ *Gnew*. *?Φ G Gnew*))
proof (*cases* $\varepsilon < 1$)
case *False*
define *Gnew* **where** *Gnew* $\equiv \lambda G$. ((*uverts G*), {}::*uedge set*)
show *?thesis*
proof (*intro exI conjI strip*)
fix *G*
assume *G*: *uwellformed G card*(*uverts G*) > 0
then show *triangle-free-graph* (*Gnew G*) *uverts* (*Gnew G*) = *uverts G uedges* (*Gnew G*) $\subseteq$ *uedges G*
by (*auto simp*: *Gnew-def triangle-free-graph-empty*)
have *real* (*card* (*uedges G*)) $\leq$ (*card* (*uverts G*))^2
by (*meson G card-gt-0-iff max-edges-graph of-nat-le-iff*)
also have ... $\leq \varepsilon *$ (*card* (*uverts G*))^2
using *False mult-le-cancel-right1* **by** *fastforce*
finally show *real* (*card* (*uedges G* − *uedges* (*Gnew G*))) $\leq \varepsilon *$ *real* ((*card* (*uverts G*))^2)
by (*simp add*: *Gnew-def*)
qed (*rule zero-less-one*)
next
case *True*
have *e4gt*: $\varepsilon/4 > 0$ **using** ‹$\varepsilon > 0$› **by** *auto*
then obtain *M0* **where**
M0: $\bigwedge$*G*. *card* (*uverts G*) $> 0 \Longrightarrow \exists$ *P*. *regular-partition* ($\varepsilon/4$) *G P* $\wedge$ *card P* $\leq$ *M0*
and *M0*>*0*
by (*metis Szemeredi-Regularity-Lemma le0 neq0-conv not-le not-numeral-le-zero*)
define *D0* **where** *D0* $\equiv$ *1/6* $*(1-(\varepsilon/2))*((\varepsilon/4)\hat{}3)*((\varepsilon/(4*M0))\hat{}3)$
have *D0* > 0
using ‹*0* $< \varepsilon$› ‹$\varepsilon <$ *1*› ‹*M0* > 0› **by** (*simp add*: *D0-def zero-less-mult-iff*)
then obtain δ:: *real* **where** δ: *0* $< \delta$ $\delta <$ *D0*
by (*meson dense*)

show *?thesis*
proof (*rule exI*, *intro conjI strip*)
fix *G*
assume *card*(*uverts G*) > *0* **and** *wf*: *uwellformed G*
then have *fin*: *finite* (*uverts G*)
by (*simp add*: *card-gt-0-iff*)

Assume that, for a yet to be determined δ, we have:

assume *ineq*: *real* (*card* (*triangle-set G*)) ≤ *δ* ∗ *card* (*uverts G*) ^ *3*

Step 1: Partition: Using Szemerédi's Regularity Lemma, we get an $\epsilon/4$ partition.

let *?n* = *card* (*uverts G*)
have *vne*: *uverts G* ≠ {}
using ‹*0* < *card* (*uverts G*)› **by** *force*
then have *ngt0*: *?n* > *0*
by (*simp add*: *fin card-gt-0-iff*)
with *M0* **obtain** *P* **where** *M*: *regular-partition* (*ε*/*4*) *G P* **and** *card P* ≤ *M0*
by *blast*
define *M* **where** *M* ≡ *card P*
have *finite P*
by (*meson M fin finite-elements regular-partition-def*)
with *M0* **have** *M* > *0*
unfolding *M-def*
by (*metis M card-gt-0-iff partition-onD1 partition-on-empty regular-partition-def vne*)
let *?e4M* = *ε* / (*4* ∗ *real M*)
define *D* **where** *D* ≡ *1/6* ∗(*1*−(*ε*/*2*)) ∗ ((*ε*/*4*) ^ *3*) ∗ *?e4M* ^ *3*
have *D* > *0*
using ‹*0* < *ε*› ‹*ε* < *1*› ‹*M* > *0*› **by** (*simp add*: *D-def zero-less-mult-iff*)
have *D0* ≤ *D*
unfolding *D0-def D-def* **using** ‹*0* < *ε*› ‹*ε* < *1*› ‹*card P* ≤ *M0*› ‹*M* > *0*›
by (*intro mult-mono*) (*auto simp*: *frac-le M-def*)

have *fin-part*: *finite-graph-partition* (*uverts G*) *P M*
using *M* **unfolding** *regular-partition-def finite-graph-partition-def*
by (*metis M-def* ‹*0* < *M*› *card-gt-0-iff*)
then have *fin-P*: *finite R* **and** *card-P-gt0*: *card R* > *0* **if** *R*∈*P* **for** *R*
using *fin finite-graph-partition-finite finite-graph-partition-gt0 that* **by** *auto*
have *card-P-le*: *card R* ≤ *?n* **if** *R*∈*P* **for** *R*
using *fin fin-part finite-graph-partition-le that* **by** *meson*
have *P-disjnt*: ⋀*R S*. ⟦*R* ≠ *S*; *R* ∈ *P*; *S* ∈ *P*⟧ ⟹ *R* ∩ *S* = {}
using *fin-part*
by (*metis disjnt-def finite-graph-partition-def insert-absorb pairwise-insert partition-on-def*)
have *sum-card-P*: (∑ *R*∈*P*. *card R*) = *?n*
using *card-finite-graph-partition fin fin-part* **by** *meson*

Step 2. Cleaning. For each ordered pair of parts (P_i, P_j), remove all edges between P_i and P_j if (a) it is an irregular pair, (b) its edge density

$< \epsilon/2$, (c) either P_i or P_j is small ($\leq (\epsilon/4M)n$) Process (a) removes at most $(\epsilon/4)n^2$ edges. Process (b) removes at most $(\epsilon/2)n^2$ edges. Process (c) removes at most $(\epsilon/4)n^2$ edges. The remaining graph is triangle-free for some choice of δ. We call the graph obtained after this process *Gnew*.

define *edge* **where** *edge* ≡ λ*R S*. *mk-uedge* ‘ (*all-edges-between R S G*)
have *edge-commute*: *edge R S* = *edge S R* **for** *R S*
by (*force simp add*: *edge-def all-edges-between-swap* [*of S*] *split*: *prod.split*)
have *card-edge-le-card*: *card* (*edge R S*) ≤ *card* (*all-edges-between R S G*) **for** *R S*
by (*simp add*: *card-image-le edge-def fin finite-all-edges-between′ local.wf*)
have *card-edge-le*: *card* (*edge R S*) ≤ *card R* ∗ *card S* **if** *R*∈*P S*∈*P* **for** *R S*
by (*meson card-edge-le-card fin-P le-trans max-all-edges-between that*)

Obtain the set of edges meeting condition (a).

define *irreg-pairs* **where** *irreg-pairs* ≡ {(*R*,*S*). *R* ∈ *P* ∧ *S* ∈ *P* ∧ *irregular-pair R S G* (*ε*/*4*)}
define *Ea* **where** *Ea* ≡ (⋃(*R*,*S*) ∈ *irreg-pairs*. *edge R S*)

Obtain the set of edges meeting condition (b).

define *low-density-pairs*
where *low-density-pairs* ≡ {(*R*,*S*). *R* ∈ *P* ∧ *S* ∈ *P* ∧ ¬ *edge-dense R S G* (*ε*/*2*)}
define *Eb* **where** *Eb* ≡ (⋃(*i*,*j*) ∈ *low-density-pairs*. *edge i j*)

Obtain the set of edges meeting condition (c).

define *small* **where** *small* ≡ λ*R*. *R* ∈ *P* ∧ *card R* ≤ *?e4M* ∗ *?n*
let *?SMALL* = *Collect small*
define *small-pairs* **where** *small-pairs* ≡ {(*R*,*S*). *R* ∈ *P* ∧ *S* ∈ *P* ∧ (*small R* ∨ *small S*)}
define *Ec* **where** *Ec* ≡ (⋃*R* ∈ *?SMALL*. ⋃*S* ∈ *P*. *edge R S*)
have *Ec-def′*: *Ec* = (⋃(*i*,*j*) ∈ *small-pairs*. *edge i j*)
by (*force simp*: *edge-commute small-pairs-def small-def Ec-def*)
have *eabound*: *card Ea* ≤ (*ε*/*4*) ∗ *?n*^*2* — Count the edge bound for *Ea*
proof −
have §: ⋀*R S*. ⟦*R* ∈ *P*; *S* ∈ *P*⟧ ⟹ *card* (*edge R S*) ≤ *card R* ∗ *card S*
unfolding *edge-def*
by (*meson card-image-le fin-P finite-all-edges-between max-all-edges-between order-trans*)
have *irreg-pairs* ⊆ *P* × *P*
by (*auto simp*: *irreg-pairs-def*)
then have *finite irreg-pairs*
by (*meson* ‹*finite P*› *finite-SigmaI finite-subset*)
have *card Ea* ≤ (∑(*R*,*S*)∈*irreg-pairs*. *card* (*edge R S*))
by (*simp add*: *Ea-def card-UN-le* [*OF* ‹*finite irreg-pairs*›] *case-prod-unfold*)
also have ... ≤ (∑(*R*,*S*) ∈ {(*R*,*S*). *R*∈*P* ∧ *S*∈*P* ∧ *irregular-pair R S G* (*ε*/*4*)}. *card R* ∗ *card S*)
unfolding *irreg-pairs-def* **using** § **by** (*force intro*: *sum-mono*)
also have ... = (∑(*R*,*S*) ∈ *irregular-set* (*ε*/*4*) *G P*. *card R* ∗ *card S*)

```
      by (simp add: irregular-set-def)
    finally have card Ea ≤ (∑(R,S) ∈ irregular-set (ε/4) G P. card R * card
S) .
    with M show ?thesis
      unfolding regular-partition-def by linarith
  qed
  have ebbound: card Eb ≤ (ε/2) * (?n^2)  — Count the edge bound for Eb.
  proof −
    have subs: low-density-pairs ⊆ P × P
      by (auto simp: low-density-pairs-def)
    then have finite low-density-pairs
      by (metis ‹finite P› finite-SigmaI finite-subset)
    have real (card Eb) ≤ (∑(i,j)∈low-density-pairs. real (card (edge i j)))
      unfolding Eb-def
      by (smt (verit, ccfv-SIG) ‹finite low-density-pairs› card-UN-le of-nat-mono
of-nat-sum
 case-prod-unfold sum-mono)
    also have ... ≤ (∑(R,S)∈low-density-pairs. ε/2 * card R * card S)
      apply (rule sum-mono)
    apply(auto simp add: divide-simps card-P-gt0 low-density-pairs-def edge-density-def

 edge-dense-def)
      by (smt (verit, best) card-edge-le-card of-nat-le-iff mult.assoc)
    also have ... ≤ (∑(R,S)∈P × P. ε/2 * card R * card S)
      using subs ‹ε > 0› by (intro sum-mono2) (auto simp: ‹finite P›)
    also have ... = ε/2 * (∑(R,S)∈P × P. card R * card S)
      by (simp add: sum-distrib-left case-prod-unfold mult-ac)
    also have ... ≤ (ε/2) * (?n^2)
      using ‹ε>0› sum-prod-le-prod-sum
        by (simp add: power2-eq-square sum-product flip: sum.cartesian-product
sum-card-P)
    finally show ?thesis .
  qed
  have ecbound: card Ec ≤ (ε/4) * (?n^2)  — Count the edge bound for Ec.
  proof −
    have edge-bound: (card (edge R S)) ≤ ?e4M * ?n^2
      if S ∈ P small R for R S
    proof −
      have real (card R) ≤ ε * ?n / (4 * real M)
        using that by (simp add: small-def)
      with card-P-le [OF ‹S∈P›]
      have ∗: real (card R) * real (card S) ≤ ε * card (uverts G) / (4 * real M)
* ?n
        by (meson mult-mono of-nat-0-le-iff of-nat-mono order.trans)
      also have ... = ?e4M * ?n^2
        by (simp add: power2-eq-square)
      finally show ?thesis
        by (smt (verit) card-edge-le of-nat-mono of-nat-mult small-def that)
    qed
```

have *subs*: *?SMALL* $\subseteq$ *P*
by (*auto simp*: *small-def*)
then obtain *card-sp*: *card* (*?SMALL*) $\leq$ *M* **and** *finite ?SMALL*
using *M-def* ‹*finite P*› *card-mono* **by** (*metis finite-subset*)
have *real* (*card Ec*) $\leq$ ($\sum$ *R* $\in$ *?SMALL*. *real* (*card* ($\bigcup$ *S* $\in$ *P*. *edge R S*)))
unfolding *Ec-def*
by (*smt* (*verit*, *ccfv-SIG*) ‹*finite ?SMALL*› *card-UN-le of-nat-mono of-nat-sum case-prod-unfold sum-mono*)
also have ... $\leq$ ($\sum$ *R* $\in$ *?SMALL*. *?e4M* $*$ *?n^2*)
proof (*intro sum-mono*)
fix *R* **assume** *i*: *R* $\in$ *Collect small*
then have *R*$\in$*P* **and** *card-Pi*: *card R* $\leq$ *?e4M* $*$ *?n*
by (*auto simp*: *small-def*)
let *?UE* = $\bigcup$(*edge R* ‘ (*P*))
have $*$: *real* (*card ?UE*) $\leq$ *real* (*card R* $*$ *?n*)
proof −
have *?UE* $\subseteq$ *mk-uedge* ‘ (*all-edges-between R* (*uverts G*) *G*)
apply (*simp add*: *edge-def UN-subset-iff Ball-def*)
by (*meson all-edges-between-mono2 fin-part finite-graph-partition-subset image-mono*)
then have *card ?UE* $\leq$ *card* (*all-edges-between R* (*uverts G*) *G*)
by (*meson card-image-le card-mono fin finite-all-edges-between′ finite-imageI wf le-trans*)
then show *?thesis*
by (*meson of-nat-mono fin fin-P max-all-edges-between order.trans* ‹*R*$\in$*P*›)
qed
also have ... $\leq$ *?e4M* $*$ *real* (*?n*^{2})
using *card-Pi* ‹*M* $>$ *0*› ‹*?n* $>$ *0*› **by** (*force simp add*: *divide-simps power2-eq-square*)
finally show *real* (*card ?UE*) $\leq$ *?e4M* $*$ *real* (*?n*^{2}) **.**
qed
also have ... $\leq$ *card ?SMALL* $*$ (*?e4M* $*$ *?n^2*)
by *simp*
also have ... $\leq$ *M* $*$ (*?e4M* $*$ *?n^2*)
using *egt* **by** (*intro mult-right-mono*) (*auto simp add*: *card-sp*)
also have ... $\leq$ (ε/4) $*$ (*?n^2*)
using ‹*M* $>$ *0*› **by** *simp*
finally show *?thesis* **.**
qed
— total count
have *prev1*: *card* (*Ea* $\cup$ *Eb* $\cup$ *Ec*) $\leq$ *card* (*Ea* $\cup$ *Eb*) + *card Ec* **by** (*simp add*: *card-Un-le*)
also have ... $\leq$ *card Ea* + *card Eb* + *card Ec* **by** (*simp add*: *card-Un-le*)
also have *prev*: ... $\leq$ (ε/4)$*$(*?n^2*) + (ε/2)$*$(*?n^2*) + (ε/4)$*$(*?n^2*)
using *eabound ebbound ecbound* **by** *linarith*
finally have *cutedgesbound*: *card* (*Ea* $\cup$ *Eb* $\cup$ *Ec*) $\leq$ ε $*$ (*?n^2*) **by** *simp*

define *Gnew* **where** *Gnew* $\equiv$ (*uverts G*, *uedges G* − (*Ea* $\cup$ *Eb* $\cup$ *Ec*))
show $\exists$ *Gnew*. *?Φ G Gnew*

proof (*intro exI conjI*)
show *verts*: *uverts Gnew* = *uverts G* **by** (*simp add*: *Gnew-def*)
have *allij*: ⋀*R S*. *edge R S* ⊆ *uedges G*
using *all-edges-between-ss-uedges edge-def* **by** *presburger*
then have *eae*: *Ea* ⊆ *uedges G* **by** (*auto simp*: *Ea-def*)
have *eab*: *Eb* ⊆ *uedges G* **using** *allij* **by** (*auto simp*: *Eb-def*)
have *Ec* ⊆ *uedges G* **using** *allij* **by** (*auto simp*: *Ec-def*)
then have *diffedges*: (*Ea* ∪ *Eb* ∪ *Ec*) ⊆ *uedges G*
using *eae eab* **by** *auto*
then show *edges*: *uedges Gnew* ⊆ *uedges G*
by (*simp add*: *Gnew-def*)
then have *uedges G* − (*uedges Gnew*) = *uedges G* ∩ (*Ea* ∪ *Eb* ∪ *Ec*)
by (*simp add*: *Gnew-def Diff-Diff-Int*)
then have *uedges G* − (*uedges Gnew*) = (*Ea* ∪ *Eb* ∪ *Ec*) **using** *diffedges*
by (*simp add*: *Int-absorb1*)
then have *cardbound*: *card* (*uedges G* − *uedges Gnew*) ≤ *ε* ∗ (*?n*^2)
using *cutedgesbound* **by** *simp*
have *graph-partition-new*: *finite-graph-partition* (*uverts Gnew*) *P M* **using** *verts*
by (*simp add*: *fin-part*)
have *new-wf*: *uwellformed Gnew* **using** *subgraph-edge-wf verts edges wf* **by** *simp*
have *new-fin*: *finite* (*uverts Gnew*) **using** *verts fin* **by** *simp*

The notes by Bell and Grodzicki are quite useful for understanding the lines below. See pg 4 in the middle after the summary of the min edge counts.

have *irreg-pairs-swap*: (*R*,*S*) ∈ *irreg-pairs* ⟷ (*S*, *R*) ∈ *irreg-pairs* **for** *R S*
by (*auto simp*: *irreg-pairs-def regular-pair-commute*)
have *low-density-pairs-swap*: (*R*,*S*) ∈ *low-density-pairs* ⟷ (*S*,*R*) ∈ *low-density-pairs* **for** *R S*
by (*simp add*: *low-density-pairs-def edge-density-commute edge-dense-def*)
(*use all-edges-between-swap* **in** *blast*)
have *small-pairs-swap*: (*R*,*S*) ∈ *small-pairs* ⟷ (*S*,*R*) ∈ *small-pairs* **for** *R S*
by (*auto simp*: *small-pairs-def*)

have *all-edges-if*:
all-edges-between R S Gnew
= (*if* (*R*,*S*) ∈ *irreg-pairs* ∪ *low-density-pairs* ∪ *small-pairs then* {}
else all-edges-between R S G)
(**is** *?lhs* = *?rhs*)
if *ij*: *R* ∈ *P S* ∈ *P* **for** *R S*
proof
show *?lhs* ⊆ *?rhs*
using *that fin-part* **unfolding** *Gnew-def Ea-def Eb-def Ec-def'*
apply (*simp add*: *all-edges-between-E-diff all-edges-between-E-Un all-edges-between-E-UN*)
apply (*auto simp*: *edge-def in-mk-uedge-img-iff all-edges-between-def*)
done
next

have *Ea*: *all-edges-between R S* (*V*, *Ea*) = {}
if (*R*,*S*) ∉ *irreg-pairs* **for** *V*
using *ij that P-disjnt*
by (*auto simp*: *Ea-def doubleton-eq-iff edge-def all-edges-between-def irreg-pairs-def*;
metis regular-pair-commute disjoint-iff-not-equal)
have *Eb*: *all-edges-between R S* (*V*, *Eb*) = {}
if (*R*,*S*) ∉ *low-density-pairs* **for** *V*
using *ij that*
apply (*auto simp*: *Eb-def edge-def all-edges-between-def low-density-pairs-def edge-dense-def*)
apply *metis*
by (*metis IntI P-disjnt doubleton-eq-iff edge-density-commute equals0D*)
have *Ec*: *all-edges-between R S* (*V*, *Ec*) = {}
if (*R*,*S*) ∉ *small-pairs* **for** *V*
using *ij that*
by (*auto simp*: *Ec-def′ doubleton-eq-iff edge-def all-edges-between-def small-pairs-def*;
metis P-disjnt disjoint-iff)
show *?rhs* ⊆ *?lhs*
by (*auto simp add*: *Gnew-def Ea Eb Ec all-edges-between-E-diff all-edges-between-E-Un*)
qed

have *rp*: *regular-pair R S Gnew* (*ε*/4) **if** *ij*: *R* ∈ *P S* ∈ *P* **for** *R S*
proof (*cases* (*R*,*S*) ∈ *irreg-pairs*)
case *False*
have *ed*: *edge-density X Y Gnew* =
(*if* (*R*,*S*) ∈ *irreg-pairs* ∪ *low-density-pairs* ∪ *small-pairs then 0*
else edge-density X Y G)
if *X* ⊆ *R Y* ⊆ *S* **for** *X Y*
using *all-edges-if that ij False* **by** (*smt* (*verit*) *all-edges-preserved edge-density-eq0 edge-density-def*)
show *?thesis*
using *that False* ‹*ε* > 0›
by (*auto simp add*: *irreg-pairs-def regular-pair-def less-le ed*)
next
case *True*
then have *ed*: *edge-density X Y Gnew* = *0* **if** *X* ⊆ *R Y* ⊆ *S* **for** *X Y*
by (*meson edge-density-eq0 all-edges-if that* ‹*R* ∈ *P*› ‹*S* ∈ *P*› *UnCI*)
with *egt that* **show** *?thesis*
by (*auto simp*: *regular-pair-def ed*)
qed
then have *reg-pairs*: *regular-graph P Gnew* (*ε*/4)
by (*meson regular-graph-def*)

have *edge-dense R S Gnew* (*ε*/2)
if *R* ∈ *P S* ∈ *P* **for** *R S*
proof (*cases* (*R*,*S*) ∈ *low-density-pairs*)

case *False*
have *ed*: *edge-density R S Gnew* =
(*if* (*R*,*S*) ∈ *irreg-pairs* ∪ *low-density-pairs* ∪ *small-pairs then 0*
else edge-density R S G)
using *all-edges-if that that* **by** (*simp add*: *edge-density-def*)
with *that* ‹*ε* > *0*› *False* **show** *?thesis*
by (*auto simp*: *low-density-pairs-def edge-dense-def all-edges-if*)
next
case *True*
then have *edge-density R S Gnew* = *0*
by (*simp add*: *all-edges-if edge-density-def that*)
with ‹*ε* > *0*› *that* **show** *?thesis*
by (*simp add*: *True all-edges-if edge-dense-def*)
qed
then have *density-bound*: *dense-graph P Gnew* (*ε*/*2*)
by (*meson dense-graph-def*)

have *min-subset-size*: *decent-graph P Gnew* (*?e4M* ∗ *?n*)
using ‹*ε* > *0*›
by (*auto simp*: *decent-graph-def small-pairs-def small-def decent-def all-edges-if*)
show *triangle-free-graph Gnew*
proof (*rule ccontr*)
assume *non*: ¬*?thesis*
then obtain *x y z* **where** *trig-ex*: *triangle-in-graph x y z Gnew*
using *triangle-free-graph-def non* **by** *auto*
then have *xin*: *x* ∈ (*uverts Gnew*) **and** *yin*: *y* ∈ (*uverts Gnew*) **and** *zin*: *z*
∈ (*uverts Gnew*)
using *triangle-in-graph-verts new-wf* **by** *auto*
then obtain *R* **where** *xinp*: *x* ∈ *R* **and** *ilt*: *R*∈*P*
using *graph-partition-new finite-graph-partition-obtain xin* **by** *metis*
then obtain *S* **where** *yinp*: *y* ∈ *S* **and** *jlt*: *S*∈*P*
using *graph-partition-new finite-graph-partition-obtain yin* **by** *metis*
then obtain *T* **where** *zinp*: *z* ∈ *T* **and** *klt*: *T*∈*P*
using *graph-partition-new finite-graph-partition-obtain zin* **by** *metis*
have *finitesubsets*: *finite R finite S finite T*
using *ilt jlt klt new-fin fin-part finite-graph-partition-finite fin* **by** *auto*
have *subsets*: *R* ⊆ *uverts Gnew S* ⊆ *uverts Gnew T* ⊆ *uverts Gnew*
using *finite-graph-partition-subset ilt jlt klt graph-partition-new* **by** *auto*
have *min-sizes*: *card R* ≥ *?e4M*∗*?n card S* ≥ *?e4M*∗*?n card T* ≥ *?e4M*∗*?n*
using *trig-ex min-subset-size xinp yinp zinp ilt jlt klt*
by (*auto simp*: *triangle-in-graph-def decent-graph-def decent-def all-edges-between-def*)
have *min-dens*: *edge-density R S Gnew* ≥ *ε*/*2 edge-density R T Gnew* ≥
ε/*2*
edge-density S T Gnew ≥ *ε*/*2*
using *density-bound subsets ilt jlt klt xinp yinp zinp* **unfolding** *dense-graph-def*
edge-dense-def
by (*metis all-edges-betw-I equals0D triangle-in-graph-def trig-ex*)+
then have *min-dens-diff*:
edge-density R S Gnew − *ε*/*4* ≥ *ε*/*4 edge-density R T Gnew* − *ε*/*4* ≥ *ε*/*4*

edge-density S T Gnew − *ε*/4 ≥ *ε*/4
by *auto*
have *mincard0*: (*card R*)∗(*card S*)∗ (*card T*) ≥ *0* **by** *simp*
have *gtcube*: ((*edge-density R S Gnew*) − *ε*/4)∗((*edge-density R T Gnew*) − *ε*/4) ∗((*edge-density S T Gnew*) − *ε*/4) ≥ (*ε*/4)^3
using *min-dens-diff e4gt real-mult-gt-cube* **by** *auto*
then have *c1*: ((*edge-density R S Gnew*) − *ε*/4)∗((*edge-density R T Gnew*) − *ε*/4) ∗((*edge-density S T Gnew*) − *ε*/4) ≥ *0*
by (*smt* (*verit*) *e4gt zero-less-power*)
have *?e4M* ∗ *?n* ≥ *0*
using *egt* **by** *force*
then have *card R* ∗ *card S* ∗ *card T* ≥ (*?e4M* ∗ *?n*)∗(*?e4M* ∗ *?n*) ∗ (*?e4M* ∗ *?n*)
by (*metis* (*no-types*) *of-nat-0-le-iff of-nat-mult min-sizes mult-mono*)
then have (*card R*)∗(*card S*)∗ (*card T*) ≥ (*?e4M*∗ *?n*)^3
by (*simp add*: *power3-eq-cube*)
then have *cardgtbound*:*card R* ∗ *card S* ∗ *card T* ≥ *?e4M*^ *3* ∗ *?n*^3
by (*metis of-nat-power power-mult-distrib*)

have (*1*−*ε*/2) ∗ (*ε*/4)^3 ∗ (*ε*/(*4*∗*M*))^3 ∗ *?n*^3 ≤ (*1*−*ε*/2) ∗ (*ε*/4)^3 ∗ *card R* ∗ *card S* ∗ *card T*
using *cardgtbound ordered-comm-semiring-class.comm-mult-left-mono True e4gt* **by** *fastforce*
also have ... ≤ (*1*−*2*∗(*ε*/4)) ∗ (*edge-density R S Gnew* − *ε*/4)∗(*edge-density R T Gnew* − *ε*/4) ∗ (*edge-density S T Gnew* − *ε*/4) ∗ *card R* ∗ *card S* ∗ *card T*
using *gtcube c1* ‹*ε* < *1*› *mincard0* **by** (*simp add*: *mult.commute mult.left-commute mult-left-mono*)
also have ... ≤ *card* (*triangle-triples R S T Gnew*)
by (*smt* (*verit*, *best*) *e4gt ilt jlt klt min-dens-diff new-fin new-wf rp subsets triangle-counting-lemma*)
finally have *card* (*triangle-set Gnew*) ≥ *D* ∗ *?n*^3
using *card-convert-triangle-rep-bound new-wf new-fin subsets*
by (*auto simp*: *triangle-triples-def D-def*)
then have *g-tset-bound*: *card* (*triangle-set G*) ≥ *D* ∗ *?n*^3
using *triangle-set-graph-edge-ss-bound* **by** (*smt* (*verit*) *edges fin local.wf of-nat-mono verts*)
have *card* (*triangle-set G*) > *δ* ∗ *?n*^3
proof −
have *?n*^3 > *0*
by (*simp add*: ‹*uverts G* ≠ {}› *card-gt-0-iff fin*)
with *δ* ‹*D0* ≤ *D*› **have** *D* ∗ *?n*^3 > *δ* ∗ *?n*^3
by *force*
thus *card* (*triangle-set G*) > *δ* ∗ *?n* ^3
using *g-tset-bound* **unfolding** *D-def* **by** *linarith*
qed
thus *False*
using *ineq* **by** *linarith*
qed
show *real* (*card* (*uedges G* − *uedges Gnew*)) ≤ *ε* ∗ *real* ((*card* (*uverts G*))^2)

using *cardbound edges verts* **by** *blast*
qed
qed (*rule* ‹*0 < δ*›)
qed

1.6 Roth's Theorem

We will first need the following corollary of the Triangle Removal Lemma. This is Corollary 3.18 in Zhao's notes:

corollary *corollary-triangle-removal*:
fixes *ε* :: *real*
assumes *0 < ε*
shows ∃ *N*>*0*. ∀ *G*. *card*(*uverts G*) > *N* ⟶ *uwellformed G* ⟶ *unique-triangles G* ⟶
card (*uedges G*) ≤ *ε* ∗ (*card* (*uverts G*))^2
proof −
have *ε*/*3* > *0*
using *assms* **by** *auto*
then obtain *δ*::*real* **where** *δ* > *0*
and *δ*: ⋀*G*. ⟦*card*(*uverts G*) > *0*; *uwellformed G*;
card (*triangle-set G*) ≤ *δ* ∗ *card*(*uverts G*) ^ *3*⟧ ⟹
(∃ *Gnew*. *triangle-free-graph Gnew* ∧ *uverts Gnew* = *uverts G* ∧ (*uedges Gnew* ⊆ *uedges G*) ∧
card (*uedges G* − *uedges Gnew*) ≤ *ε*/*3* ∗ (*card* (*uverts G*))^2)
using *triangle-removal-lemma* **by** *metis*
obtain *N*::*nat* **where** *N*: *real N* ≥ *1* / (*3*∗*δ*)
by (*meson real-arch-simple*)
show *?thesis*
proof (*intro exI conjI strip*)
show *N* > *0*
using *N* ‹*0 < δ*› *zero-less-iff-neq-zero* **by** *fastforce*
fix *G*
let *?n* = *card* (*uverts G*)
assume *G-gt-N*: *N* < *?n*
and *wf*: *uwellformed G*
and *uniq*: *unique-triangles G*
have *G-ne*: *?n* > *0*
using *G-gt-N* **by** *linarith*
obtain *TF* **where** *TF*: ⋀*e*. *e* ∈ *uedges G* ⟹ ∃ *x y z*. *TF e* = {*x*,*y*,*z*} ∧ *triangle-in-graph x y z G* ∧ *e* ⊆ *TF e*
using *uniq* **unfolding** *unique-triangles-def* **by** *metis*
let *?TWO* = (λ*t*. [*t*]^2)
have *tri-nsets-2*: [{*x*,*y*,*z*}]^2 = {{*x*,*y*},{*y*,*z*},{*x*,*z*}} **if** *triangle-in-graph x y z G* **for** *x y z*
using *that* **unfolding** *nsets-def triangle-in-graph-def card-2-iff doubleton-eq-iff*
by (*blast dest*!: *edge-vertices-not-equal* [*OF wf*])
have *tri-nsets-3*: {{*x*,*y*},{*y*,*z*},{*x*,*z*}} ∈ [*uedges G*]^3 **if** *triangle-in-graph x y z G* **for** *x y z*
using *that*

by (*simp add*: *nsets-def card-3-iff triangle-in-graph-def*) (*metis doubleton-eq-iff*

edge-vertices-not-equal [*OF wf*])
have *sub*: *?TWO ‘ triangle-set G* $\subseteq$ [*uedges G*]^3
using *tri-nsets-2 tri-nsets-3 triangle-set-def* **by** *auto*
have $\bigwedge i.\ i \in$ *triangle-set G* $\Longrightarrow$ *?TWO i* $\neq \{\}$
using *tri-nsets-2 triangle-set-def* **by** *auto*
moreover have *dfam*: *disjoint-family-on ?TWO* (*triangle-set G*)
using *sub* [*unfolded image-subset-iff*] *uniq*
unfolding *disjoint-family-on-def triangle-set-def nsets-def unique-triangles-def*
by (*smt* (*verit*) *disjoint-iff-not-equal insert-subset mem-Collect-eq mk-disjoint-insert*
)
ultimately have *inj*: *inj-on ?TWO* (*triangle-set G*)
by (*simp add*: *disjoint-family-on-iff-disjoint-image*)
have §: $\exists$ *T*$\in$*triangle-set G*. *e* $\in$ [*T*]^2 **if** *e* $\in$ *uedges G* **for** *e*
using *uniq* [*unfolded unique-triangles-def*] *that local.wf*
apply (*simp add*: *triangle-set-def triangle-in-graph-def nsets-def uwellformed-def*)
by (*metis* (*mono-tags*, *lifting*) *finite.emptyI finite.insertI finite-subset*)
with *sub* **have** $\bigcup$(*?TWO ‘ triangle-set G*) = *uedges G*
by (*auto simp*: *image-subset-iff nsets-def*)
then have *card* ($\bigcup$(*?TWO ‘ triangle-set G*)) = *card* (*uedges G*)
by *simp*
moreover have *card* ($\bigcup$(*?TWO ‘ triangle-set G*)) = *3* $*$ *card* (*triangle-set G*)
proof (*subst card-UN-disjoint'* [*OF dfam*])
show *finite* ([*i*]^2) **if** *i* $\in$ *triangle-set G* **for** *i*
using *that tri-nsets-2 triangle-set-def* **by** *fastforce*
show *finite* (*triangle-set G*)
by (*meson G-ne card-gt-0-iff local.wf triangle-set-finite*)
have *card* ([*i*]^2) = *3* **if** *i* $\in$ *triangle-set G* **for** *i*
using *that wf*
unfolding *triangle-set-def triangle-in-graph-def uwellformed-def*
by (*smt* (*z3*) *image-subset-iff mem-Collect-eq nsets-def sub that*)
then show ($\sum$ *i*$\in$*triangle-set G*. *card* ([*i*]^2)) = *3* $*$ *card* (*triangle-set G*)
by *simp*
qed
ultimately have *A*: *3* $*$ *card* (*triangle-set G*) = *card* (*uedges G*)
by *auto*
have *card* (*uedges G*) $\leq$ *card* (*all-edges*(*uverts G*))
by (*meson G-ne all-edges-finite card-gt-0-iff card-mono local.wf wellformed-all-edges*)
also have ... = *card* (*uverts G*) *choose 2*
by (*metis G-ne card-all-edges card-eq-0-iff not-less0*)
also have ... = *card* (*uverts G*) $*$ (*card* (*uverts G*) $-$ *1*) *div 2*
by (*meson n-choose-2-nat*)
also have ... $<$ (*card* (*uverts G*))^2
by (*simp add*: *G-ne less-imp-diff-less less-mult-imp-div-less power2-eq-square*)
finally have *B*: *card* (*uedges G*) $<$ (*card* (*uverts G*))^2 **.**

have *card* (*triangle-set G*) $\leq$ (*card* (*uverts G*))^2 / *3*
using *A B* **by** *linarith*

```
  also have ... ≤ δ * card(uverts G) ^ 3
  proof −
    have 1 ≤ 3 * δ * N
      using N ‹δ > 0› by (simp add: field-simps)
    also have ... ≤ 3 * δ * ?n
      using G-gt-N ‹0 < δ› by force
    finally have 1 * ?n^2 ≤ (3 * δ * ?n) * ?n^2
      by (simp add: G-ne)
    then show ?thesis
      by (simp add: eval-nat-numeral mult-ac)
  qed
  finally have card (triangle-set G) ≤ δ * ?n ^ 3 .
   then obtain Gnew where Gnew: triangle-free-graph Gnew uverts Gnew =
uverts G
    uedges Gnew ⊆ uedges G and card-edge-diff: card (uedges G − uedges Gnew)
≤ ε/3 * ?n^2
    using G-ne δ local.wf by meson
```

Deleting an edge removes at most one triangle from the graph by assumption, so the number of edges removed in this process is at least the number of triangles.

```
  have False
    if non: ⋀e. e ∈ uedges G − uedges Gnew ⟹ {x,y,z} ≠ TF e
      and tri: triangle-in-graph x y z G for x y z
  proof −
    have ¬ triangle-in-graph x y z Gnew
      using Gnew triangle-free-graph-def by blast
    with tri obtain e where eG: e ∈ uedges G − uedges Gnew and esub: e ⊆
{x,y,z}
      using insert-commute triangle-in-graph-def by auto
    then show False
      by (metis DiffD1 TF tri uniq unique-triangles-def non [OF eG])
  qed
  then have triangle-set G ⊆ TF ‘ (uedges G − uedges Gnew)
    unfolding triangle-set-def by blast
  moreover have finite (uedges G − uedges Gnew)
    by (meson G-ne card-gt-0-iff finite-Diff finite-graph-def wf wellformed-finite)
  ultimately have card (triangle-set G) ≤ card (uedges G − uedges Gnew)
    by (meson surj-card-le)
  then show card (uedges G) ≤ ε * ?n^2
    using A card-edge-diff by linarith
 qed
qed
```

We are now ready to proceed to the proof of Roth's Theorem for Arithmetic Progressions.

```
definition progression3 :: 'a::comm-monoid-add ⇒ 'a ⇒ 'a set
 where progression3 k d ≡ {k, k+d, k+d+d}
```

lemma *p3-int-iff*: *progression3* (*int k*) (*int d*) ⊆ *int* ‘ *A* ⟷ *progression3 k d* ⊆ *A*
apply (*simp add*: *progression3-def image-iff*)
by (*smt* (*verit*, *best*) *int-plus of-nat-eq-iff*)

We assume that a set of naturals $A \subseteq \{... < N\}$ does not have any arithmetic progression. We will then show that A is of cardinality $o(N)$.

lemma *RothArithmeticProgressions-aux*:
fixes *ε*::*real*
assumes *ε* > *0*
obtains *X* **where** ∀ *N* ≥ *X*. ∀ *A* ⊆ {..<*N*}. (∄ *k d*. *d*>*0* ∧ *progression3 k d* ⊆ *A*) ⟶ *card A* < *ε* ∗ *real N*
proof −
obtain *X* **where** *X*>*0*
and *X*: ⋀*G*. ⟦*card*(*uverts G*) > *X*; *uwellformed G*; *unique-triangles G*⟧
⟹ *card* (*uedges G*) ≤ *ε*/*12* ∗ (*card* (*uverts G*))²
by (*metis assms corollary-triangle-removal less-divide-eq-numeral1*(*1*) *mult-eq-0-iff*)
show *thesis*
proof (*intro strip that*)
fix *N A*
assume *X* ≤ *N* **and** *A*: *A* ⊆ {..<*N*}
and *non*: ∄ *k d*. *0* < *d* ∧ *progression3 k d* ⊆ *A*
then have *N* > *0* **using** ‹*0* < *X*› **by** *linarith*
define *M* **where** *M* ≡ *Suc* (*2*∗*N*)
have *M-mod-bound*[*simp*]: *x mod M* < *M* **for** *x*
by (*simp add*: *M-def*)
have *odd M M*>*0 N*<*M* **by** (*auto simp*: *M-def*)
have *coprime M* (*Suc N*)
unfolding *M-def*
by (*metis add.commute coprime-Suc-right-nat coprime-mult-right-iff mult-2 nat-arith.suc1*)
then have *cop*: *coprime M* (*1* + *int N*)
by (*metis coprime-int-iff of-nat-Suc*)
have *A-sub-M*: *int* ‘ *A* ⊆ {..<*M*}
using *A* **by** (*force simp*: *M-def*)
have *non-img-A*: ∄ *k d*. *d* > *0* ∧ *progression3 k d* ⊆ *int* ‘ *A*
by (*metis p3-int-iff non pos-int-cases zero-le-imp-eq-int imageE insert-subset of-nat-0-le-iff progression3-def*)

Construct a tripartite graph G whose three parts are copies of $\mathbb{Z}/M\mathbb{Z}$.

define *part-of* **where** *part-of* ≡ λξ. (λ*i*. *prod-encode* (ξ,*i*)) ‘ {..<*M*}
define *label-of-part* **where** *label-of-part* ≡ λ*p*. *fst* (*prod-decode p*)
define *from-part* **where** *from-part* ≡ λ*p*. *snd* (*prod-decode p*)
have *enc-iff* [*simp*]: *prod-encode* (*a*,*i*) ∈ *part-of a*′ ⟷ *a*′=*a* ∧ *i*<*M* **for** *a a*′ *i*
using ‹*0* < *M*› **by** (*clarsimp simp*: *part-of-def image-iff Bex-def*) *presburger*
have *part-of-M*: *p* ∈ *part-of a* ⟹ *from-part p* < *M* **for** *a p*
using *from-part-def part-of-def* **by** *fastforce*
have *disjnt-part-of*: *a* ≠ *b* ⟹ *disjnt* (*part-of a*) (*part-of b*) **for** *a b*

by (*auto simp*: *part-of-def disjnt-iff*)
have *from-enc* [*simp*]: *from-part* (*prod-encode* (*a*,*i*)) = *i* **for** *a i*
by (*simp add*: *from-part-def*)
have *finpart* [*iff*]: *finite* (*part-of a*) **for** *a*
by (*simp add*: *part-of-def* ‹*0* < *M*›)
have *cardpart* [*simp*]: *card* (*part-of a*) = *M* **for** *a*
using ‹*0* < *M*›
by (*simp add*: *part-of-def eq-nat-nat-iff inj-on-def card-image*)
let *?X* = *part-of 0*
let *?Y* = *part-of* (*Suc 0*)
let *?Z* = *part-of* (*Suc* (*Suc 0*))
define *diff* **where** *diff* ≡ λ*a b*. (*int a* − *int b*) *mod* (*int M*)
have *inj-on-diff*: *inj-on* (λ*x*. *diff x a*) {..<*M*} **for** *a*
apply (*clarsimp simp*: *diff-def inj-on-def*)
by (*metis diff-add-cancel mod-add-left-eq mod-less nat-int of-nat-mod*)
have *eq-mod-M*: (*x* − *y*) *mod int M* = (*x'* − *y*) *mod int M* ⟹ *x mod int M* = *x'* *mod int M* **for** *x x' y*
by (*simp add*: *mod-eq-dvd-iff*)

have *diff-invert*: *diff y x* = *int a* ⟷ *y* = (*x* + *a*) *mod M* **if** *y* < *M a*∈*A* **for** *x y a*
proof −
have *a* < *M*
using *A* ‹*N* < *M*› *that* **by** *auto*
show *?thesis*
proof
assume *diff y x* = *int a*
with *that* ‹*a*<*M*› **have** *int y* = *int* (*x*+*a*) *mod int M*
unfolding *diff-def* **by** (*smt* (*verit*, *ccfv-SIG*) *eq-mod-M mod-less of-nat-add zmod-int*)
with *that* **show** *y* = (*x* + *a*) *mod M*
by (*metis nat-int zmod-int*)
qed (*simp add*: ‹*a* < *M*› *diff-def mod-diff-left-eq zmod-int*)
qed

define *diff2* **where** *diff2* ≡ λ*a b*. ((*int a* − *int b*) ∗ *int*(*Suc N*)) *mod* (*int M*)
have *inj-on-diff2*: *inj-on* (λ*x*. *diff2 x a*) {..<*M*} **for** *a*
apply (*clarsimp simp*: *diff2-def inj-on-def*)
by (*metis eq-mod-M mult-mod-cancel-right* [*OF - cop*] *int-int-eq mod-less zmod-int*)
have [*simp*]: (*1* + *int N*) *mod int M* = *1* + *int N*
using *M-def* ‹*0* < *N*› **by** *auto*
have *diff2-by2*: (*diff2 a b* ∗ *2*) *mod M* = *diff a b* **for** *a b*
proof −
have *int M dvd* ((*int a* − *int b*) ∗ *int M*)
by *simp*
then have *int M dvd* ((*int a* − *int b*) ∗ *int* (*Suc N*) ∗ *2* − (*int a* − *int b*))
by (*auto simp*: *M-def algebra-simps*)
then show *?thesis*

by (*metis diff2-def diff-def mod-eq-dvd-iff mod-mult-left-eq*)
qed
have *diff2-invert*: *diff2* (((*x* + *a*) *mod M* + *a*) *mod M*) *x* = *int a* **if** *a*∈*A* **for** *x a*
proof −
have *1*: ((*x* + *a*) *mod M* + *a*) *mod M* = (*x* + *2*∗*a*) *mod M*
by (*metis group-cancel.add1 mod-add-left-eq mult-2*)
have (*int* ((*x* + *2*∗*a*) *mod M*) − *int x*) ∗ (*1* + *int N*) *mod int M*
= (*int* (*x* + *2*∗*a*) − *int x*) ∗ (*1* + *int N*) *mod int M*
by (*metis mod-diff-left-eq mod-mult-cong of-nat-mod*)
also have ... = *int* (*a* ∗ (*Suc M*)) *mod int M*
by (*simp add*: *algebra-simps M-def*)
also have ... = *int a mod int M*
by *simp*
also have ... = *int a*
using *A M-def subsetD that* **by** *auto*
finally show *?thesis*
using *that* **by** (*auto simp*: *1 diff2-def*)
qed

define *Edges* **where** *Edges* ≡ λ*X Y df*. {{*x*,*y*}| *x y*. *x* ∈ *X* ∧ *y* ∈ *Y* ∧ *df*(*from-part y*) (*from-part x*) ∈ *int* ‘ *A*}
have *Edges-subset*: *Edges X Y df* ⊆ *Pow* (*X* ∪ *Y*) **for** *X Y df*
by (*auto simp*: *Edges-def*)
define *XY* **where** *XY* ≡ *Edges ?X ?Y diff*
define *YZ* **where** *YZ* ≡ *Edges ?Y ?Z diff*
define *XZ* **where** *XZ* ≡ *Edges ?X ?Z diff2*
obtain [*simp*]: *finite XY finite YZ finite XZ*
using *Edges-subset* **unfolding** *XY-def YZ-def XZ-def*
by (*metis finite-Pow-iff finite-UnI finite-subset finpart*)
define *G* **where** *G* ≡ (*?X* ∪ *?Y* ∪ *?Z*, *XY* ∪ *YZ* ∪ *XZ*)
have *finG*: *finite* (*uverts G*) **and** *cardG*: *card* (*uverts G*) = *3*∗*M*
by (*simp-all add*: *G-def card-Un-disjnt disjnt-part-of*)
then have *card*(*uverts G*) > *X*
using *M-def* ‹*X* ≤ *N*› **by** *linarith*
have *uwellformed G*
by (*fastforce simp*: *card-insert-if part-of-def G-def XY-def YZ-def XZ-def Edges-def uwellformed-def*)
have [*simp*]: {*prod-encode* (ξ,*x*), *prod-encode* (ξ,*y*)} ∉ *XY*
{*prod-encode* (ξ,*x*), *prod-encode* (ξ,*y*)} ∉ *YZ*
{*prod-encode* (ξ,*x*), *prod-encode* (ξ,*y*)} ∉ *XZ* **for** *x y* ξ
by (*auto simp*: *XY-def YZ-def XZ-def Edges-def doubleton-eq-iff*)
have *label-ne-XY* [*simp*]: *label-of-part p* ≠ *label-of-part q* **if** {*p*,*q*} ∈ *XY* **for** *p q*
using *that* **by** (*auto simp add*: *XY-def part-of-def Edges-def doubleton-eq-iff label-of-part-def*)
then have [*simp*]: {*p*} ∉ *XY* **for** *p*
by (*metis insert-absorb2*)
have *label-ne-YZ* [*simp*]: *label-of-part p* ≠ *label-of-part q* **if** {*p*,*q*} ∈ *YZ* **for** *p q*

using *that* **by** (*auto simp add*: *YZ-def part-of-def Edges-def doubleton-eq-iff label-of-part-def*)
then have [*simp*]: $\{p\} \notin YZ$ **for** p
by (*metis insert-absorb2*)
have *label-ne-XZ* [*simp*]: *label-of-part* $p \neq$ *label-of-part* q **if** $\{p,q\} \in XZ$ **for** p q
using *that* **by** (*auto simp add*: *XZ-def part-of-def Edges-def doubleton-eq-iff label-of-part-def*)
then have [*simp*]: $\{p\} \notin XZ$ **for** p
by (*metis insert-absorb2*)
have *label012*: *label-of-part* $v < 3$ **if** $v \in$ *uverts* G **for** v
using *that* **by** (*auto simp add*: *G-def eval-nat-numeral part-of-def label-of-part-def*)

have *Edges-distinct*: $\bigwedge p\ q\ r\ \xi\ \zeta\ \gamma\ \beta\ df\ df'$. $[\![\{p,q\} \in Edges\ (part\text{-}of\ \xi)\ (part\text{-}of\ \zeta)\ df;$
$\{q,r\} \in Edges\ (part\text{-}of\ \xi)\ (part\text{-}of\ \zeta)\ df;$
$\{p,r\} \in Edges\ (part\text{-}of\ \gamma)\ (part\text{-}of\ \beta)\ df';\ \xi \neq \zeta;\ \gamma \neq \beta]\!] \Longrightarrow False$
apply (*auto simp*: *disjnt-iff Edges-def doubleton-eq-iff conj-disj-distribR ex-disj-distrib*)
apply (*metis disjnt-iff disjnt-part-of*)+
done
have *uniq*: $\exists\, i{<}M.\ \exists\, d{\in}A.\ \exists\, x \in \{p,q,r\}.\ \exists\, y \in \{p,q,r\}.\ \exists\, z \in \{p,q,r\}.$
$x = prod\text{-}encode(0,\ i)$
$\wedge\ y = prod\text{-}encode(1,\ (i+d)\ mod\ M)$
$\wedge\ z = prod\text{-}encode(2,\ (i+d+d)\ mod\ M)$
if T: *triangle-in-graph* $p\ q\ r\ G$ **for** $p\ q\ r$
proof −
obtain $x\ y\ z$ **where** xy: $\{x,y\} \in XY$ **and** yz: $\{y,z\} \in YZ$ **and** xz: $\{x,z\} \in XZ$
and x: $x \in \{p,q,r\}$ **and** y: $y \in \{p,q,r\}$ **and** z: $z \in \{p,q,r\}$
using T **apply** (*simp add*: *triangle-in-graph-def G-def XY-def YZ-def XZ-def*)
by (*smt* (*verit*, *ccfv-SIG*) *Edges-distinct Zero-not-Suc insert-commute n-not-Suc-n*)
then have $x \in ?X\ y \in ?Y\ z \in ?Z$
by (*auto simp*: *XY-def YZ-def XZ-def Edges-def doubleton-eq-iff*; *metis disjnt-iff disjnt-part-of*)+
then obtain $i\ j\ k$ **where** i: $x = prod\text{-}encode(0,i)$ **and** j: $y = prod\text{-}encode(1,j)$
and k: $z = prod\text{-}encode(2,k)$
by (*metis One-nat-def Suc-1 enc-iff prod-decode-aux.cases prod-decode-inverse*)
obtain *a1* **where** $a1 \in A$ **and** $a1$: $(int\ j - int\ i)\ mod\ int\ M = int\ a1$
using xy ‹$x \in ?X$› $i\ j$ **by** (*auto simp add*: *XY-def Edges-def doubleton-eq-iff diff-def*)
obtain *a3* **where** $a3 \in A$ **and** $a3$: $(int\ k - int\ j)\ mod\ int\ M = int\ a3$
using yz ‹$x \in ?X$› $j\ k$ **by** (*auto simp add*: *YZ-def Edges-def doubleton-eq-iff diff-def*)
obtain *a2* **where** $a2 \in A$ **and** $a2$: $(int\ k - int\ i)\ mod\ int\ M = int\ (a2 * 2)\ mod\ int\ M$
using xz ‹$x \in ?X$› $i\ k$ **apply** (*auto simp add*: *XZ-def Edges-def doubleton-eq-iff*)

by (*metis diff2-by2 diff-def int-plus mult-2-right*)
obtain *a1<N a2<N a3<N*
using *A* ‹*a1* ∈ *A*› ‹*a2* ∈ *A*› ‹*a3* ∈ *A*› **by** *blast*
then obtain *a1+a3* < *M a2* ∗ *2* < *M*
by (*simp add*: *M-def*)
then have *int* (*a2* ∗ *2*) = *int* (*a2* ∗ *2*) *mod M*
by *force*
also have ... = *int* (*a1* + *a3*) *mod int M*
using *a1 a2 a3* **by** (*smt* (*verit*, *del-insts*) *int-plus mod-add-eq*)
also have ... = *int* (*a1+a3*)
using ‹*a1* + *a3* < *M*› **by** *force*
finally have *a2∗2* = *a1+a3*
by *presburger*
then obtain *equal*: *a3* − *a2* = *a2* − *a1 a2* − *a3* = *a1* − *a2*
by (*metis Nat.diff-cancel diff-cancel2 mult-2-right*)
with ‹*a1* ∈ *A*› ‹*a2* ∈ *A*› ‹*a3* ∈ *A*› **have** *progression3 a1* (*a2* − *a1*) ⊆ *A*
apply (*clarsimp simp*: *progression3-def*)
by (*metis diff-is-0-eq′ le-add-diff-inverse nle-le*)
with *non equal* **have** *a2* = *a1*
unfolding *progression3-def*
by (*metis* ‹*a2* ∈ *A*› ‹*a3* ∈ *A*› *add.right-neutral diff-is-0-eq insert-subset*
le-add-diff-inverse not-gr-zero)
then have *a3* = *a2*
using ‹*a2* ∗ *2* = *a1* + *a3*› **by** *force*
have *k-minus-j*: (*int k* − *int j*) *mod int M* = *int a1*
by (*simp add*: ‹*a2* = *a1*› ‹*a3* = *a2*› *a3*)
have *i-to-j*: *j mod M* = (*i*+*a1*) *mod M*
by (*metis a1 add-diff-cancel-left′ add-diff-eq mod-add-right-eq nat-int of-nat-add*
of-nat-mod)
have *j-to-k*: *k mod M* = (*j*+*a1*) *mod M*
by (*metis* ‹*a2* = *a1*› ‹*a3* = *a2*› *a3 add-diff-cancel-left′ add-diff-eq mod-add-right-eq*

nat-int of-nat-add of-nat-mod)
have *i<M*
using ‹*x* ∈ *?X*› *i* **by** *simp*
then show *?thesis*
using *i j k x y z* ‹*a1* ∈ *A*›
by (*metis* ‹*y* ∈ *?Y*› ‹*z* ∈*?Z*› *enc-iff i-to-j j-to-k mod-add-left-eq mod-less*)
qed

Every edge of the graph G lies in exactly one triangle.

have *unique-triangles G*
unfolding *unique-triangles-def*
proof (*intro strip*)
fix *e*
assume *e* ∈ *uedges G*
then consider *e* ∈ *XY* | *e* ∈ *YZ* | *e* ∈ *XZ*
using *G-def* **by** *fastforce*
then show ∃!*T*. ∃ *x y z*. *T* = {*x*, *y*, *z*} ∧ *triangle-in-graph x y z G* ∧ *e* ⊆ *T*

proof *cases*
case *1*
then obtain $i\ j\ a$ **where** *eeq*: $e = \{$*prod-encode*$(0,i)$, *prod-encode*$(1,j)\}$
and $i<M$ **and** $j<M$
and *df*: *diff* $j\ i =$ *int* a **and** $a \in A$
by (*auto simp*: *XY-def Edges-def part-of-def*)
let *?x* = *prod-encode* $(0,\ i)$
let *?y* = *prod-encode* $(1,\ j)$
let *?z* = *prod-encode* $(2,\ (j+a)\ mod\ M)$
have *yeq*: $j = (i+a)\ mod\ M$
using *diff-invert* **using** ‹$a \in A$› *df* ‹$j<M$› **by** *blast*
with ‹$a \in A$› ‹$j<M$› **have** $\{$*?y*,*?z*$\} \in YZ$
by (*fastforce simp*: *YZ-def Edges-def image-iff diff-invert*)
moreover have $\{$*?x*,*?z*$\} \in XZ$
using ‹$a \in A$› **by** (*fastforce simp*: *XZ-def Edges-def yeq diff2-invert* ‹$i<M$›)
ultimately have *T*: *triangle-in-graph ?x ?y ?z G*
using ‹$e \in$ *uedges G*› **by** (*force simp add*: *G-def eeq triangle-in-graph-def*)
show *?thesis*
proof (*intro ex1I*)
show $\exists x\ y\ z.\ \{$*?x*,*?y*,*?z*$\} = \{x,\ y,\ z\} \wedge$ *triangle-in-graph* $x\ y\ z\ G \wedge e \subseteq \{$*?x*,*?y*,*?z*$\}$
using *T eeq* **by** *blast*
fix *T*
assume $\exists p\ q\ r.\ T = \{p,\ q,\ r\} \wedge$ *triangle-in-graph* $p\ q\ r\ G \wedge e \subseteq T$
then obtain $p\ q\ r$ **where** *Teq*: $T = \{p,q,r\}$
and *tri*: *triangle-in-graph* $p\ q\ r\ G$ **and** $e \subseteq T$
by *blast*
with *uniq*
obtain $i'\ a'\ x\ y\ z$ **where** $i'<M\ a' \in A$
and *x*: $x \in \{p,q,r\}$ **and** *y*: $y \in \{p,q,r\}$ **and** *z*: $z \in \{p,q,r\}$
and *xeq*: $x =$ *prod-encode*$(0,\ i')$
and *yeq*: $y =$ *prod-encode*$(1,\ (i'+a')\ mod\ M)$
and *zeq*: $z =$ *prod-encode*$(2,\ (i'+a'+a')\ mod\ M)$
by *metis*
then have *sets-eq*: $\{x,y,z\} = \{p,q,r\}$ **by** *auto*
with *Teq* ‹$e \subseteq T$› **have** *esub'*: $e \subseteq \{x,y,z\}$ **by** *blast*
have $a' < M$
using A ‹$N < M$› ‹$a' \in A$› **by** *auto*
obtain *?x* $\in e$ *?y* $\in e$ **using** *eeq* **by** *force*
then have $x =$ *?x*
using *esub' eeq yeq zeq* **by** *simp*
then have $y =$ *?y*
using *esub' eeq zeq* **by** *simp*
obtain *eq'*: $i' = i\ (i'+a')\ mod\ M = j$
using ‹$x =$ *?x*› *xeq* **using** ‹$y =$*?y*› *yeq* **by** *auto*
then have *diff* $(i'+a')\ i' =$ *int* a'
by (*simp add*: *diff-def* ‹$a' < M$›)
then have $a' = a$
by (*metis eq' df diff-def mod-diff-left-eq nat-int zmod-int*)

```
      then have z = ?z
        by (metis ‹y = ?y› mod-add-left-eq prod-encode-eq snd-conv yeq zeq)
      then show T = {?x,?y,?z}
        using Teq ‹x = ?x› ‹y = ?y› sets-eq by presburger
    qed
  next
    case 2
    then obtain j k a where eeq: e = {prod-encode(1,j), prod-encode(2,k)}
      and j<M k<M
      and df: diff k j = int a and a ∈ A
      by (auto simp: YZ-def Edges-def part-of-def numeral-2-eq-2)
    let ?x = prod-encode (0, (M+j−a) mod M)
    let ?y = prod-encode (1, j)
    let ?z = prod-encode (2, k)
    have zeq: k = (j+a) mod M
      using diff-invert using ‹a ∈ A› df ‹k<M› by blast
    with ‹a ∈ A› ‹k<M› have {?x,?z} ∈ XZ
      unfolding XZ-def Edges-def image-iff
       apply (clarsimp simp: mod-add-left-eq doubleton-eq-iff conj-disj-distribR
ex-disj-distrib)
     apply (smt (verit, ccfv-threshold) A ‹N < M› diff2-invert le-add-diff-inverse2
lessThan-iff
linorder-not-less mod-add-left-eq
                mod-add-self1 not-add-less1 order.strict-trans subsetD)
      done
    moreover
    have a < N using A ‹a ∈ A› by blast
    with ‹N < M› have ((M + j − a) mod M + a) mod M = j mod M
      by (simp add: mod-add-left-eq)
    then have {?x,?y} ∈ XY
      using ‹a ∈ A› ‹j<M›
      by (force simp add: XY-def Edges-def zeq image-iff diff-invert
doubleton-eq-iff ex-disj-distrib)
    ultimately have T: triangle-in-graph ?x ?y ?z G
      using ‹e ∈ uedges G› by (auto simp: G-def eeq triangle-in-graph-def)
    show ?thesis
    proof (intro ex1I)
      show ∃x y z. {?x,?y,?z} = {x, y, z} ∧ triangle-in-graph x y z G ∧ e ⊆
{?x,?y,?z}
       using T eeq by blast
      fix T
      assume ∃p q r. T = {p, q, r} ∧ triangle-in-graph p q r G ∧ e ⊆ T
      then obtain p q r where Teq: T = {p,q,r} and tri: triangle-in-graph p
q r G and e ⊆ T
        by blast
      with uniq
      obtain i' a' x y z where i'<M a' ∈ A
          and x: x ∈ {p,q,r} and y: y ∈ {p,q,r} and z: z ∈ {p,q,r}
          and xeq: x = prod-encode(0, i')
```

```
            and yeq: y = prod-encode(1, (i'+a') mod M)
            and zeq: z = prod-encode(2, (i'+a'+a') mod M)
          by metis
        then have sets-eq: {x,y,z} = {p,q,r} by auto
        with Teq ‹e ⊆ T› have esub': e ⊆ {x,y,z} by blast
        have a' < M
          using A ‹N < M› ‹a' ∈ A› by auto
        obtain ?y ∈ e ?z ∈ e
          using eeq by force
        then have y = ?y
          using esub' eeq xeq zeq by simp
        then have z = ?z
          using esub' eeq xeq by simp
        obtain eq': (i'+a') mod M = j (i'+a'+a') mod M = k
          using ‹y = ?y› yeq using ‹z =?z› zeq by auto
        then have diff (i'+a'+a') (i'+a') = int a'
          by (simp add: diff-def ‹a' < M›)
        then have a' = a
           by (metis M-mod-bound ‹a' ∈ A› df diff-invert eq' mod-add-eq mod-if
of-nat-eq-iff)
        have (M + ((i'+a') mod M) - a') mod M = i'
            by (metis Nat.add-diff-assoc2 ‹a' < M› ‹i' < M› add.left-commute
add-implies-diff
less-imp-le-nat mod-add-right-eq mod-add-self2 mod-less)
        with ‹a' = a› eq' have (M + j - a) mod M = i'
          by force
        with xeq have x = ?x by blast
        then show T = {?x,?y,?z}
          using Teq ‹z = ?z› ‹y = ?y› sets-eq by presburger
      qed
    next
      case 3
      then obtain i k a where eeq: e = {prod-encode(0,i), prod-encode(2,k)}
        and i<M and k<M
        and df: diff2 k i = int a and a ∈ A
        by (auto simp: XZ-def Edges-def part-of-def eval-nat-numeral)
      let ?x = prod-encode (0, i)
      let ?y = prod-encode (1, (i+a) mod M)
      let ?z = prod-encode (2, k)
      have keq: k = (i+a+a) mod M
        using diff2-invert [OF ‹a ∈ A›, of i] df ‹k<M› using inj-on-diff2 [of i]
        by (simp add: inj-on-def Ball-def mod-add-left-eq)
      with ‹a ∈ A› have {?x,?y} ∈ XY
        using ‹a ∈ A› ‹i<M› ‹k<M› apply (auto simp: XY-def Edges-def)
        by (metis M-mod-bound diff-invert enc-iff from-enc imageI)
      moreover have {?y,?z} ∈ YZ
        apply (auto simp: YZ-def Edges-def image-iff eval-nat-numeral)
       by (metis M-mod-bound ‹a ∈ A› diff-invert enc-iff from-enc mod-add-left-eq
keq)
```

```
      ultimately have T: triangle-in-graph ?x ?y ?z G
       using ‹e ∈ uedges G› by (force simp add: G-def eeq triangle-in-graph-def)
      show ?thesis
      proof (intro ex1I)
        show ∃x y z. {?x,?y,?z} = {x, y, z} ∧ triangle-in-graph x y z G ∧ e ⊆
{?x,?y,?z}
         using T eeq by blast
        fix T
        assume ∃p q r. T = {p, q, r} ∧ triangle-in-graph p q r G ∧ e ⊆ T
        then obtain p q r where Teq: T = {p,q,r} and tri: triangle-in-graph p
q r G and e ⊆ T
         by blast
        with uniq obtain i' a' x y z where i'<M a' ∈ A
             and x: x ∈ {p,q,r} and y: y ∈ {p,q,r} and z: z ∈ {p,q,r}
             and xeq: x = prod-encode(0, i')
             and yeq: y = prod-encode(1, (i'+a') mod M)
             and zeq: z = prod-encode(2, (i'+a'+a') mod M)
         by metis
        then have sets-eq: {x,y,z} = {p,q,r} by auto
        with Teq ‹e ⊆ T› have esub': e ⊆ {x,y,z} by blast
        have a' < M
         using A ‹N < M› ‹a' ∈ A› by auto
        obtain ?x ∈ e ?z ∈ e using eeq by force
        then have x = ?x
         using esub' eeq yeq zeq by simp
        then have z = ?z
         using esub' eeq yeq by simp
        obtain eq': i' = i (i'+a'+a') mod M = k
         using ‹x = ?x› xeq using ‹z =?z› zeq by auto
        then have diff (i'+a') i' = int a'
         by (simp add: diff-def ‹a' < M›)
        then have a' = a
           by (metis ‹a' ∈ A› add.commute df diff2-invert eq' mod-add-right-eq
nat-int)
        then have y = ?y
         by (metis ‹x = ?x› prod-encode-eq snd-conv yeq xeq)
        then show T = {?x,?y,?z}
         using Teq ‹x = ?x› ‹z = ?z› sets-eq by presburger
      qed
     qed
    qed
    have ∗: card (uedges G) ≤ ε/12 ∗ (card (uverts G))²
      using X ‹X < card (uverts G)› ‹unique-triangles G› ‹uwellformed G› by
blast

    have diff-cancel: ∃j<M. diff j i = int a if a ∈ A for i a
    proof −
     have int a < int M
      using A M-def that by auto
```

then have (*int* ((*i* + *a*) *mod M*) − *int i*) *mod int M* = *int a*
by (*simp add*: *mod-diff-left-eq of-nat-mod*)
then show *?thesis*
using ‹*0* < *M*› *diff-def mod-less-divisor* **by** *blast*
qed
have *diff2-cancel*: ∃*j*<*M*. *diff2 j i* = *int a* **if** *a* ∈ *A i*<*M* **for** *i a*
proof −
have *a*<*M*
using *that M-def A* **by** *auto*
have (*int* ((*i* + *2*∗*a*) *mod M*) − *int i*) ∗ (*1* + *int N*) *mod int M* =
(((*int i* + *2* ∗ *int a*) *mod M*) − *int i*) ∗ (*1* + *int N*) *mod int M*
by (*simp add*: *zmod-int*)
also have ... = *2* ∗ *int a* ∗ (*1* + *int N*) *mod int M*
by (*smt* (*verit*) *mod-diff-left-eq mod-mult-eq mod-mult-right-eq*)
also have ... = *int a mod int M*
proof −
have (*2* ∗ *int a* ∗ (*1* + *int N*) − *int a*) = *M* ∗ *a*
by (*simp add*: *M-def algebra-simps*)
then have *M dvd* (*2* ∗ *int a* ∗ (*1* + *int N*) − *int a*)
by *simp*
then show *?thesis*
using *mod-eq-dvd-iff* **by** *blast*
qed
also have ... = *a* **by** (*simp add*: ‹*a* < *M*›)
finally show *?thesis*
by (*metis* ‹*0* < *M*› *diff2-def mod-less-divisor of-nat-Suc*)
qed

have *card-Edges*: *card* (*Edges* (*part-of* ξ) (*part-of* ζ) *df*) = *M* ∗ *card A* (**is** *card ?E* = -)
if ξ ≠ ζ **and** *df-cancel*: ∀ *a*∈*A*. ∀ *i*<*M*. ∃*j*<*M*. *df j i* = *int a*
and *df-inj*: ∀ *a*. *inj-on* (λ*x*. *df x a*) {..<*M*} **for** ξ ζ *df*
proof −
define *R* **where** *R* ≡ λξ *Y df u p*. ∃ *x y i a*. *u* = {*x*,*y*} ∧ *p* = (*i*,*a*) ∧ *x* = *prod-encode* (ξ,*i*)
∧ *y* ∈ *Y* ∧ *a* ∈ *A* ∧ *df*(*from-part y*) (*from-part x*) = *int a*
have *R-uniq*: ⟦*R* ξ (*part-of* ζ) *df u p*; *R* ξ (*part-of* ζ) *df u p*′; ξ ≠ ζ⟧ ⟹ *p*′ = *p* **for** *u p p*′ ξ ζ *df*
by (*auto simp add*: *R-def doubleton-eq-iff*)
define *f* **where** *f* ≡ λξ *Y df u*. @*p*. *R* ξ *Y df u p*
have *f-if-R*: *f* ξ (*part-of* ζ) *df u* = *p* **if** *R* ξ (*part-of* ζ) *df u p* ξ ≠ ζ **for** *u p* ξ ζ *df*
using *R-uniq f-def that* **by** *blast*
have *bij-betw* (*f* ξ (*part-of* ζ) *df*) *?E* ({..<*M*} × *A*)
unfolding *bij-betw-def inj-on-def*
proof (*intro conjI strip*)
fix *u u*′
assume *u* ∈ *?E* **and** *u*′ ∈ *?E*

and *eq*: $f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ u = f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ u'$
obtain $x\ y\ a$ **where** u: $u = \{x,y\}\ x \in \textit{part-of}\ \xi\ y \in \textit{part-of}\ \zeta\ a \in A$
and *df*: $\textit{df}\ (\textit{from-part}\ y)\ (\textit{from-part}\ x) = \textit{int}\ a$
using ‹$u \in$ *?E*›
by (*force simp add*: *Edges-def image-iff*)
then obtain i **where** i: $x = \textit{prod-encode}\ (\xi,i)$
using *part-of-def* **by** *blast*
with *u df R-def f-if-R that* **have** *fu*: $f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ u = (i,a)$
by *blast*
obtain $x'\ y'\ a'$ **where** u': $u' = \{x',y'\}\ x' \in \textit{part-of}\ \xi\ y' \in \textit{part-of}\ \zeta\ a' \in A$
and *df'*: $\textit{df}\ (\textit{from-part}\ y')\ (\textit{from-part}\ x') = \textit{int}\ a'$
using ‹$u' \in$ *?E*› **by** (*force simp add*: *Edges-def image-iff*)
then obtain i' **where** i': $x' = \textit{prod-encode}\ (\xi,i')$
using *part-of-def* **by** *blast*
with *u' df' R-def f-if-R that* **have** *fu'*: $f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ u' = (i',a')$
by *blast*
have $i'{=}i\ a' = a$
using *fu fu' eq* **by** *auto*
with $i\ i'$ **have** $x = x'$
by *meson*
moreover have $\textit{from-part}\ y = \textit{from-part}\ y'$
using *df df'* ‹$x = x'$› ‹$a' = a$› *df-inj u'*(*3*) *u*(*3*)
by (*clarsimp simp add*: *inj-on-def*) (*metis part-of-M lessThan-iff*)
then have $y = y'$
using *part-of-def u'*(*3*) *u*(*3*) **by** *fastforce*
ultimately show $u = u'$
using *u'*(*1*) *u*(*1*) **by** *force*
next
have $f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ `\ \textit{?E} \subseteq \{..<M\} \times A$
proof (*clarsimp simp*: *Edges-def*)
fix $i\ a\ x\ y\ b$
assume $x \in \textit{part-of}\ \xi\ y \in \textit{part-of}\ \zeta\ \textit{df}\ (\textit{from-part}\ y)\ (\textit{from-part}\ x) = \textit{int}\ b$
$b \in A$ **and** *feq*: $(i,\ a) = f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ \{x,\ y\}$
then have $R\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ \{x,y\}\ (\textit{from-part}\ x,\ b)$
by (*auto simp*: *R-def doubleton-eq-iff part-of-def*)
then have $(\textit{from-part}\ x,\ b) = (i,\ a)$
by (*simp add*: *f-if-R feq from-part-def that*)
then show $i < M \wedge a \in A$
using ‹$x \in \textit{part-of}\ \xi$› ‹$b \in A$› *part-of-M* **by** *fastforce*
qed
moreover have $\{..<M\} \times A \subseteq f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ `\ \textit{?E}$
proof *clarsimp*
fix $i\ a$ **assume** $a \in A$ **and** $i < M$
then obtain j **where** $j{<}M$ **and** j: $\textit{df}\ j\ i = \textit{int}\ a$
using *df-cancel* **by** *metis*
then have *fj*: $f\ \xi\ (\textit{part-of}\ \zeta)\ \textit{df}\ \{\textit{prod-encode}\ (\xi,\ i),\ \textit{prod-encode}\ (\zeta,\ j)\} = (i,a)$
by (*metis R-def* ‹$a \in A$› *enc-iff f-if-R from-enc* ‹$\xi \neq \zeta$›)
then have $\{\textit{prod-encode}\ (\xi,i),\ \textit{prod-encode}\ (\zeta,\ j\ \textit{mod}\ M)\} \in \textit{Edges}\ (\textit{part-of}$

```
ξ) (part-of ζ) df
          apply (clarsimp simp: Edges-def doubleton-eq-iff)
         by (metis ‹a ∈ A› ‹i < M› ‹j < M› enc-iff from-enc image-eqI j mod-if)
        then show (i,a) ∈ f ξ (part-of ζ) df ‘ Edges (part-of ξ) (part-of ζ) df
          using ‹j < M› fj image-iff by fastforce
      qed
      ultimately show f ξ (part-of ζ) df ‘ ?E = {..<M} × A by blast
    qed
    then show ?thesis
      by (simp add: bij-betw-same-card card-cartesian-product)
  qed
  have [simp]: disjnt XY YZ disjnt XY XZ disjnt YZ XZ
    using disjnt-part-of unfolding XY-def YZ-def XZ-def Edges-def disjnt-def
   by (clarsimp simp add: disjoint-iff doubleton-eq-iff, meson disjnt-iff n-not-Suc-n
nat.discI)+
  have [simp]: card XY = M * card A card YZ = M * card A
    by (simp-all add: XY-def YZ-def card-Edges diff-cancel inj-on-diff)
  have [simp]: card XZ = M * card A
    by (simp-all add: XZ-def card-Edges diff2-cancel inj-on-diff2)
  have card-edges: card (uedges G) = 3 * M * card A
    by (simp add: G-def card-Un-disjnt)
  have card A ≤ ε * (real M / 4)
    using * ‹0 < M› by (simp add: cardG card-edges power2-eq-square)
  also have ... < ε * N
    using ‹N>0› by (simp add: M-def assms)
  finally show card A < ε * N .
 qed
qed
```

We finally present the main statement formulated using the upper asymptotic density condition.

```
theorem RothArithmeticProgressions:
 assumes upper-asymptotic-density A > 0
 shows ∃k d. d>0 ∧ progression3 k d ⊆ A
proof (rule ccontr)
 assume non: ∄k d. 0 < d ∧ progression3 k d ⊆ A
 obtain X where X: ∀N ≥ X. ∀A' ⊆ {..<N}. (∄k d. d>0 ∧ progression3 k d
⊆ A')
                      ⟶ card A' < upper-asymptotic-density A / 2 * real N
   by (metis half-gt-zero RothArithmeticProgressions-aux assms)
 then have ∀N ≥ X. card (A ∩ {..<N}) < upper-asymptotic-density A / 2 * N
   by (meson order-trans inf-le1 inf-le2 non)
 then have upper-asymptotic-density A ≤ upper-asymptotic-density A / 2
  by (force simp add: eventually-sequentially less-eq-real-def intro!: upper-asymptotic-densityI)
 with assms show False by linarith
qed

end
```