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

May 26, 2024

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://yufeizhao.com/gtacbook/ and https://yufeizhao.com/gtac/gtac.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 Miscellaneous Preliminaries

lemma *sum-prod-le-prod-sum*:
fixes $a :: 'a \Rightarrow 'b::linordered\text{-}idom$
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 *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-mono1*:
$Y \subseteq Z \Longrightarrow$ *all-edges-between* $Y\ X\ G \subseteq$ *all-edges-between* $Z\ X\ G$
by (*auto simp*: *all-edges-between-def*)

lemma *all-edges-between-mono2*:
$Y \subseteq Z \Longrightarrow$ *all-edges-between* $X\ Y\ G \subseteq$ *all-edges-between* $X\ Z\ G$
by (*auto simp*: *all-edges-between-def*)

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

lemma *uwellformed-alt-snd*:
assumes *uwellformed G* $\{x, y\} \in$ *uedges G*
shows $y \in$ *uverts G*
using *uwellformed-def 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 *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-preserved*: $\llbracket$*all-edges-between A B G'* = *all-edges-between A B G*; $X \subseteq A$; $Y \subseteq B\rrbracket$
$\Longrightarrow$ *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*)

1.2 Preliminaries on Neighbors in Graphs

definition *neighbor-in-graph*:: *uvert* ⇒ *uvert* ⇒ *ugraph* ⇒ *bool*
  where *neighbor-in-graph x y G* ≡ (*x* ∈ (*uverts G*) ∧ *y* ∈ (*uverts G*) ∧ {*x*,*y*} ∈ (*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-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*) = ($\sum$*x*∈*X*. *card* (*neighbors-ss x Y G*))
  using *all-edges-betw-sigma-neighbor assms* **by** (*simp add*: *neighbors-ss-def*)

1.3 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 *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* ≠ *y*
proof (*rule ccontr*)
  assume *a*: ¬ *x* ≠ *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-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 *triangle-in-graph-verts*:
assumes *uwellformed G 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* $\Rightarrow$ *uvert set set*
where *triangle-set G* $\equiv$ { $\{x,y,z\}$ | *x y z*. *triangle-in-graph x y z G*}

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

lemma *finite-triangle-set*:
assumes *fin*: *finite* (*uverts G*) **and** *wf*: *uwellformed G*
shows *finite* (*triangle-set G*)
proof –

have *triangle-set G* $\subseteq$ *Pow* (*uverts G*)
using *PowI 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*:
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-in-graph-ss*:
assumes *uedges Gnew* $\subseteq$ *uedges G*
assumes *triangle-in-graph x y z Gnew*
shows *triangle-in-graph x y z G*
using *assms triangle-in-graph-def* **by** *auto*

lemma *triangle-set-graph-edge-ss*:
assumes *uedges Gnew* $\subseteq$ *uedges G*
assumes *uverts Gnew* = *uverts G*
shows *triangle-set Gnew* $\subseteq$ *triangle-set G*
using *assms* **unfolding** *triangle-set-def* **by** (*blast intro*: *triangle-in-graph-ss*)

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

1.4 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-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** ε*−regular-pair X Y G* **and** *ed*: *edge-density X Y G* $\geq 2*\varepsilon$
defines $X' \equiv \{x \in X.$ *card* (*neighbors-ss x Y G*) < (*edge-density X Y G* − ε) ∗ *card* (*Y*)$\}$
shows *card* $X' < \varepsilon *$ *card X*

(**is** *card* (*?X'*) $<$ ε $*$ -)
proof (*cases ?X'* = {})
case *False* — Following Gowers's proof - more in depth with reasoning on contradiction
let *?rxy* = *1*/(*card X'* $*$ *card Y*)
show *?thesis*
proof (*rule ccontr*)
assume $\neg$ (*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 X'-def* **by** *simp*
have $\bigwedge$ *x*. *x* $\in$ *X'* $\Longrightarrow$ (*card* (*neighbors-ss x Y G*)) $<$ (*edge-density X Y G* $-$ ε) $*$ (*card Y*)
unfolding *X'-def* **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 X'-def*
by (*smt* (*verit*, *del-insts*) *finx of-nat-sum*)
then have *upper*: ($\sum$ *x*$\in$*X'*. *card* (*neighbors-ss x Y G*)) $<$ (*card X'*) $*$ ((*edge-density X Y G* $-$ ε) $*$ (*card Y*))
by (*simp add*: *sum-bounded-above*)
have *yge0*: *card Y* $>$ *0*
by (*metis gr0I mult-eq-0-iff of-nat-0 of-nat-less-0-iff upper*)
have *?rxy* $>$ *0*
using *card-0-eq finx False yge0 X'-def* **by** *fastforce*
then have *upper2*: *?rxy* $*$ ($\sum$ *x*$\in$*X'*. *card* (*neighbors-ss x Y G*)) $<$ *?rxy* $*$ (*card X'*) $*$ ((*edge-density X Y G* $-$ ε) $*$ (*card Y*))
by (*smt* (*verit*) *mult.assoc mult-le-cancel-left upper*)
have *?rxy* $*$ (*card X'*) $*$ ((*edge-density X Y G* $-$ ε) $*$ (*card Y*)) = *edge-density X Y G* $-$ ε
using *False X'-def finx* **by** *force*
with ‹ε $>$ *0*› *upper2* **have** *con*: *edge-density X Y G* $-$ *?rxy* $*$ ($\sum$ *x*$\in$*X'*. *card* (*neighbors-ss x Y G*)) $>$ ε
by *linarith*
have |*edge-density X Y G* $-$ *?rxy* $*$ ($\sum$ *x*$\in$*X'*. *card* (*neighbors-ss x Y G*))|
= |*?rxy* $*$ (*card* (*all-edges-between X' Y G*)) $-$ *edge-density X Y G*|
using *card-all-edges-betw-neighbor fin wf* **by** (*simp add*: *X'-def*)
also have ... = |*edge-density X' Y G* $-$ *edge-density X Y G*|
by (*simp add*: *edge-density-def*)
also have ... $\leq$ ε
using *assms ebound yge0 a* **by** (*force simp add*: *X'-def regular-pair-def*)
finally show *False* **using** *con* **by** *linarith*
qed
qed (*simp add*: ‹*card X* $>$ *0*› *eg0*)

lemma *neighbor-set-meets-e-reg-cond*:

fixes ε::*real*
assumes *edge-density X Y G* $\geq$ *2*$*\varepsilon$
and *card* (*neighbors-ss x Y G*) $\geq$ (*edge-density X Y G* $- \varepsilon$) $*$ *card Y*
shows *card* (*neighbors-ss x Y G*) $\geq \varepsilon *$ *card* (*Y*)
by (*smt* (*verit*) *assms mult-right-mono of-nat-0-le-iff*)

lemma *all-edges-btwn-neighbor-sets-lower-bound*:
fixes ε::*real*
assumes *rp2*: $\varepsilon-$*regular-pair Y Z G*
and *ed1*: *edge-density X Y G* $\geq$ *2*$*\varepsilon$ **and** *ed2*: *edge-density X Z G* $\geq$ *2*$*\varepsilon$
and *cond1*: *card* (*neighbors-ss x Y G*) $\geq$ (*edge-density X Y G* $- \varepsilon$) $*$ *card Y*
and *cond2*: *card* (*neighbors-ss x Z G*) $\geq$ (*edge-density X Z G* $- \varepsilon$) $*$ *card Z*
shows *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*)
$\geq$ (*edge-density Y Z G* $- \varepsilon$) $*$ *card* (*neighbors-ss x Y G*) $*$ *card* (*neighbors-ss x Z G*)
(**is** *card* (*all-edges-between ?Y′ ?Z′ G*) $\geq$ (*edge-density Y Z G* $- \varepsilon$) $*$ *card ?Y′* $*$ *card ?Z′*)
proof $-$
have *yss′*: *?Y′* $\subseteq$ *Y* **using** *neighbors-ss-def* **by** *simp*
have *zss′*: *?Z′* $\subseteq$ *Z* **using** *neighbors-ss-def* **by** *simp*
have *min-sizeY*: *card ?Y′* $\geq \varepsilon *$ *card Y*
using *cond1 ed1 neighbor-set-meets-e-reg-cond* **by** *blast*
have *min-sizeZ*: *card ?Z′* $\geq \varepsilon *$ *card Z*
using *cond2 ed2 neighbor-set-meets-e-reg-cond* **by** *blast*
then have | *edge-density ?Y′ ?Z′ G* $-$ *edge-density Y Z G* | $\leq \varepsilon$
using *min-sizeY yss′ zss′ assms* **by** (*force simp add*: *regular-pair-def*)
then have *edge-density Y Z G* $- \varepsilon \leq$ (*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* $- \varepsilon$) $\leq$ (*card* (*all-edges-between ?Y′ ?Z′ G*))
by (*fastforce simp*: *divide-simps mult.commute simp flip*: *of-nat-mult split*: *if-split-asm*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *mult.assoc mult-of-nat-commute of-nat-mult*)
qed

We are now ready to show the Triangle Counting Lemma:

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*: $\varepsilon-$*regular-pair X Y G* **and** *rp2*: $\varepsilon-$*regular-pair Y Z G* **and** *rp3*: $\varepsilon-$*regular-pair X Z G*
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*$*\varepsilon$) $*$ (*edge-density X Y G* $- \varepsilon$) $*$ (*edge-density X Z G* $- \varepsilon$) $*$

(*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)$\}$
let *?ediff* $= \lambda X\ Y.$ *edge-density* X Y $G - \varepsilon$
define *XF* **where** $XF \equiv \lambda Y. \{x \in X.$ *card*(*neighbors-ss* x Y G) $<$ *?ediff* X Y $*$ *card* $Y\}$
have *fin*: *finite* X *finite* Y *finite* Z **using** *finG rev-finite-subset xss yss zss* **by** *auto*
then have *card* $X > 0$
using *card-0-eq ed1 edge-density-def en0* **by** *fastforce*

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*: $X2 \subseteq X$ **and** *finx2*: *finite* $X2$
by (*auto 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* $(XF\ Y \cup XF\ Z) \leq 2 * \varepsilon *$ (*card* X)
by (*smt* (*verit*) *card-XFY card-XFZ card-Un-le comm-semiring-class.distrib of-nat-add of-nat-mono*)
then have *card* $X2 \geq$ *card* $X - 2 * \varepsilon *$ *card* X
using *part1* **by** (*smt* (*verit*, *del-insts*) *card-Un-le of-nat-add of-nat-mono*)
then have *minx2*: *card* $X2 \geq (1 - 2 * \varepsilon) *$ *card* X
by (*metis mult.commute mult-cancel-left2 right-diff-distrib*)

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

have *edyzgt0*: *?ediff* Y $Z > 0$ **and** *edxygt0*: *?ediff* X $Y > 0$
using *ed1 ed3* ‹$\varepsilon > 0$› **by** *linarith*+
have *card-y-bound*: *card* (*neighbors-ss* x Y G) $\geq$ *?ediff* X Y $*$ *card* Y
and *card-z-bound*: *card* (*neighbors-ss* x Z G) $\geq$ *?ediff* X Z $*$ *card* Z
if $x \in X2$ **for** x
using *that* **by** (*auto simp*: *XF-def X2-def*)
have *card-y-bound'*:
($\sum x \in X2.$ *?ediff* Y Z $*$ (*card* (*neighbors-ss* x Y G)) $*$ (*card* (*neighbors-ss* x Z G))) $\geq$
($\sum x \in X2.$ *?ediff* Y Z $*$ *?ediff* X Y $*$ (*card* Y) $*$ (*card* (*neighbors-ss* x Z G)))

by (*rule sum-mono*) (*smt* (*verit*, *best*) *mult.left-commute card-y-bound edyzgt0 mult.commute mult-right-mono of-nat-0-le-iff*)
have *card-z-bound'*:
($\sum x \in X2$. *?ediff Y Z* $*$ *?ediff X Y* $*$ (*card Y*) $*$ (*card* (*neighbors-ss x Z G*))) $\geq$
($\sum x \in X2$. *?ediff Y Z* $*$ *?ediff X Y* $*$ (*card Y*) $*$ *?ediff X Z* $*$ (*card Z*))
using *card-z-bound mult-left-mono edxygt0 edyzgt0* **by** (*fastforce intro*!: *sum-mono*)
have *eq-set*: $\bigwedge x$. $\{(y,z)$. $y \in Y \wedge z \in Z \wedge \{y, z\} \in$ *uedges G* $\wedge$ *neighbor-in-graph x y G* $\wedge$ *neighbor-in-graph x z G* $\} =$
$\{(y,z)$. $y \in$ (*neighbors-ss x Y G*) $\wedge z \in$ (*neighbors-ss x Z G*) $\wedge \{y, z\} \in$ *uedges G* $\}$
by (*auto simp*: *neighbors-ss-def*)
have *card ?T-all* = ($\sum x \in X$. *card* $\{(y,z) \in Y \times Z$. *triangle-in-graph x y z G*$\}$)
using *triple-sigma-rewrite-card fin* **by** *force*
also have ... = ($\sum x \in X$. *card* $\{(y,z)$. $y \in Y \wedge z \in Z \wedge \{y, z\} \in$ *uedges G* $\wedge$ *neighbor-in-graph x y G* $\wedge$ *neighbor-in-graph x z G* $\}$)
using *triangle-in-graph-edge-point assms* **by** *auto*
also have ... = ($\sum x \in X$. *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*))
using *all-edges-between-def eq-set* **by** *presburger*
finally have *l*: *card ?T-all* $\geq$ ($\sum x \in X2$. *card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*))
by (*simp add*: *fin xss sum-mono2*)
have ($\sum x \in X2$. *?ediff Y Z* $*$ (*card* (*neighbors-ss x Y G*)) $*$ (*card* (*neighbors-ss x Z G*))) $\leq$
($\sum x \in X2$. *real* (*card* (*all-edges-between* (*neighbors-ss x Y G*) (*neighbors-ss x Z G*) *G*)))
(**is** *sum ?F* - $\leq$ *sum ?G* -)
proof (*rule sum-mono*)
show $\bigwedge x$. $x \in X2 \Longrightarrow$ *?F x* $\leq$ *?G x*
using *all-edges-btwn-neighbor-sets-lower-bound card-y-bound card-z-bound ed1 ed2 rp2* **by** *blast*
qed
then have *card ?T-all* $\geq$ *card X2* $*$ *?ediff Y Z* $*$ *?ediff X Y* $*$ *card Y* $*$ *?ediff X Z* $*$ *card Z*
using *card-z-bound' card-y-bound' l of-nat-le-iff* [*symmetric*, **where** *'a=real*] **by** *force*
then have *real* (*card ?T-all*) $\geq$ (($1 - 2 * \varepsilon$) $*$ *card X*) $*$ *?ediff Y Z* $*$ *?ediff X Y* $*$ (*card Y*) $*$ *?ediff X Z* $*$ (*card Z*)
by (*smt* (*verit*, *best*) *ed2 edxygt0 edyzgt0 en0 minx2 mult-right-mono of-nat-0-le-iff*)
then show *?thesis* **by** (*simp add*: *triangle-triples-def mult.commute mult.left-commute*)

qed

definition *regular-graph* :: *real* $\Rightarrow$ *uvert set set* $\Rightarrow$ *ugraph* $\Rightarrow$ *bool*
(*-–regular'-graph* [*999*]*1000*)
where ε*–regular-graph P G* $\equiv \forall R\ S$. $R \in P \longrightarrow S \in P \longrightarrow \varepsilon$*–regular-pair R S G*
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* = {} ∨ (*card X* ≥ η ∧ *card Y* ≥ η) **for** η::*real*

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

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.

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*) *set-simps permutations-of-setI set-empty*)
qed
have *?TT* ⊆ {(*x*,*y*,*z*). (*triangle-in-graph x y z G*)}
by *auto*
also have ... ⊆ (⋃ *t* ∈ *triangle-set G*. *tofl* ' *permutations-of-set t*)
using *edge-vertices-not-equal* [*OF wf*] *in-tofl*
by (*clarsimp simp add*: *triangle-set-def triangle-in-graph-def*) *metis*
finally have *?TT* ⊆ (⋃ *t* ∈ *triangle-set G*. *tofl* ' *permutations-of-set t*) **.**
then have *card ?TT* ≤ *card*(⋃ *t* ∈ *triangle-set G*. *tofl* ' *permutations-of-set t*)
by (*intro card-mono finite-UN-I finite-triangle-set*) (*auto simp*: *assms*)
also have ... ≤ (∑ *t* ∈ *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*)
by (*rule sum-mono*) (*metis card.infinite card-permutations-of-set card-triangle-3 eq-refl local.wf nat.case 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 *X* $\subseteq$ *uverts G* **and** *Y* $\subseteq$ *uverts G* **and** *Z* $\subseteq$ *uverts G* **and** *fin*: *finite* (*uverts G*)
and *wf*: *uwellformed G*
assumes *card* {(*x*,*y*,*z*) $\in$ *X* $\times$ *Y* $\times$ *Z* . (*triangle-in-graph x y z G*)} $\geq$ *t*
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 *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*) $\leq$ δ $*$ *card*(*uverts G*) ^ *3* $\longrightarrow$
($\exists$ G'. *triangle-free-graph* G' $\wedge$ *uverts* G' = *uverts G* $\wedge$ *uedges* G' $\subseteq$ *uedges G* $\wedge$
card (*uedges G* − *uedges* G') $\leq$ ε $*$ (*card* (*uverts G*))^2)
(**is** $\exists\,\delta$::*real* > *0*. $\forall$ *G*. - $\longrightarrow$ - $\longrightarrow$ - $\longrightarrow$ ($\exists$ *Gnew*. *?Φ G Gnew*))
proof (*cases* ε < *1*)

case *False*
show *?thesis*
proof (*intro exI conjI strip*)
fix *G*
define *Gnew* **where** *Gnew* $\equiv$ ((*uverts G*), {}::*uedge set*)
assume *G*: *uwellformed G card*(*uverts G*) > 0
then show *triangle-free-graph Gnew uverts Gnew* = *uverts G uedges Gnew* $\subseteq$ *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*)) $\leq \varepsilon *$ ((*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*)) $\leq \delta *$ *card* (*uverts G*) $\hat{~}$ *3*

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

let *?n* = *card* (*uverts G*)
have *vne*: *uverts G* $\neq$ {}
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* ($\varepsilon/4$) *G P* **and** *card P* $\leq$ *M0*
by *blast*
define *M* **where** *M* $\equiv$ *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 $?e4M = \varepsilon\ /\ (4 * real\ M)$
define *D* **where** $D \equiv 1/6 *(1-(\varepsilon/2)) * ((\varepsilon/4)\hat{~}3) * ?e4M\hat{~}3$
have $D > 0$
using ‹$0 < \varepsilon$› ‹$\varepsilon < 1$› ‹$M > 0$› **by** (*simp add*: *D-def zero-less-mult-iff*)
have $D0 \le D$
unfolding *D0-def D-def* **using** ‹$0 < \varepsilon$› ‹$\varepsilon < 1$› ‹$card\ P \le M0$› ‹$M > 0$›
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** $R{\in}P$ **for** *R*
using *fin finite-graph-partition-finite finite-graph-partition-gt0 that* **by** *auto*
have *card-P-le*: $card\ R \le\ ?n$ **if** $R{\in}P$ **for** *R*
by (*meson card-mono fin fin-part finite-graph-partition-subset that*)
have *P-disjnt*: $\bigwedge R\ S.\ [\![R \neq S;\ R \in P;\ S \in P]\!] \Longrightarrow R \cap S = \{\}$
using *fin-part*
by (*metis disjnt-def finite-graph-partition-def insert-absorb pairwise-insert partition-on-def*)
have *sum-card-P*: $(\sum R{\in}P.\ card\ R) =\ ?n$
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 ($\le (\epsilon/4M)n$) 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.

define *edge* **where** $edge \equiv \lambda R\ S.\ mk\text{-}uedge$ ‘ (*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*) $\le$ *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*) $\le$ *card R* $*$ *card S* **if** $R{\in}P\ S{\in}P$ **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** $irreg\text{-}pairs \equiv \{(R,S).\ R \in P \wedge S \in P \wedge \neg\ (\varepsilon/4){-}regular\text{-}pair\ R\ S\ G\}$
define *Ea* **where** $Ea \equiv (\bigcup (R,S) \in irreg\text{-}pairs.\ edge\ R\ S)$

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*ˆ*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* ∧ ¬ (ε/*4*)−*regular-pair R S G*}. *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*ˆ*2*) — Count the edge bound for *Eb*.
proof −
have §: ⋀*R S*. ⟦*R* ∈ *P*; *S* ∈ *P*; ¬ *edge-dense R S G* (ε / *2*)⟧
⟹ *real* (*card* (*edge R S*)) ∗ *2* ≤ ε ∗ *real* (*card R*) ∗ *real* (*card S*)
by (*simp add*: *divide-simps edge-dense-def edge-density-def card-P-gt0*)
(*smt* (*verit*, *best*) *card-edge-le-card of-nat-le-iff mult.assoc*)
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 ... $\leq$ ($\sum$ (*R*,*S*)$\in$*low-density-pairs*. $\varepsilon/2$ $*$ *card R* $*$ *card S*)
unfolding *low-density-pairs-def* **by** (*force intro*: *sum-mono* §)
also have ... $\leq$ ($\sum$ (*R*,*S*)$\in$*P* $\times$ *P*. $\varepsilon/2$ $*$ *card R* $*$ *card S*)
using *subs* ‹ε > *0*› **by** (*intro sum-mono2*) (*auto simp*: ‹*finite P*›)
also have ... = $\varepsilon/2$ $*$ ($\sum$ (*R*,*S*)$\in$*P* $\times$ *P*. *card R* $*$ *card S*)
by (*simp add*: *sum-distrib-left case-prod-unfold mult-ac*)
also have ... $\leq$ ($\varepsilon/2$) $*$ (*?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* $\leq$ ($\varepsilon/4$) $*$ (*?n^2*) — Count the edge bound for *Ec*.
proof −
have *edge-bound*: (*card* (*edge R S*)) $\leq$ *?e4M* $*$ *?n^2*
if *S* $\in$ *P small R* **for** *R S*
proof −
have *real* (*card R*) $\leq$ ε $*$ *?n* / (*4* $*$ *real M*)
using *that* **by** (*simp add*: *small-def*)
with *card-P-le* [*OF* ‹*S*$\in$*P*›]
have $*$: *real* (*card R*) $*$ *real* (*card S*) $\leq$ 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*∈*P*›)
qed
also have ... ≤ *?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*) ≤ *?e4M* ∗ *real* (*?n*^{2}) **.**
qed
also have ... ≤ *card ?SMALL* ∗ (*?e4M* ∗ *?n*^*2*)
by *simp*
also have ... ≤ *M* ∗ (*?e4M* ∗ *?n*^*2*)
using *egt* **by** (*intro mult-right-mono*) (*auto simp add*: *card-sp*)
also have ... ≤ (*ε*/*4*) ∗ (*?n*^*2*)
using ‹*M* > *0*› **by** *simp*
finally show *?thesis* **.**
qed
— total count
have *prev1*: *card* (*Ea* ∪ *Eb* ∪ *Ec*) ≤ *card* (*Ea* ∪ *Eb*) + *card Ec* **by** (*simp add*: *card-Un-le*)
also have ... ≤ *card Ea* + *card Eb* + *card Ec* **by** (*simp add*: *card-Un-le*)
also have *prev*: ... ≤ (*ε*/*4*)∗(*?n*^*2*) + (*ε*/*2*)∗(*?n*^*2*) + (*ε*/*4*)∗(*?n*^*2*)
using *eabound ebbound ecbound* **by** *linarith*
finally have *cutedgesbound*: *card* (*Ea* ∪ *Eb* ∪ *Ec*) ≤ *ε* ∗ (*?n*^*2*) **by** *simp*

define *Gnew* **where** *Gnew* ≡ (*uverts G*, *uedges G* − (*Ea* ∪ *Eb* ∪ *Ec*))
show ∃ *Gnew*. *?Φ G Gnew*
proof (*intro exI conjI*)
show *verts*: *uverts Gnew* = *uverts G* **by** (*simp add*: *Gnew-def*)
have *diffedges*: (*Ea* ∪ *Eb* ∪ *Ec*) ⊆ *uedges G*
by (*auto simp*: *Ea-def Eb-def Ec-def all-edges-between-def edge-def*)
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 ir-
reg-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: (ε/4)−regular-pair R S Gnew 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: (ε/4)−regular-graph P Gnew
  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 S T* **where** *xinp*: $x \in R$ **and** *ilt*: $R \in P$ **and** *yinp*: $y \in S$ **and** *jlt*: $S \in P$
and *zinp*: $z \in T$ **and** *klt*: $T \in P$
by (*metis graph-partition-new xin Union-iff finite-graph-partition-equals*)
then have *finitesubsets*: *finite R finite S finite T*
using *new-fin fin-part finite-graph-partition-finite fin* **by** *auto*
have *subsets*: $R \subseteq$ *uverts Gnew* $S \subseteq$ *uverts Gnew* $T \subseteq$ *uverts Gnew*
using *finite-graph-partition-subset ilt jlt klt graph-partition-new* **by** *auto*
have *min-sizes*: *card* $R \geq$ *?e4M*∗*?n* *card* $S \geq$ *?e4M*∗*?n* *card* $T \geq$ *?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* $\geq \varepsilon/2$ *edge-density R T Gnew* $\geq \varepsilon/2$ *edge-density S T Gnew* $\geq \varepsilon/2$
using *density-bound ilt jlt klt xinp yinp zinp trig-ex*
by (*auto simp*: *dense-graph-def edge-dense-def all-edges-between-def triangle-in-graph-def*)
then have *min-dens-diff*:
edge-density R S Gnew $- \varepsilon/4 \geq \varepsilon/4$ *edge-density R T Gnew* $- \varepsilon/4 \geq \varepsilon/4$ *edge-density S T Gnew* $- \varepsilon/4 \geq \varepsilon/4$
by *auto*
have *mincard0*: (*card R*) ∗ (*card S*) ∗ (*card T*) ≥ 0 **by** *simp*
have *gtcube*: ((*edge-density R S Gnew*) $- \varepsilon/4$)∗((*edge-density R T Gnew*) $- \varepsilon/4$) ∗((*edge-density S T Gnew*) $- \varepsilon/4$) $\geq (\varepsilon/4)$^*3*
using *min-dens-diff e4gt real-mult-gt-cube* **by** *auto*
then have *c1*: ((*edge-density R S Gnew*) $- \varepsilon/4$)∗((*edge-density R T Gnew*) $- \varepsilon/4$) ∗((*edge-density S T Gnew*) $- \varepsilon/4$) ≥ 0
by (*smt* (*verit*) *e4gt zero-less-power*)
have *?e4M* ∗ *?n* ≥ 0
using *egt* **by** *force*
then have *card R* ∗ *card S* ∗ *card T* $\geq$ (*?e4M* ∗ *?n*)^*3*
by (*metis min-sizes of-nat-mult real-mult-gt-cube*)
then have *cardgtbound*:*card R* ∗ *card S* ∗ *card T* $\geq$ *?e4M*^ *3* ∗ *?n*^*3*
by (*metis of-nat-power power-mult-distrib*)

have ($1-\varepsilon/2$) ∗ $(\varepsilon/4)$^*3* ∗ $(\varepsilon/(4*M))$^*3* ∗ *?n*^*3* $\leq$ ($1-\varepsilon/2$) ∗ $(\varepsilon/4)$^*3* ∗ *card R* ∗ *card S* ∗ *card T*
using *cardgtbound ordered-comm-semiring-class.comm-mult-left-mono True e4gt* **by** *fastforce*
also have ... $\leq$ ($1-2*(\varepsilon/4)$) ∗ (*edge-density R S Gnew* $- \varepsilon/4$) ∗ (*edge-density R T Gnew* $- \varepsilon/4$)
∗ (*edge-density S T Gnew* $- \varepsilon/4$) ∗ *card R* ∗ *card S* ∗ *card T*
using *gtcube c1* ‹$\varepsilon < 1$› *mincard0* **by** (*simp add*: *mult.commute mult.left-commute mult-left-mono*)
also have ... $\leq$ *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*) $\geq$ *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) $\geq$ 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* $G \neq \{\}$› *card-gt-0-iff fin*)
with δ ‹*D0* $\leq$ 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*)) $\leq$ ε $*$ *real* ((*card* (*uverts* G))^2)
using *cardbound edges verts* **by** *blast*
qed
qed (*rule* ‹*0* $<$ δ›)
qed

1.5 Roth's Theorem

We will first need the following corollary of the Triangle Removal Lemma.

See https://en.wikipedia.org/wiki/Ruzsa--Szemerédi_problem. Suggested by Yaël Dillies

corollary *Diamond-free*:
fixes ε :: *real*
assumes *0* $<$ ε
shows $\exists$ *N*>*0*. $\forall$ *G*. *card*(*uverts* G) $>$ N $\longrightarrow$ *uwellformed* G $\longrightarrow$ *unique-triangles*
G $\longrightarrow$
card (*uedges* G) $\leq$ ε $*$ (*card* (*uverts* G))^2
proof –
have $\varepsilon/3$ $>$ *0*
using *assms* **by** *auto*
then obtain δ::*real* **where** δ $>$ *0*
and δ: $\bigwedge G$. ⟦*card*(*uverts* G) $>$ *0*; *uwellformed* G; *card* (*triangle-set* G) $\leq$ δ $*$
card(*uverts* G) ^ *3*⟧
$\Longrightarrow$ $\exists$ G'. *triangle-free-graph* G' $\wedge$ *uverts* G' = *uverts* G $\wedge$ (*uedges* G'
$\subseteq$ *uedges* G) $\wedge$
card (*uedges* G – *uedges* G') $\leq$ $\varepsilon/3$ $*$ (*card* (*uverts* G))^2
using *triangle-removal-lemma* **by** *metis*
obtain *N*::*nat* **where** N: *real* N $\geq$ *1* / (*3*$*$$\delta$)
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 $G\text{-}gt\text{-}N$: $N < ?n$
and wf: $uwellformed\ G$
and $uniq$: $unique\text{-}triangles\ G$
have $G\text{-}ne$: $?n > 0$
using $G\text{-}gt\text{-}N$ **by** $linarith$
let $?TWO = (\lambda t.\ [t]^2)$
have $tri\text{-}nsets\text{-}2$: $[\{x,y,z\}]^2 = \{\{x,y\},\{y,z\},\{x,z\}\}$ **if** $triangle\text{-}in\text{-}graph\ x\ y\ z\ G$
for $x\ y\ z$
using $that$ **unfolding** $nsets\text{-}def\ triangle\text{-}in\text{-}graph\text{-}def\ card\text{-}2\text{-}iff\ doubleton\text{-}eq\text{-}iff$
by $(blast\ dest!$: $edge\text{-}vertices\text{-}not\text{-}equal\ [OF\ wf])$
have $tri\text{-}nsets\text{-}3$: $\{\{x,y\},\{y,z\},\{x,z\}\} \in [uedges\ G]^3$ **if** $triangle\text{-}in\text{-}graph\ x\ y\ z\ G$
for $x\ y\ z$
using $that$ **by** $(simp\ add$: $nsets\text{-}def\ card\text{-}3\text{-}iff\ triangle\text{-}in\text{-}graph\text{-}def)$
$(metis\ doubleton\text{-}eq\text{-}iff\ edge\text{-}vertices\text{-}not\text{-}equal\ [OF\ wf])$
have sub: $?TWO\ `\ triangle\text{-}set\ G \subseteq [uedges\ G]^3$
using $tri\text{-}nsets\text{-}2\ tri\text{-}nsets\text{-}3\ triangle\text{-}set\text{-}def$ **by** $auto$
have $\bigwedge i.\ i \in triangle\text{-}set\ G \Longrightarrow ?TWO\ i \neq \{\}$
using $tri\text{-}nsets\text{-}2\ triangle\text{-}set\text{-}def$ **by** $auto$
moreover have $dfam$: $disjoint\text{-}family\text{-}on\ ?TWO\ (triangle\text{-}set\ G)$
using $sub\ [unfolded\ image\text{-}subset\text{-}iff]\ uniq$
unfolding $disjoint\text{-}family\text{-}on\text{-}def\ triangle\text{-}set\text{-}def\ nsets\text{-}def\ unique\text{-}triangles\text{-}def$
by $(smt\ (verit)\ disjoint\text{-}iff\text{-}not\text{-}equal\ insert\text{-}subset\ mem\text{-}Collect\text{-}eq\ mk\text{-}disjoint\text{-}insert$
$)$
ultimately have inj: $inj\text{-}on\ ?TWO\ (triangle\text{-}set\ G)$
by $(simp\ add$: $disjoint\text{-}family\text{-}on\text{-}iff\text{-}disjoint\text{-}image)$
have §: $\exists\ T{\in}triangle\text{-}set\ G.\ e \in [T]^2$ **if** $e \in uedges\ G$ **for** e
using $uniq\ [unfolded\ unique\text{-}triangles\text{-}def]\ that\ local.wf$
apply $(simp\ add$: $triangle\text{-}set\text{-}def\ triangle\text{-}in\text{-}graph\text{-}def\ nsets\text{-}def\ uwellformed\text{-}def)$
by $(metis\ (mono\text{-}tags,\ lifting)\ finite.emptyI\ finite.insertI\ finite\text{-}subset)$
with sub **have** $\bigcup(?TWO\ `\ triangle\text{-}set\ G) = uedges\ G$
by $(auto\ simp$: $image\text{-}subset\text{-}iff\ nsets\text{-}def)$
then have $card\ (\bigcup(?TWO\ `\ triangle\text{-}set\ G)) = card\ (uedges\ G)$
by $simp$
moreover have $card\ (\bigcup(?TWO\ `\ triangle\text{-}set\ G)) = 3 * card\ (triangle\text{-}set\ G)$
proof $(subst\ card\text{-}UN\text{-}disjoint'\ [OF\ dfam])$
show $finite\ ([i]^2)$ **if** $i \in triangle\text{-}set\ G$ **for** i
using $that\ tri\text{-}nsets\text{-}2\ triangle\text{-}set\text{-}def$ **by** $fastforce$
show $finite\ (triangle\text{-}set\ G)$
by $(meson\ G\text{-}ne\ card\text{-}gt\text{-}0\text{-}iff\ local.wf\ finite\text{-}triangle\text{-}set)$
have $card\ ([i]^2) = 3$ **if** $i \in triangle\text{-}set\ G$ **for** i
using $that\ wf\ tri\text{-}nsets\text{-}2\ tri\text{-}nsets\text{-}3$ **by** $(force\ simp\ add$: $nsets\text{-}def\ triangle\text{-}set\text{-}def)$
then show $(\sum i{\in}triangle\text{-}set\ G.\ card\ ([i]^2)) = 3 * card\ (triangle\text{-}set\ G)$
by $simp$
qed
ultimately have 3: $3 * card\ (triangle\text{-}set\ G) = card\ (uedges\ G)$
by $auto$

have *card* (*uedges G*) ≤ *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 ... = *?n choose 2*
by (*metis G-ne card-all-edges card-eq-0-iff not-less0*)
also have ... ≤ $?n^2$
by (*metis binomial-eq-0-iff binomial-le-pow linorder-not-le zero-le*)
finally have *card* (*uedges G*) ≤ $?n^2$ **.**
with *3* **have** *card* (*triangle-set G*) ≤ $?n^2$ / *3* **by** *linarith*
also have ... ≤ δ ∗ *?n* ^ *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.

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*
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*) $\leq \varepsilon * ?n^2$
using *3 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* $\Rightarrow$ *'a* $\Rightarrow$ *'a set*
where *progression3 k d* $\equiv$ $\{k,\ k+d,\ k+d+d\}$

lemma *p3-int-iff*: *progression3* (*int k*) (*int d*) $\subseteq$ *int* ' *A* $\longleftrightarrow$ *progression3 k d* $\subseteq$ *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 $\varepsilon > 0$
obtains *M* **where** $\forall N \geq M.\ \forall A \subseteq \{..<N\}.\ (\nexists k\ d.\ d>0 \wedge$ *progression3 k d* $\subseteq$ *A*) $\longrightarrow$ *card A* $< \varepsilon *$ *real N*
proof −
obtain *L* **where** $L>0$
and *L*: $\bigwedge G.$ ⟦*card*(*uverts G*) $> L$; *uwellformed G*; *unique-triangles G*⟧
$\Longrightarrow$ *card* (*uedges G*) $\leq \varepsilon/12 *$ (*card* (*uverts G*))^2
by (*metis assms Diamond-free less-divide-eq-numeral1*(*1*) *mult-eq-0-iff*)
show *thesis*
proof (*intro strip that*)
fix *N A*
assume $L \leq N$ **and** *A*: $A \subseteq \{..<N\}$
and *non*: $\nexists k\ d.\ 0 < d \wedge$ *progression3 k d* $\subseteq$ *A*
then have $N > 0$ **using** ‹$0 < L$› **by** *linarith*
define *M* **where** $M \equiv$ *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-2-eq-Suc coprime-Suc-right-nat coprime-mult-right-iff mult-Suc-right*)
then have *cop*: *coprime M* (*1* + *int N*)
by (*metis coprime-int-iff of-nat-Suc*)
have *A-sub-M*: *int* ' $A \subseteq \{..<M\}$
using *A* **by** (*force simp*: *M-def*)
have *non-img-A*: $\nexists k\ d.\ d > 0 \wedge$ *progression3 k d* $\subseteq$ *int* ' *A*
by (*metis imageE insert-subset non p3-int-iff pos-int-cases 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*
by (*smt* (*verit*) *diff-def 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*) > *L*
using *M-def* ‹*L* ≤ *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* $p \neq$ *label-of-part* q **if** $\{p,q\} \in XY$ **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*]: $\{p\} \notin XY$ **for** p
by (*metis insert-absorb2*)
have *label-ne-YZ* [*simp*]: *label-of-part* $p \neq$ *label-of-part* q **if** $\{p,q\} \in YZ$ **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-of* ξ) (*part-of* ζ) df;
$\{q,r\} \in$ *Edges* (*part-of* ξ) (*part-of* ζ) df;
$\{p,r\} \in$ *Edges* (*part-of* γ) (*part-of* β) 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-encode*$(0,\ i)$
$\wedge\ y =$ *prod-encode*$(1,\ (i+d)\ mod\ M)$
$\wedge\ z =$ *prod-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-encode*$(0,i)$ **and** j: $y =$ *prod-encode*$(1,j)$

and k: $z =$ *prod-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 ∈ A* **and** *a3*: (*int k − int j*) *mod int M* = *int a3*
using *yz* ‹*x ∈ ?X*› *j k* **by** (*auto simp add*: *YZ-def Edges-def doubleton-eq-iff diff-def*)
obtain *a2* **where** *a2 ∈ A* **and** *a2*: (*int k − int i*) *mod int M* = *int* (*a2 ∗ 2*) *mod int M*
using *xz* ‹*x ∈ ?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 ∈ 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 ∈ A› df ‹j<M› by blast
      with ‹a ∈ A› ‹j<M› have {?y,?z} ∈ YZ
        by (fastforce simp: YZ-def Edges-def image-iff diff-invert)
      moreover have {?x,?z} ∈ XZ
       using ‹a ∈ 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 ∈ 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 ?y ∈ 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*: $e = \{prod\text{-}encode(1,j),\ prod\text{-}encode(2,k)\}$
and $j{<}M\ k{<}M$
and *df*: *diff k j* = *int a* **and** $a \in A$
by (*auto simp*: *YZ-def Edges-def part-of-def numeral-2-eq-2*)
let $?x = prod\text{-}encode\ (0,\ (M{+}j{-}a)\ mod\ M)$
let $?y = prod\text{-}encode\ (1,\ j)$
let $?z = prod\text{-}encode\ (2,\ k)$
have *zeq*: $k = (j{+}a)\ mod\ M$
using *diff-invert* **using** ‹$a \in A$› *df* ‹$k{<}M$› **by** *blast*
with ‹$a \in A$› ‹$k{<}M$› **have** $\{?x,?z\} \in XZ$
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* ‹$a \in A$› **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 $\{?x,?y\} \in XY$
using ‹$a \in A$› ‹$j{<}M$› **unfolding** *XY-def Edges-def*
by (*force simp add*: *zeq image-iff diff-invert doubleton-eq-iff ex-disj-distrib*)
ultimately have *T*: *triangle-in-graph ?x ?y ?z G*
using ‹$e \in uedges\ G$› **by** (*auto simp*: *G-def eeq triangle-in-graph-def*)
show *?thesis*
proof (*intro ex1I*)
show $\exists\, x\ y\ z.\ \{?x,?y,?z\} = \{x,\ y,\ z\} \wedge triangle\text{-}in\text{-}graph\ x\ y\ z\ G \wedge e \subseteq \{?x,?y,?z\}$
using *T eeq* **by** *blast*
fix *T*
assume $\exists\, p\ q\ r.\ T = \{p,\ q,\ r\} \wedge triangle\text{-}in\text{-}graph\ p\ q\ r\ G \wedge e \subseteq T$
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*: $x = \textit{prod-encode}(0,\ i')$
and *yeq*: $y = \textit{prod-encode}(1,\ (i'{+}a')\ \textit{mod}\ M)$
and *zeq*: $z = \textit{prod-encode}(2,\ (i'{+}a'{+}a')\ \textit{mod}\ M)$
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 $?y \in e\ ?z \in e$
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'*: $(i'{+}a')\ \textit{mod}\ M = j\ (i'{+}a'{+}a')\ \textit{mod}\ M = k$
using ‹$y = ?y$› *yeq* **using** ‹$z = ?z$› *zeq* **by** *auto*
then have $\textit{diff}\ (i'{+}a'{+}a')\ (i'{+}a') = \textit{int}\ a'$
by (*simp add*: *diff-def* ‹$a' < M$›)
then have $a' = a$
by (*metis M-mod-bound* ‹$a' \in A$› *df diff-invert eq' mod-add-eq mod-if of-nat-eq-iff*)
have $(M + ((i'{+}a')\ \textit{mod}\ M) - a')\ \textit{mod}\ M = (M + (i' + a') - a')\ \textit{mod}\ M$
by (*metis Nat.add-diff-assoc2* ‹$a' < M$› *less-imp-le-nat mod-add-right-eq*)
with ‹$i' < M$› **have** $(M + ((i'{+}a')\ \textit{mod}\ M) - a')\ \textit{mod}\ M = i'$
by *force*
with ‹$a' = a$› *eq'* **have** $(M + j - a)\ \textit{mod}\ M = i'$
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*: $e = \{\textit{prod-encode}(0,i),\ \textit{prod-encode}(2,k)\}$
and $i{<}M$ **and** $k{<}M$
and *df*: $\textit{diff2}\ k\ i = \textit{int}\ a$ **and** $a \in A$
by (*auto simp*: *XZ-def Edges-def part-of-def eval-nat-numeral*)
let $?x = \textit{prod-encode}\ (0,\ i)$
let $?y = \textit{prod-encode}\ (1,\ (i{+}a)\ \textit{mod}\ M)$
let $?z = \textit{prod-encode}\ (2,\ k)$
have *keq*: $k = (i{+}a{+}a)\ \textit{mod}\ M$
using *diff2-invert* [*OF* ‹$a \in A$›, *of i*] *df* ‹$k{<}M$› **using** *inj-on-diff2* [*of i*]
by (*simp add*: *inj-on-def Ball-def mod-add-left-eq*)
with ‹$a \in A$› **have** $\{?x,?y\} \in XY$

using ‹$a \in A$› ‹$i<M$› ‹$k<M$› **apply** (*auto simp*: *XY-def Edges-def*)
by (*metis M-mod-bound diff-invert enc-iff from-enc imageI*)
moreover have $\{?y,?z\} \in YZ$
apply (*auto simp*: *YZ-def Edges-def image-iff eval-nat-numeral*)
by (*metis M-mod-bound* ‹$a \in A$› *diff-invert enc-iff from-enc mod-add-left-eq keq*)
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 $?x \in e\ ?z \in e$ **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* ‹$a' \in A$› *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*) $\leq \varepsilon/12 * ($*card* (*uverts G*)$)^2$
using *L* ‹$L <$ *card* (*uverts G*)› ‹*unique-triangles G*› ‹*uwellformed G*› **by** *blast*

have *diff-cancel*: $\exists j<M.\ diff\ j\ i = int\ a$ **if** $a \in A$ **for** $i\ a$
using *M-mod-bound diff-invert that* **by** *blast*
have *diff2-cancel*: $\exists j<M.\ diff2\ j\ i = int\ a$ **if** $a \in A$ **for** $i\ a$
using *M-mod-bound diff2-invert that* **by** *blast*

have *card-Edges*: $card\ (Edges\ (part\text{-}of\ \xi)\ (part\text{-}of\ \zeta)\ df) = M * card\ A$ (**is** $card\ ?E = \text{-}$)
if $\xi \neq \zeta$ **and** *df-cancel*: $\forall a\in A.\ \forall i<M.\ \exists j<M.\ df\ j\ i = int\ a$
and *df-inj*: $\forall a.\ inj\text{-}on\ (\lambda x.\ df\ x\ a)\ \{..<M\}$ **for** $\xi\ \zeta\ df$
proof −
define R **where** $R \equiv \lambda\xi\ Y\ df\ u\ p.\ \exists x\ y\ i\ a.\ u = \{x,y\} \wedge p = (i,a) \wedge x = prod\text{-}encode\ (\xi,i)$
$\wedge\ y \in Y \wedge a \in A \wedge df(from\text{-}part\ y)\ (from\text{-}part\ x) = int\ a$
have *R-uniq*: $[\![R\ \xi\ (part\text{-}of\ \zeta)\ df\ u\ p;\ R\ \xi\ (part\text{-}of\ \zeta)\ df\ u\ p';\ \xi \neq \zeta]\!] \Longrightarrow p' = p$ **for** $u\ p\ p'\ \xi\ \zeta\ df$
by (*auto simp add*: *R-def doubleton-eq-iff*)
define f **where** $f \equiv \lambda\xi\ Y\ df\ u.\ @p.\ R\ \xi\ Y\ df\ u\ p$
have *f-if-R*: $f\ \xi\ (part\text{-}of\ \zeta)\ df\ u = p$ **if** $R\ \xi\ (part\text{-}of\ \zeta)\ df\ u\ p\ \xi \neq \zeta$ **for** $u\ p\ \xi\ \zeta\ df$
using *R-uniq f-def that* **by** *blast*
have $bij\text{-}betw\ (f\ \xi\ (part\text{-}of\ \zeta)\ df)\ ?E\ (\{..<M\} \times A)$
unfolding *bij-betw-def inj-on-def*
proof (*intro conjI strip*)
fix $u\ u'$
assume $u \in ?E$ **and** $u' \in ?E$
and *eq*: $f\ \xi\ (part\text{-}of\ \zeta)\ df\ u = f\ \xi\ (part\text{-}of\ \zeta)\ df\ u'$
obtain $x\ y\ a$ **where** *u*: $u = \{x,y\}\ x \in part\text{-}of\ \xi\ y \in part\text{-}of\ \zeta\ a \in A$
and *df*: $df\ (from\text{-}part\ y)\ (from\text{-}part\ x) = int\ a$
using ‹$u \in ?E$› **by** (*force simp add*: *Edges-def image-iff*)
then obtain i **where** *i*: $x = prod\text{-}encode\ (\xi,i)$
using *part-of-def* **by** *blast*
with *u df R-def f-if-R that* **have** *fu*: $f\ \xi\ (part\text{-}of\ \zeta)\ df\ u = (i,a)$
by *blast*
obtain $x'\ y'\ a'$ **where** *u′*: $u' = \{x',y'\}\ x' \in part\text{-}of\ \xi\ y' \in part\text{-}of\ \zeta\ a'\in A$
and *df′*: $df\ (from\text{-}part\ y')\ (from\text{-}part\ x') = int\ a'$
using ‹$u' \in ?E$› **by** (*force simp add*: *Edges-def image-iff*)
then obtain i' **where** *i′*: $x' = prod\text{-}encode\ (\xi,i')$
using *part-of-def* **by** *blast*
with *u′ df′ R-def f-if-R that* **have** *fu′*: $f\ \xi\ (part\text{-}of\ \zeta)\ df\ u' = (i',a')$
by *blast*
have $i'=i\ a' = a$
using *fu fu′ eq* **by** *auto*
with $i\ i'$ **have** $x = x'$
by *meson*
moreover have $from\text{-}part\ y = from\text{-}part\ y'$
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*)

ultimately show $u = u'$
using $u\ u'$ **by** (*metis enc-iff from-part-def prod.collapse prod-decode-inverse*)
next
have $f\ \xi$ (*part-of* ζ) *df* ‘ *?E* $\subseteq \{..<M\} \times A$
proof (*clarsimp simp*: *Edges-def*)
fix $i\ a\ x\ y\ b$
assume $x \in$ *part-of* ξ $y \in$ *part-of* ζ *df* (*from-part* y) (*from-part* x) = *int* b
$b \in A$ **and** *feq*: $(i,\ a) = f\ \xi$ (*part-of* ζ) *df* $\{x,\ y\}$
then have $R\ \xi$ (*part-of* ζ) *df* $\{x,y\}$ (*from-part* x, b)
by (*auto simp*: *R-def doubleton-eq-iff part-of-def*)
then have (*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 ‹$x \in$ *part-of* ξ› ‹$b \in A$› *part-of-M* **by** *fastforce*
qed
moreover have $\{..<M\} \times A \subseteq f\ \xi$ (*part-of* ζ) *df* ‘ *?E*
proof *clarsimp*
fix $i\ a$ **assume** $a \in A$ **and** $i < M$
then obtain j **where** $j<M$ **and** j: *df* $j\ i$ = *int* a
using *df-cancel* **by** *metis*
then have *fj*: $f\ \xi$ (*part-of* ζ) *df* $\{$*prod-encode* $(\xi,\ i)$, *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 $\{$*prod-encode* (ξ,i), *prod-encode* $(\zeta,\ j$ *mod* $M)\} \in$ *Edges* (*part-of*
ξ) (*part-of* ζ) *df*
apply (*clarsimp simp*: *Edges-def doubleton-eq-iff*)
by (*metis* ‹$a \in A$› ‹$i < M$› ‹$j < M$› *enc-iff from-enc image-eqI j mod-if*)
then show $(i,a) \in f\ \xi$ (*part-of* ζ) *df* ‘ *Edges* (*part-of* ξ) (*part-of* ζ) *df*
using ‹$j < M$› *fj image-iff* **by** *fastforce*
qed
ultimately show $f\ \xi$ (*part-of* ζ) *df* ‘ *?E* $= \{..<M\} \times A$ **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* (*uedges G*) $= 3 * M *$ *card A*
by (*simp add*: *G-def card-Un-disjnt*)
then have *card A* $\leq \varepsilon *$ (*real M* / *4*)
using $*$ ‹$0 < M$› **by** (*simp add*: *cardG power2-eq-square*)
also have $\ldots < \varepsilon * N$
using ‹$N>0$› **by** (*simp add*: *M-def assms*)

finally show *card* $A < \varepsilon * N$ **.**
qed
qed

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

theorem *RothArithmeticProgressions*:
assumes *upper-asymptotic-density* $A > 0$
shows $\exists k\ d.\ d{>}0 \wedge$ *progression3* $k\ d \subseteq A$
proof (*rule ccontr*)
assume *non*: $\nexists k\ d.\ 0 < d \wedge$ *progression3* $k\ d \subseteq A$
obtain M **where** X: $\forall N \geq M.\ \forall A' \subseteq \{..{<}N\}.\ (\nexists k\ d.\ d{>}0 \wedge$ *progression3* $k\ d$
$\subseteq A')$
$\longrightarrow$ *card* $A' <$ *upper-asymptotic-density* $A\ /\ 2 *$ *real* N
by (*metis half-gt-zero RothArithmeticProgressions-aux assms*)
then have $\forall N \geq M.$ *card* $(A \cap \{..{<}N\}) <$ *upper-asymptotic-density* $A\ /\ 2 * N$
by (*meson order-trans inf-le1 inf-le2 non*)
then have *upper-asymptotic-density* $A \leq$ *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