Ramsey Number Bounds

Lawrence C. Paulson

17 March 2025

Abstract

Ramsey's theorem [1] implies that for any given natural numbers k and l, there exists some $R(k, l)$ such that a graph having at least $R(k, l)$ vertices must have either a clique of cardinality k or an anticlique (independent set) of cardinality l. Equivalently, for a *complete* graph of size $R(k, l)$, every red/blue colouring of the edges must yield an entirely red k-clique or an entirely blue l-clique. Although $R(k, l)$ is for practical purposes impossible to calculate from k and l, some upper and lower bounds are known. The celebrated probabilistic argument by Paul Erdős is formalised here, with various of its consequences.

Contents

1 Lower bounds for Ramsey numbers **3**
- 1.1 Preliminaries . . . 3
- 1.2 Relating cliques to graphs; Ramsey numbers . . . 4
- 1.3 Elementary properties of Ramsey numbers . . . 9
- 1.4 The product lower bound . . . 11
- 1.5 A variety of upper bounds, including a stronger Erdős–Szekeres 13
- 1.6 Probabilistic lower bounds: the main theorem and applications 15

Acknowledgements Many thanks to Andrew Thomason and Chelsea Edmonds for their help with the probabilistic proofs, and to Bhavik Mehta for making his large Ramsey development available online. The author was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council.

1 Lower bounds for Ramsey numbers

Probabilistic proofs of lower bounds for Ramsey numbers. Variations and strengthenings of the classical Erdős–Szekeres upper bound, which is proved in the original Ramsey theory Also a number of simple properties of Ramsey numbers, including the equivalence of the clique/anticlique and edge colouring definitions.

theory *Ramsey-Bounds*
imports
HOL−Library.Ramsey
HOL−Library.Infinite-Typeclass
HOL−Probability.Probability
Undirected-Graph-Theory.Undirected-Graph-Basics
begin

1.1 Preliminaries

Elementary facts involving binomial coefficients

lemma *choose-two-real*: *of-nat* (*n choose 2*) = *real n* ∗ (*real n* − *1*) / *2*
proof (*cases even n*)
case *True*
then show *?thesis*
by (*auto simp*: *choose-two dvd-def*)
next
case *False*
then have *even* (*n−1*)
by *simp*
then show *?thesis*
by (*auto simp*: *choose-two dvd-def*)
qed

lemma *add-choose-le-power*: (*n* + *k*) *choose n* ≤ *Suc k* ^ *n*
proof −
have ∗: (∏ *i*<*n*. *of-nat* (*n*+*k* − *i*) / *of-nat* (*n* − *i*)) ≤ (∏ *i*<*n*. *real* (*Suc k*))
proof (*intro prod-mono conjI*)
fix *i*
assume *i*: *i* ∈ {..<*n*}
then have *real* (*n* + *k* − *i*) / *real* (*n* − *i*) = *1* + *k*/*real*(*n*−*i*)
by (*auto simp*: *divide-simps*)
also have ... ≤ *1* + *real k*
using *i* **by** (*simp add*: *divide-inverse inverse-le-1-iff mult-left-le*)
finally show *real* (*n* + *k* − *i*) / *real* (*n* − *i*) ≤ *real* (*Suc k*)
by *simp*
qed *auto*
then have *real*((*n* + *k*) *choose n*) ≤ *real* (*Suc k* ^ *n*)
by (*simp add*: *binomial-altdef-of-nat lessThan-atLeast0*)
then show *?thesis*
by *linarith*

qed

lemma *choose-le-power*: *m choose k* $\leq$ (*Suc m* $-$ *k*) ^ *k*
by (*metis Suc-diff-le add-choose-le-power add-diff-inverse-nat binomial-eq-0-iff less-le-not-le nle-le zero-le*)

lemma *sum-nsets-one*: ($\sum$ *U*$\in$[*V*]$^{Suc\ 0}$. *f U*) = ($\sum$ *x*$\in$*V*. *f* {*x*})
proof $-$
have *bij*: *bij-betw* (λx. {*x*}) *V* ([*V*]$^{Suc\ 0}$)
by (*auto simp*: *inj-on-def bij-betw-def nsets-one*)
show *?thesis*
using *sum.reindex-bij-betw* [*OF bij*] **by** (*metis* (*no-types*, *lifting*) *sum.cong*)
qed

1.2 Relating cliques to graphs; Ramsey numbers

When talking about Ramsey numbers, sometimes cliques are best, sometimes colour maps

lemma *nsets2-eq-all-edges*: [*A*]^2 = *all-edges A*
using *card-2-iff'* **unfolding** *nsets-def all-edges-def*
by *fastforce*

lemma *indep-eq-clique-compl*: *indep R E* = *clique R* (*all-edges R* $-$ *E*)
by (*auto simp*: *indep-def clique-def all-edges-def*)

lemma *all-edges-subset-iff-clique*: *all-edges K* $\subseteq$ *E* $\longleftrightarrow$ *clique K E*
by (*fastforce simp*: *card-2-iff clique-def all-edges-def*)

definition *clique-indep* $\equiv$ λ*m n K E*. *card K* = *m* $\wedge$ *clique K E* $\vee$ *card K* = *n* $\wedge$ *indep K E*

lemma *clique-all-edges-iff*: *clique K* (*E* $\cap$ *all-edges K*) $\longleftrightarrow$ *clique K E*
by (*simp add*: *clique-def all-edges-def*)

lemma *indep-all-edges-iff*: *indep K* (*E* $\cap$ *all-edges K*) $\longleftrightarrow$ *indep K E*
by (*simp add*: *indep-def all-edges-def*)

lemma *clique-indep-all-edges-iff*: *clique-indep s t K* (*E* $\cap$ *all-edges K*) = *clique-indep s t K E*
by (*simp add*: *clique-all-edges-iff clique-indep-def indep-all-edges-iff*)

identifying Ramsey numbers (possibly not the minimum) for a given type and pair of integers

definition *is-clique-RN* **where**
is-clique-RN $\equiv$ λ*U*::'*a itself*. λ*m n r*.
($\forall$ *V*::'*a set*. $\forall$ *E*. *finite V* $\longrightarrow$ *card V* $\geq$ *r* $\longrightarrow$ ($\exists$ *K*$\subseteq$*V*. *clique-indep m n K E*))

could be generalised to allow e.g. any hereditarily finite set

```
abbreviation is-Ramsey-number :: [nat,nat,nat] ⇒ bool where
  is-Ramsey-number m n r ≡ partn-lst {..<r} [m,n] 2

lemma is-clique-RN-imp-partn-lst:
  fixes U :: 'a itself
  assumes r: is-clique-RN U m n r and inf: infinite (UNIV::'a set)
  shows partn-lst {..<r} [m,n] 2
  unfolding partn-lst-def
proof (intro strip)
  fix f
  assume f: f ∈ [{..<r}]^2 → {..<length [m,n]}
  obtain V::'a set where finite V and V: card V = r
    by (metis inf infinite-arbitrarily-large)
  then obtain φ where φ: bij-betw φ V {..<r}
    using to-nat-on-finite by blast
  have φ-iff: φ v = φ w ⟷ v=w if v∈V w∈V for v w
    by (metis φ bij-betw-inv-into-left that)
  define E where E ≡ {e. ∃x∈V. ∃y∈V. e = {x,y} ∧ x ≠ y ∧ f {φ x, φ y} =
0}
  obtain K where K⊆V and K: clique-indep m n K E
    by (metis r V ‹finite V› is-clique-RN-def nle-le)
  then consider (0) card K = m clique K E | (1) card K = n indep K E
    by (meson clique-indep-def)
  then have ∃i<2. monochromatic {..<r} ([m, n] ! i) 2 f i
  proof cases
    case 0
    have f e = 0
      if e: e ⊆ φ ` K finite e card e = 2 for e :: nat set
    proof −
      obtain x y where x∈V y∈V e = {φ x, φ y} ∧ x ≠ y
        using e ‹K⊆V› φ by (fastforce simp: card-2-iff)
      then show ?thesis
        using e 0
        apply (simp add: φ-iff clique-def E-def doubleton-eq-iff image-iff)
        by (metis φ-iff insert-commute)
    qed
    moreover have φ ` K ∈ [{..<r}]^m
      unfolding nsets-def
    proof (intro conjI CollectI)
      show φ ` K ⊆ {..<r}
        by (metis ‹K ⊆ V› φ bij-betw-def image-mono)
      show finite (φ ` K)
        using ‹φ ` K ⊆ {..<r}› finite-nat-iff-bounded by auto
      show card (φ ` K) = m
        by (metis 0(1) ‹K ⊆ V› φ bij-betw-same-card bij-betw-subset)
    qed
    ultimately show ?thesis
      apply (simp add: image-subset-iff monochromatic-def)
      by (metis (mono-tags, lifting) mem-Collect-eq nsets-def nth-Cons-0 pos2)
```

```
  next
    case 1
   have f e = Suc 0
      if e: e ⊆ φ ‘ K finite e card e = 2 for e :: nat set
    proof −
      obtain x y where x∈V y∈V e = {φ x, φ y} ∧ x ≠ y
        using e ‹K⊆V› φ by (fastforce simp: card-2-iff)
      then show ?thesis
        using e 1 f bij-betw-imp-surj-on [OF φ]
       apply (simp add: indep-def E-def card-2-iff Pi-iff doubleton-eq-iff image-iff)
         by (metis ‹K ⊆ V› doubleton-in-nsets-2 imageI in-mono less-2-cases-iff
less-irrefl numeral-2-eq-2)
    qed
    then have f ‘ [φ ‘ K]^2 ⊆ {Suc 0}
      by (simp add: image-subset-iff nsets-def)
    moreover have φ ‘ K ∈ [{..<r}]^n
      unfolding nsets-def
    proof (intro conjI CollectI)
      show φ ‘ K ⊆ {..<r}
        by (metis ‹K ⊆ V› φ bij-betw-def image-mono)
      show finite (φ ‘ K)
        using ‹φ ‘ K ⊆ {..<r}› finite-nat-iff-bounded by auto
      show card (φ ‘ K) = n
        by (metis 1(1) ‹K ⊆ V› φ bij-betw-same-card bij-betw-subset)
    qed
    ultimately show ?thesis
      by (metis less-2-cases-iff monochromatic-def nth-Cons-0 nth-Cons-Suc)
  qed
  then show ∃i<length [m,n]. monochromatic {..<r} ([m, n] ! i) 2 f i
    by (simp add: numeral-2-eq-2)
qed

lemma partn-lst-imp-is-clique-RN:
  fixes U :: 'a itself
  assumes partn-lst {..<r} [m,n] 2
  shows is-clique-RN U m n r
  unfolding is-clique-RN-def
proof (intro strip)
  fix V::'a set and E ::'a set set
  assume V: finite V r ≤ card V
  obtain φ where φ: bij-betw φ {..<card V} V
    using ‹finite V› bij-betw-from-nat-into-finite by blast
  define f :: nat set ⇒ nat where f ≡ λe. if φ‘e ∈ E then 0 else 1
  have f: f ∈ nsets {..<r} 2 → {..<2}
    by (simp add: f-def)
  obtain i H where i<2 and H: H ⊆ {..<r} finite H card H = [m,n] ! i
    and mono: f ‘ (nsets H 2) ⊆ {i}
    using partn-lstE [OF assms f]
   by (metis (mono-tags, lifting) length-Cons list.size(3) mem-Collect-eq nsets-def
```

```
numeral-2-eq-2)
  have [simp]: ⋀v w. ⟦v ∈ H; w ∈ H⟧ ⟹ φ v = φ w ⟷ v=w
    using bij-betw-imp-inj-on [OF φ] H
    by (meson V(2) inj-on-def inj-on-subset lessThan-subset-iff)
  define K where K ≡ φ ` H
  have [simp]: ⋀v w. ⟦v ∈ K; w ∈ K⟧ ⟹ inv-into {..<card V} φ v = inv-into
{..<card V} φ w ⟷ v=w
    using bij-betw-inv-into-right [OF φ] H V φ
    by (metis K-def image-mono inv-into-injective lessThan-subset-iff subset-iff)
  have K ⊆ V
    using H φ V bij-betw-imp-surj-on by (fastforce simp: K-def nsets-def)
  have [simp]: card (φ ` H) = card H
   using H by (metis V(2) φ bij-betw-same-card bij-betw-subset lessThan-subset-iff)
  consider (0) i=0 | (1) i=1
    using ‹i<2› by linarith
  then have clique-indep m n K E
  proof cases
    case 0
    have {v, w} ∈ E if v ∈ K and w ∈ K and v ≠ w  for v w
    proof −
      have ∗: {inv-into {..<card V} φ v, inv-into {..<card V} φ w} ∈ [H]^2
        using that bij-betw-inv-into-left [OF φ] H(1) V(2)
        by (auto simp: nsets-def card-insert-if K-def)
      show ?thesis
        using 0 ‹K ⊆ V› mono bij-betw-inv-into-right[OF φ] that
        apply (simp add: f-def image-subset-iff)
        by (metis ∗ image-empty image-insert subsetD)
    qed
    then show ?thesis
      unfolding clique-indep-def clique-def
      by (simp add: 0 H(3) K-def)
  next
    case 1
    have {v, w} ∉ E if v ∈ K and w ∈ K and v ≠ w  for v w
    proof −
      have ∗: {inv-into {..<card V} φ v, inv-into {..<card V} φ w} ∈ [H]^2
        using that bij-betw-inv-into-left [OF φ] H(1) V(2)
        by (auto simp: nsets-def card-insert-if K-def)
      show ?thesis
        using 1 ‹K ⊆ V› mono bij-betw-inv-into-right[OF φ] that
        apply (simp add: f-def image-subset-iff)
        by (metis ∗ image-empty image-insert subsetD)
    qed
    then show ?thesis
      unfolding clique-indep-def indep-def
      by (simp add: 1 H(3) K-def)
  qed
  with ‹K ⊆ V› show ∃K. K ⊆ V ∧ clique-indep m n K E by blast
qed
```

All complete graphs of a given cardinality are the same

lemma *is-clique-RN-any-type*:
assumes *is-clique-RN* (*U*::'*a itself*) *m n r infinite* (*UNIV*::'*a set*)
shows *is-clique-RN* (*V*::'*b*::*infinite itself*) *m n r*
by (*metis partn-lst-imp-is-clique-RN is-clique-RN-imp-partn-lst assms*)

lemma *is-Ramsey-number-le*:
assumes *is-Ramsey-number m n r* **and** *le*: $m' \leq m$ $n' \leq n$
shows *is-Ramsey-number m' n' r*
using *partn-lst-less* [**where** $\alpha = [m,n]$ **and** $\alpha' = [m',n']$] *assms*
by (*force simp*: *less-Suc-eq*)

definition *RN* **where**
RN $\equiv \lambda m\ n.$ *LEAST r. is-Ramsey-number m n r*

lemma *is-Ramsey-number-RN*: *partn-lst* {..< (*RN m n*)} [*m*,*n*] *2*
by (*metis LeastI-ex RN-def ramsey2-full*)

lemma *RN-le*: $[\![$*is-Ramsey-number m n r*$]\!] \Longrightarrow$ *RN m n* $\leq r$
by (*simp add*: *Least-le RN-def*)

lemma *RN-le-ES*: *RN i j* $\leq$ *ES 2 i j*
by (*simp add*: *RN-le ramsey2-full*)

lemma *RN-mono*:
assumes $m' \leq m$ $n' \leq n$
shows *RN m' n'* $\leq$ *RN m n*
by (*meson RN-le assms is-Ramsey-number-RN is-Ramsey-number-le*)

lemma *indep-iff-clique* [*simp*]: $K \subseteq V \Longrightarrow$ *indep K* (*all-edges V* $-$ *E*) $\longleftrightarrow$ *clique K E*
by (*auto simp*: *clique-def indep-def all-edges-def*)

lemma *clique-iff-indep* [*simp*]: $K \subseteq V \Longrightarrow$ *clique K* (*all-edges V* $-$ *E*) $\longleftrightarrow$ *indep K E*
by (*auto simp*: *clique-def indep-def all-edges-def*)

lemma *is-Ramsey-number-commute-aux*:
assumes *is-Ramsey-number m n r*
shows *is-Ramsey-number n m r*
unfolding *partn-lst-def*
proof (*intro strip*)
fix *f*
assume *f*: $f \in [\{..<r\}]^2 \to \{..<$*length* $[n, m]\}$
define *f'* **where** $f' \equiv \lambda A.\ 1 - f\ A$
then have $f' \in [\{..<r\}]^2 \to \{..<2\}$
by (*auto simp*: *f'-def*)
then obtain *i H* **where** $i<2$ **and** *H*: $H \in [\{..<r\}]^{([m,n]\ !\ i)}$ f' ` $[H]^2 \subseteq \{i\}$
using *assms* **by** (*auto simp*: *partn-lst-def monochromatic-def numeral-2-eq-2*)

then have $H \subseteq \{..<r\}$
by (*auto simp*: *nsets-def*)
then have *fless2*: $\forall x{\in}[H]^2.\ f\ x < Suc\ (Suc\ 0)$
using *funcset-mem* $[OF\ f]$ *nsets-mono* **by** *force*
show $\exists i{<}length\ [n,\ m].\ monochromatic\ \{..<r\}\ ([n,m]\ !\ i)\ 2\ f\ i$
unfolding *monochromatic-def*
proof (*intro exI bexI conjI*)
show $f\ `\ [H]^2 \subseteq \{1{-}i\}$
using *H fless2* **by** (*fastforce simp*: *f'-def*)
show $H \in [\{..<r\}]^{([n,\ m]\ !\ (1-i))}$
using ‹$i{<}2$› *H* **by** (*fastforce simp*: *less-2-cases-iff f'-def image-subset-iff*)
qed *auto*
qed

1.3 Elementary properties of Ramsey numbers

lemma *is-Ramsey-number-commute*: *is-Ramsey-number m n r* $\longleftrightarrow$ *is-Ramsey-number n m r*
by (*meson is-Ramsey-number-commute-aux*)

lemma *RN-commute-aux*: *RN n m* $\leq$ *RN m n*
using *RN-le is-Ramsey-number-RN is-Ramsey-number-commute* **by** *blast*

lemma *RN-commute*: *RN m n* = *RN n m*
by (*simp add*: *RN-commute-aux le-antisym*)

lemma *RN-le-choose*: *RN k l* $\leq$ (*k+l choose k*)
by (*metis ES2-choose ramsey2-full RN-le*)

lemma *RN-le-choose'*: *RN k l* $\leq$ (*k+l choose l*)
by (*metis RN-commute RN-le-choose add.commute*)

lemma *RN-0* [*simp*]: *RN 0 m* = *0*
unfolding *RN-def*
proof (*intro Least-equality*)
show *is-Ramsey-number 0 m 0*
by (*auto simp*: *partn-lst-def monochromatic-def nsets-def*)
qed *auto*

lemma *RN-1* [*simp*]:
assumes *m>0* **shows** *RN* (*Suc 0*) *m* = *Suc 0*
unfolding *RN-def*
proof (*intro Least-equality*)
have [*simp*]: $[\{..<Suc\ 0\}]^2 = \{\}\ [\{\}]^2 = \{\}$
by (*auto simp*: *nsets-def card-2-iff*)
show *is-Ramsey-number* (*Suc 0*) *m* (*Suc 0*)
by (*auto simp*: *partn-lst-def monochromatic-def*)
fix *i*

```
  assume i: is-Ramsey-number (Suc 0) m i
  show i ≥ Suc 0
  proof (cases i=0)
    case True
    with i assms show ?thesis
      by (auto simp: partn-lst-def monochromatic-def nsets-empty-iff less-Suc-eq)
  qed auto
qed

lemma RN-0' [simp]: RN m 0 = 0 and RN-1' [simp]: m>0 ⟹ RN m (Suc 0)
= Suc 0
  using RN-1 RN-commute by auto

lemma is-clique-RN-2: is-clique-RN TYPE(nat) 2 m m
  unfolding is-clique-RN-def
proof (intro strip)
  fix V :: 'a set and E
  assume finite V
    and m ≤ card V
  show ∃K. K ⊆ V ∧ clique-indep 2 m K E
  proof (cases ∃K. K ⊆ V ∧ card K = 2 ∧ clique K E)
    case False
    then have indep V E
      apply (clarsimp simp: clique-def indep-def card-2-iff)
      by (smt (verit, best) doubleton-eq-iff insert-absorb insert-iff subset-iff)
    then show ?thesis
      unfolding clique-indep-def
      by (meson ‹m ≤ card V› card-Ex-subset smaller-indep)
  qed (metis clique-indep-def)
qed

lemma RN-2 [simp]:
  shows RN 2 m = m
proof (cases  m>1)
  case True
  show ?thesis
    unfolding RN-def
  proof (intro Least-equality)
    show is-Ramsey-number 2 m m
      using is-clique-RN-imp-partn-lst is-clique-RN-2 by blast
    fix i
    assume is-Ramsey-number 2 m i
    then have i: is-clique-RN TYPE(nat) 2 m i
      using partn-lst-imp-is-clique-RN by blast
    obtain V :: nat set where V: card V = i finite V
      by force
    show i ≥ m
    proof (cases i<m)
      case True
```

```
    then have ¬ (∃K⊆V. card K = 2 ∧ clique K {})
      by (auto simp: clique-def card-2-iff')
    with i V True show ?thesis
    unfolding is-clique-RN-def clique-indep-def by (metis card-mono dual-order.refl)
   qed auto
  qed
next
  case False
  then show ?thesis
   by (metis RN-0' RN-1' Suc-1 less-2-cases-iff not-less-eq)
qed

lemma RN-2' [simp]:
  shows RN m 2 = m
  using RN-2 RN-commute by force

lemma RN-3plus:
  assumes k ≥ 3
  shows RN k m ≥ m
proof −
  have RN 2 m = m
   using assms by auto
  with RN-mono[of 2 k m m] assms show ?thesis
   by force
qed

lemma RN-3plus':
  assumes k ≥ 3
  shows RN m k ≥ m
  using RN-3plus RN-commute assms by presburger

lemma clique-iff: F ⊆ all-edges K ⟹ clique K F ⟷ F = all-edges K
  by (auto simp: clique-def all-edges-def card-2-iff)

lemma indep-iff: F ⊆ all-edges K ⟹ indep K F ⟷ F = {}
  by (auto simp: indep-def all-edges-def card-2-iff)

lemma all-edges-empty-iff: all-edges K = {} ⟷ (∃v. K ⊆ {v})
   using clique-iff [OF empty-subsetI] by (metis clique-def empty-iff singleton-iff
subset-iff)

lemma Ramsey-number-zero: ¬ is-Ramsey-number (Suc m) (Suc n) 0
  by (metis RN-1 RN-le is-Ramsey-number-le not-one-le-zero Suc-le-eq One-nat-def
zero-less-Suc)
```

1.4 The product lower bound

```
lemma Ramsey-number-times-lower: ¬ is-clique-RN (TYPE(nat*nat)) (Suc m)
(Suc n) (m*n)
```

proof
define *edges* **where** *edges* ≡ {{(*x*,*y*),(*x*′,*y*)}| *x x*′ *y*. *x*<*m* ∧ *x*′<*m* ∧ *y*<*n*}
assume *is-clique-RN* (*TYPE*(*nat*∗*nat*)) (*Suc m*) (*Suc n*) (*m*∗*n*)
then obtain *K* **where** *K*: *K* ⊆ {..<*m*} × {..<*n*} **and** *clique-indep* (*Suc m*) (*Suc n*) *K edges*
unfolding *is-clique-RN-def*
by (*metis card-cartesian-product card-lessThan finite-cartesian-product finite-lessThan le-refl*)
then consider *card K* = *Suc m* ∧ *clique K edges* | *card K* = *Suc n* ∧ *indep K edges*
by (*meson clique-indep-def*)
then show *False*
proof *cases*
case *1*
then have *inj-on fst K fst* ‘ *K* ⊆ {..<*m*}
using *K* **by** (*auto simp*: *inj-on-def clique-def edges-def doubleton-eq-iff*)
then have *card K* ≤ *m*
by (*metis card-image card-lessThan card-mono finite-lessThan*)
then show *False*
by (*simp add*: *1*)
next
case *2*
then have *snd-eq*: *snd u* ≠ *snd v* **if** *u* ∈ *K v* ∈ *K u* ≠ *v* **for** *u v*
using *that K* **unfolding** *edges-def indep-def*
by (*smt* (*verit*, *best*) *lessThan-iff mem-Collect-eq mem-Sigma-iff prod.exhaust-sel subsetD*)
then have *inj-on snd K*
by (*meson inj-onI*)
moreover have *snd* ‘ *K* ⊆ {..<*n*}
using *comp-sgraph.wellformed K* **by** *auto*
ultimately show *False*
by (*metis 2 Suc-n-not-le-n card-inj-on-le card-lessThan finite-lessThan*)
qed
qed

theorem *RN-times-lower*:
shows *RN* (*Suc m*) (*Suc n*) > *m*∗*n*
by (*metis partn-lst-imp-is-clique-RN Ramsey-number-times-lower is-Ramsey-number-RN*

partn-lst-greater-resource linorder-le-less-linear)

corollary *RN-times-lower*′:
shows ⟦*m*>*0*; *n*>*0*⟧ ⟹ *RN m n* > (*m*−*1*)∗(*n*−*1*)
using *RN-times-lower gr0-conv-Suc* **by** *force*

lemma *RN-eq-0-iff*: *RN m n* = *0* ⟷ *m*=*0* ∨ *n*=*0*
by (*metis RN-0 RN-0*′ *RN-times-lower*′ *gr0I not-less-zero*)

lemma *RN-gt1*:

assumes $2 \le k$ $3 \le l$ **shows** $k < RN\ k\ l$
using *RN-times-lower'* [*of k l*] *RN-3plus'*[*of l k*] *assms*
apply (*simp add*: *eval-nat-numeral*)
by (*metis Suc-le-eq Suc-pred leD n-less-n-mult-m nat-less-le zero-less-diff*)

lemma *RN-gt2*:
assumes $2 \le k$ $3 \le l$ **shows** $k < RN\ l\ k$
by (*simp add*: *RN-commute assms RN-gt1*)

1.5 A variety of upper bounds, including a stronger Erdős–Szekeres

lemma *RN-1-le*: $RN\ (Suc\ 0)\ l \le Suc\ 0$
by (*metis RN-0' RN-1 gr-zeroI le-cases less-imp-le*)

lemma *is-Ramsey-number-add*:
assumes *i>1 j>1*
and *n1*: *is-Ramsey-number* $(i - 1)\ j\ n1$
and *n2*: *is-Ramsey-number i* $(j - 1)\ n2$
shows *is-Ramsey-number i j* (*n1+n2*)
proof −
have *partn-lst* $\{..<Suc\ (n1 + n2 - 1)\}$ $[i, j]$ $(Suc\ (Suc\ 0))$
using *ramsey-induction-step* [*of n1 i j 1 n2 n1+n2−1*] *ramsey1-explicit assms*
by (*simp add*: *numeral-2-eq-2*)
moreover have *n1>0*
using *assms*
by (*metis Ramsey-number-zero Suc-pred' gr0I not-less-iff-gr-or-eq zero-less-diff*)
ultimately show *?thesis*
by (*metis One-nat-def Suc-1 Suc-pred' add-gr-0*)
qed

lemma *RN-le-add-RN-RN*:
assumes *i>1 j>1*
shows $RN\ i\ j \le RN\ (i - Suc\ 0)\ j + RN\ i\ (j - Suc\ 0)$
using *is-Ramsey-number-add RN-le assms is-Ramsey-number-RN*
by *simp*

Cribbed from Bhavik Mehta

lemma *RN-le-choose-strong*: $RN\ k\ l \le (k + l - 2)\ choose\ (k - 1)$
proof (*induction n* ≡ *k+l arbitrary*: *k l*)
case *0*
then show *?case*
by *simp*
next
case (*Suc n*)
have ∗: $RN\ k\ l \le k + l - 2\ choose\ (k - 1)$ **if** $k \le Suc\ 0$
using *that* **by** (*metis One-nat-def RN-1-le RN-le-choose Suc-pred binomial-n-0 neq0-conv diff-is-0-eq'*)
show *?case*

```
  proof (cases k ≤ Suc 0 ∨ l ≤ Suc 0)
    case True
    with ∗ show ?thesis
      using le-Suc-eq by fastforce
  next
    case False
    then have 2: k > 1 l > 1
      by auto
    have RN (k − Suc 0) l ≤ k − Suc 0 + l − 2 choose (k − Suc 0 − Suc 0)
      by (metis False Nat.add-diff-assoc2 One-nat-def Suc diff-Suc-1 nat-le-linear)
    moreover
    have RN k (l − Suc 0) ≤ k − Suc 0 + l − 2 choose (k − Suc 0)
      by (metis False Nat.diff-add-assoc2 Suc diff-Suc-1 nat-le-linear One-nat-def
diff-add-assoc)
    ultimately
    show ?thesis
    using RN-le-add-RN-RN [OF 2] 2 by (simp add: choose-reduce-nat eval-nat-numeral)
  qed
qed

lemma RN-le-power2: RN i j ≤ 2 ^ (i+j−2)
  by (meson RN-le-choose-strong binomial-le-pow2 le-trans)

lemma RN-le-power4: RN i i ≤ 4 ^ (i−1)
proof −
  have (i + i − 2) = 2 ∗ (i−1)
    by simp
  then show ?thesis
    using RN-le-power2 [of i i] by (simp add: power-mult)
qed
```

Bhavik Mehta again

```
lemma RN-le-argpower: RN i j ≤ j ^ (i−1)
proof (cases i=0 ∨ j=0)
  case True
  then show ?thesis
    by auto
next
  case False
  then show ?thesis
    using RN-le-choose-strong [of i j] add-choose-le-power[of i−1 j−1]
    by (simp add: numeral-2-eq-2)
qed

lemma RN-le-argpower′: RN j i ≤ j ^ (i−1)
  using RN-commute RN-le-argpower by presburger
```

1.6 Probabilistic lower bounds: the main theorem and applications

General probabilistic setup, omitting the actual probability calculation. Andrew Thomason's proof (private communication)

theorem *Ramsey-number-lower-gen*:
fixes *n k*::*nat* **and** *p*::*real*
assumes *n*: (*n choose k*) * *p* ^ (*k choose 2*) + (*n choose l*) * (*1* − *p*) ^ (*l choose 2*) < *1*
assumes *p01*: *0*<*p p*<*1*
shows ¬ *is-Ramsey-number k l n*
proof
assume *con*: *is-Ramsey-number k l n*
define *W* **where** *W* ≡ {..<*n*}
have *finite W* **and** *cardW*: *card W* = *n*
by (*auto simp*: *W-def*)
— Easier to represent the state as maps from edges to colours, not sets of coloured edges
— colour the edges randomly
define Ω :: (*nat set* ⇒ *nat*) *set* **where** Ω ≡ (*all-edges W*) →$_E$ {..<*2*}
have *card*Ω: *card* Ω = *2* ^ (*n choose 2*)
by (*simp add*: Ω-*def* ‹*finite W*› *W-def card-all-edges card-funcsetE finite-all-edges*)
define *coloured* **where** *coloured* ≡ λ*F*. λ*f*::*nat set* ⇒ *nat*. λ*c*. {*e* ∈ *F*. *f e* = *c*}
have *finite-coloured*[*simp*]: *finite* (*coloured F f c*) **if** *finite F* **for** *f c F*
using *coloured-def that* **by** *auto*
define *pr* **where** *pr* ≡ λ*F f*. *p* ^ *card* (*coloured F f 0*) * (*1*−*p*) ^ *card* (*coloured F f 1*)
have *pr01*: *0* < *pr U f pr U f* ≤ *1* **for** *U f* — the inequality could be strict
using ‹*0*<*p*› ‹*p*<*1*› **by** (*auto simp*: *mult-le-one power-le-one pr-def card*Ω)
define *M* **where** *M* ≡ *point-measure* Ω (*pr* (*all-edges W*))
have *space-eq*: *space M* = Ω
by (*simp add*: *M-def space-point-measure*)
have *sets-eq*: *sets M* = *Pow* Ω
by (*simp add*: *M-def sets-point-measure*)
have *fin*-Ω[*simp*]: *finite* Ω
by (*simp add*: Ω-*def finite-PiE* ‹*finite W*› *finite-all-edges*)
have *coloured-insert*:
coloured (*insert e F*) *f c* = (*if f e* = *c then insert e* (*coloured F f c*) *else coloured F f c*)
for *f e c F*
by (*auto simp*: *coloured-def*)
have *eq2*: {..<*2*} = {*0*, *Suc 0*}
by (*simp add*: *insert-commute lessThan-Suc numeral-2-eq-2*)
have *sum-pr-1* [*simp*]: *sum* (*pr U*) (*U* →$_E$ {..<*2*}) = *1* **if** *finite U* **for** *U*
using *that*
proof (*induction U*)
case *empty*
then show *?case*
by (*simp add*: *pr-def coloured-def*)

```
  next
    case (insert e F)
    then have [simp]: e ∉ coloured F f c coloured F (f(e := c)) c' = coloured F f
c' for f c c'
      by (auto simp: coloured-def)
    have inj: inj-on (λ(y, g). g(e := y)) ({..<2} × (F →_E {..<2}))
      using ‹e ∉ F› by (fastforce simp: inj-on-def fun-eq-iff)
    show ?case
      using insert
       apply (simp add: pr-def coloured-insert PiE-insert-eq sum.reindex [OF inj]
sum.cartesian-product')
      apply (simp add: eq2 mult-ac flip: sum-distrib-left)
      done
  qed

  interpret P: prob-space M
  proof
    have sum (pr (all-edges W)) Ω = 1
      using Ω-def sum-pr-1 ‹finite W› finite-all-edges by blast
    with pr01 show emeasure M (space M) = 1
      unfolding M-def
      by (metis fin-Ω prob-space.emeasure-space-1 prob-space-point-measure zero-le
       ennreal-1 linorder-not-less nle-le sum-ennreal)
  qed
  — the event to avoid: monochromatic cliques, given K ⊆ W; we are considering
edges over the entire graph W
  define mono where mono ≡ λc K. {f ∈ Ω. all-edges K ⊆ coloured (all-edges
W) f c}
  have mono-ev: mono c K ∈ P.events if c<2 for K c
    by (auto simp: sets-eq mono-def Ω-def)
  have mono-sub-Ω: mono c K ⊆ Ω if c<2 for K c
    using mono-ev sets-eq that by auto

   have emeasure-eq: emeasure M C = (if C ⊆ Ω then (∑a∈C. ennreal (pr
(all-edges W) a)) else 0) for C
    by (simp add: M-def emeasure-notin-sets emeasure-point-measure-finite sets-point-measure)
  define pc where pc ≡ λc::nat. if c=0 then p else 1−p
  have pc0: 0 ≤ pc c for c
    using p01 pc-def by auto
  have coloured-upd: coloured F (λl∈F. if l ∈ G then c else f l) c'
        = (if c=c' then G ∪ coloured (F−G) f c' else coloured (F−G) f c') if G ⊆
F for F G f c c'
    using that by (auto simp: coloured-def)

  have prob-mono: P.prob (mono c K) = pc c ^ (r choose 2)
    if K ∈ nsets W r c<2 for r K c
  proof −
    let ?EWK = all-edges W − all-edges K
    have §: K ⊆ W finite K card K = r
```

```
    using that by (auto simp: nsets-def)
  have *: {f ∈ Ω. all-edges K ⊆ coloured (all-edges W) f c} =
        (⋃g ∈ ?EWK →E {..<2}. {λl ∈ all-edges W. if l ∈ all-edges K then c
else g l})
    (is ?L = ?R)
  proof
    have ∃g∈?EWK →E {..<2}. f = (λl∈all-edges W. if l ∈ all-edges K then
c else g l)
      if f: f ∈ Ω and c: all-edges K ⊆ coloured (all-edges W) f c for f
      using that
      apply (intro bexI [where x=restrict f ?EWK])
       apply (force simp: Ω-def coloured-def subset-iff)+
      done
    then show ?L ⊆ ?R by auto
    show ?R ⊆ ?L
     using that all-edges-mono[OF ‹K ⊆ W›] by (auto simp: coloured-def Ω-def
nsets-def PiE-iff)
  qed

  have [simp]: card (all-edges K ∪ coloured ?EWK f c)
          = (r choose 2) + card (coloured ?EWK f c) for f c
    using § ‹finite W›
   by (subst card-Un-disjoint) (auto simp: finite-all-edges coloured-def card-all-edges)
  have pr-upd: pr (all-edges W) (λl ∈ all-edges W. if l ∈ all-edges K then c else
f l)
      = pc c ^ (r choose 2) * pr ?EWK f
    if f ∈ ?EWK →E {..<2} for f
    using that all-edges-mono[OF ‹K ⊆ W›] p01 ‹c<2› §
    by (simp add: pr-def coloured-upd pc-def power-add)
  have emeasure M (mono c K) = (∑f ∈ mono c K. ennreal (pr (all-edges W)
f))
    using that by (simp add: emeasure-eq mono-sub-Ω)
  also have ... = (∑f∈(⋃g∈?EWK →E {..<2}.
                      {λe∈all-edges W. if e ∈ all-edges K then c else g e}).
                ennreal (pr (all-edges W) f))
    by (simp add: mono-def *)
  also have ... = (∑g∈?EWK →E {..<2}.
                   ∑f∈{λe∈all-edges W. if e ∈ all-edges K then c else g e}.
                     ennreal (pr (all-edges W) f))
  proof (rule sum.UNION-disjoint-family)
    show finite (?EWK →E {..<2::nat})
      by (simp add: ‹finite W› finite-PiE finite-all-edges)
    show disjoint-family-on (λg. {λe∈all-edges W. if e ∈ all-edges K then c else
g e}) (?EWK →E {..<2})
      apply (simp add: disjoint-family-on-def fun-eq-iff)
      by (metis DiffE PiE-E)
  qed auto
  also have ... = (∑x∈?EWK →E {..<2}. ennreal (pc c ^ (r choose 2) * pr
?EWK x))
```

```
    by (simp add: pr-upd)
  also have ... = ennreal (∑f∈?EWK →E {..<2}.
                           pc c ^ (r choose 2) * pr ?EWK f)
    using pr01 pc0 sum.cong sum-ennreal by (smt (verit) mult-nonneg-nonneg
zero-le-power)
  also have ... = ennreal (pc c ^ (r choose 2))
    by (simp add: ‹finite W› finite-all-edges flip: sum-distrib-left)
  finally have emeasure M (mono c K) = ennreal (pc c ^ (r choose 2)) .
  then show ?thesis
    using p01 that by (simp add: measure-eq-emeasure-eq-ennreal pc-def)
 qed
 define Reds where Reds ≡ (⋃K ∈ nsets W k. mono 0 K)
 define Blues where Blues ≡ (⋃K ∈ nsets W l. mono 1 K)
 have Uev: ⋃ (mono c ` [W]^r) ∈ P.events for c r
   by (simp add: local.mono-def sets-eq subset-iff)
 then have Reds ∈ P.events Blues ∈ P.events
   by (auto simp: Reds-def Blues-def)
 have prob-0: P.prob Reds ≤ (n choose k) * (p ^ (k choose 2))
 proof −
   have P.prob Reds ≤ (∑K ∈ nsets W k. P.prob (mono 0 K))
     by (simp add: Reds-def ‹finite W› finite-imp-finite-nsets measure-UNION-le
mono-ev)
   also have ... ≤ (n choose k) * (p ^ (k choose 2))
     by (simp add: prob-mono pc-def cardW)
   finally show ?thesis .
 qed
 moreover
 have prob-1: P.prob Blues ≤ (n choose l) * ((1−p) ^ (l choose 2))
 proof −
   have P.prob Blues ≤ (∑K ∈ nsets W l. P.prob (mono 1 K))
     by (simp add: Blues-def ‹finite W› finite-imp-finite-nsets measure-UNION-le
mono-ev)
   also have ... ≤ (n choose l) * ((1−p) ^ (l choose 2))
     by (simp add: prob-mono pc-def cardW)
   finally show ?thesis .
 qed
 ultimately have P.prob (Reds ∪ Blues) < 1
   using P.finite-measure-subadditive ‹Blues ∈ P.events› ‹Reds ∈ P.events› n
   by fastforce
 with P.prob-space Uev sets-eq obtain F where F: F ∈ Ω − (Reds ∪ Blues)
   unfolding Reds-def Blues-def space-eq
   by (smt (verit, del-insts) Pow-iff Un-subset-iff equalityI Diff-iff subset-iff)
 have False if i < 2 H ∈ [W]^([k, l] ! i) F ` [H]^2 ⊆ {i} for i H
 proof −
   have ¬ all-edges H ⊆ {e ∈ all-edges W. F e = 0} ¬ all-edges H ⊆ {e ∈
all-edges W. F e = 1}
     using F that
     by (auto simp: less-2-cases-iff nsets2-eq-all-edges Ω-def Reds-def Blues-def
mono-def coloured-def image-subset-iff)
```

```
    moreover have H ⊆ W
      using that by (auto simp: nsets-def)
    ultimately show False
       using that all-edges-mono [OF ‹H ⊆ W›] by (auto simp: less-2-cases-iff
nsets2-eq-all-edges)
  qed
  moreover have F ∈ [{..<n}]^2 → {..<2}
    using F by (auto simp: W-def Ω-def nsets2-eq-all-edges)
  ultimately show False
   using con by (force simp: W-def partn-lst-def monochromatic-def numeral-2-eq-2)
qed
```

Andrew's calculation for the Ramsey lower bound. Symmetric, so works for both colours

```
lemma Ramsey-lower-calc:
  fixes s::nat and t::nat and p::real
  assumes s ≥ 3 t ≥ 3 n > 4
    and n: real n ≤ exp ((real s − 1) ∗ (real t − 1) / (2∗(s+t)))
  defines p ≡ real s / (real s + real t)
  shows (n choose s) ∗ p ^ (s choose 2) < 1/2
proof −
  have p01: 0<p p<1
    using assms by (auto simp: p-def)
   have exp ((real s − 1) ∗ (real t − 1) / (2∗(s+t))) ≤ exp (t / (s+t)) powr
((s−1)/2)
    using ‹s ≥ 3› by (simp add: mult-ac divide-simps of-nat-diff exp-powr-real)
  with assms p01 have n ≤ exp (t / (s+t)) powr ((s−1)/2)
    by linarith
  then have n ∗ p powr ((s−1)/2) ≤ (exp (t / (s+t)) ∗ p) powr ((s−1)/2)
    using ‹0<p› by (simp add: powr-mult)
  also have ... < 1
  proof −
    have exp (real t / real (s+t)) ∗ p < 1
    proof −
      have p = 1 − t / (s+t)
        using assms by (simp add: p-def divide-simps)
      also have ... < exp (− real t / real (s+t))
        using assms by (simp add: exp-minus-greater)
      finally show ?thesis
        by (simp add: exp-minus divide-simps mult.commute)
    qed
    then show ?thesis
      using powr01-less-one assms(1) p01(1) by auto
  qed
  finally have n ∗ p powr ((s−1)/2) < 1 .
  then have (n ∗ p powr ((s−1)/2)) ^ s < 1
    using ‹s ≥ 3› by (simp add: power-less-one-iff)
  then have B: n^s ∗ p ^ (s choose 2) < 1
    using ‹0<p› ‹4 < n› ‹s ≥ 3›
```

by (*simp add*: *choose-two-real powr-powr powr-mult of-nat-diff mult.commute flip*: *powr-realpow*)
have (*n choose s*) * *p* ^ (*s choose 2*) ≤ *n*^*s* / *fact s* * *p* ^ (*s choose 2*)
proof (*intro mult-right-mono*)
show *real* (*n choose s*) ≤ *real* (*n* ^ *s*) / *fact s*
using *binomial-fact-pow*[*of n s*] *of-nat-mono*
by (*fastforce simp*: *divide-simps mult.commute*)
qed (*use p01* **in** *auto*)
also have ... < *1* / *fact s*
using *B* **by** (*simp add*: *divide-simps*)
also have ... ≤ *1/2*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 Suc-leD assms fact-2 fact-mono frac-less2 numeral-3-eq-3*)
finally show *?thesis* **.**
qed

Andrew Thomason's specific example

corollary *Ramsey-number-lower-off-diag*:
fixes *n k*::*nat*
assumes *k* ≥ *3 l* ≥ *3* **and** *n*: *real n* ≤ *exp* ((*real k* − *1*) * (*real l* − *1*) / (*2*∗(*k*+*l*)))
shows ¬ *is-Ramsey-number k l n*
proof
assume *con*: *is-Ramsey-number k l n*
then have (*k* − *1*) * (*l* − *1*) < *n*
using *RN-times-lower'*[*of k l*] *assms* **by** (*metis RN-le numeral-3-eq-3 order-less-le-trans zero-less-Suc*)
moreover have *2*∗*2* ≤ (*k* − *1*) * (*l* − *1*)
using *assms* **by** (*intro mult-mono*) *auto*
ultimately have *n* > *4*
by *simp*
define *p* **where** *p* ≡ *k* / (*k*+*l*)
have *p01*: *0*<*p p*<*1*
using *assms* **by** (*auto simp*: *p-def*)
have *real* (*n choose k*) * *p* ^ (*k choose 2*) < *1/2*
using *Ramsey-lower-calc* ‹*4* < *n*› *assms n p-def* **by** *auto*
moreover
have *1*−*p* = *real l* / (*real l* + *real k*)
using ‹*k* ≥ *3*› **by** (*simp add*: *p-def divide-simps*)
with *assms* **have** (*n choose l*) * (*1*−*p*) ^ (*l choose 2*) < *1/2*
by (*metis Ramsey-lower-calc add.commute mult.commute* ‹*4* < *n*›)
ultimately show *False*
using *con Ramsey-number-lower-gen p01* **by** *force*
qed

theorem *RN-lower-off-diag*:
assumes *s* ≥ *3 t* ≥ *3*
shows *RN s t* > *exp* ((*real s* − *1*) * (*real t* − *1*) / (*2*∗(*s*+*t*)))
using *Ramsey-number-lower-off-diag* [*OF assms*] *is-Ramsey-number-RN* **by** *force*

The original Ramsey number lower bound, by Erdős

```
proposition Ramsey-number-lower:
  fixes n s::nat
  assumes s ≥ 3 and n: real n ≤ 2 powr (s/2)
  shows ¬ is-Ramsey-number s s n
proof
  assume con: is-Ramsey-number s s n
  then have s ≤ n
    using RN-3plus' RN-le assms(1) le-trans by blast
  have s > 1 using assms by arith
  have n>0
    using ‹1 < s› ‹s ≤ n› by linarith
  have (n choose s) ≤ n^s / fact s  — probability calculation
    using binomial-fact-pow[of n s]
   by (smt (verit) fact-gt-zero of-nat-fact of-nat-mono of-nat-mult pos-divide-less-eq)

  then have (n choose s) * (2 / 2^(s choose 2)) ≤ 2 * n^s / (fact s * 2 ^ (s *
(s-1) div 2))
    by (simp add: choose-two divide-simps)
  also have ... ≤ 2 powr (1 + s/2) / fact s
  proof -
    have [simp]: real (s * (s - Suc 0) div 2) = real s * (real s - 1) / 2
      by (subst real-of-nat-div) auto
    have n powr s ≤ (2 powr (s/2)) powr s
      using n by (simp add: powr-mono2)
    then have n powr s ≤ 2 powr (s * s / 2)
      using ‹n>0› assms by (simp add: power2-eq-square powr-powr)
    then have 2 * n powr s ≤ 2 powr ((2 + s * s) / 2)
      by (simp add: add-divide-distrib powr-add)
    then show ?thesis
     using n ‹n>0› by (simp add: divide-simps flip: powr-realpow powr-add) argo
  qed
  also have ... < 1
  proof -
    have 2 powr (1 + (k+3)/2) < fact (k+3) for k
    proof (induction k)
      case 0
      have 2 powr (5/2) = sqrt (2^5)
        by (simp add: powr-half-sqrt-powr)
      also have ... < sqrt 36
        by (intro real-sqrt-less-mono) auto
      finally show ?case
        by (simp add: eval-nat-numeral)
    next
      case (Suc k)
       have 2 powr (1 + real (Suc k + 3) / 2) = 2 powr (1/2) * 2 powr (1 +
(k+3)/2)
        by (simp add: powr-add powr-half-sqrt-powr flip: real-sqrt-mult)
      also have ... ≤ sqrt 2 * fact (k+3)
```

```
      using Suc.IH by (simp add: powr-half-sqrt)
    also have ... < real(k + 4) * fact (k + 3)
      using sqrt2-less-2 by simp
    also have ... = fact (Suc (k + 3))
      unfolding fact-Suc by simp
    finally show ?case by simp
  qed
  then have 2 powr (1 + s/2) < fact s
    by (metis add.commute ‹s≥3› le-Suc-ex)
  then show ?thesis
    by (simp add: divide-simps)
 qed
 finally have less-1: real (n choose s) * (2 / 2 ^ (s choose 2)) < 1 .
 then have ¬ is-Ramsey-number s s n
  by (intro Ramsey-number-lower-gen [where p=1/2]) (auto simp: power-one-over)
 with con show False by blast
qed

theorem RN-lower:
 assumes k ≥ 3
 shows RN k k > 2 powr (k/2)
 using Ramsey-number-lower assms is-Ramsey-number-RN by force
```

and trivially, off the diagonal too

```
corollary RN-lower-nodiag:
 assumes k ≥ 3 l ≥ k
 shows RN k l > 2 powr (k/2)
 by (meson RN-lower RN-mono assms less-le-trans le-refl of-nat-mono)

lemma powr-half-ge:
 fixes x::real
 assumes x≥4
 shows x ≤ 2 powr (x/2)
proof −
 define f where f ≡ λx::real. 2 powr (x/2) − x
 have f 4 ≤ f x
 proof (intro DERIV-nonneg-imp-nondecreasing[of concl: f] exI conjI assms)
   show (f has-real-derivative ln 2 * (2 powr (y/2 − 1)) − 1) (at y) for y
     unfolding f-def by (rule derivative-eq-intros refl | simp add: powr-diff)+
   show ln 2 * (2 powr (y/2 − 1)) − 1 ≥ 0 if 4 ≤ y for y::real
   proof −
     have 1 ≤ ln 2 * 2 powr ((4 − 2) / (2::real))
       using ln2-ge-two-thirds by simp
     also have ... ≤ ln 2 * (2 powr (y/2 − 1))
       using that by (intro mult-left-mono powr-mono) auto
     finally show ?thesis by simp
   qed
 qed
 moreover have f 4 = 0 by (simp add: f-def)
```

ultimately show *?thesis*
by (*simp add*: *f-def*)
qed

corollary *RN-lower-self*:
assumes $k \geq 3$
shows $RN\ k\ k > k$
proof (*cases* $k{=}3$)
case *False*
with *assms* **have** $k{\geq}4$ **by** *linarith*
then have $k \leq 2\ powr\ (k/2)$
using *powr-half-ge numeral-le-real-of-nat-iff* **by** *blast*
also have $\ldots < RN\ k\ k$
using *assms* **by** (*intro RN-lower*) *auto*
finally show *?thesis*
by *fastforce*
qed (*simp add*: *RN-gt2*)

end

References

[1] B. Bollobás. *Graph Theory: An Introductory Course.* Springer, 1979.