An Exponential Improvement for Diagonal Ramsey

Lawrence C. Paulson

17 March 2025

Abstract

The (diagonal) Ramsey number $R(k)$ denotes the minimum size of a complete graph such that every red-blue colouring of its edges contains a monochromatic subgraph of size k. In 1935, Erdős and Szekeres found an upper bound, proving that $R(k) \leq 4^k$. Somewhat later, a lower bound of $\sqrt{2}^k$ was established. In subsequent improvements to the upper bound, the base of the exponent stubbornly remained at 4 until March 2023, when Campos et al. [1] sensationally showed that $R(k) \leq (4 - \epsilon)^k$ for a particular small positive ϵ.

The Isabelle/HOL formalisation of the result presented here is largely independent of the prior formalisation (in Lean) by Bhavik Mehta.

Contents

Acknowledgements Many thanks to Mantas Bakšys, Chelsea Edmonds, Simon Griffiths, Bhavik Mehta, Fedor Petrov and Andrew Thomason for their help with aspects of the proofs. The author was supported by the ERC

Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council.

1 Background material: the neighbours of vertices

Preliminaries for the Book Algorithm

theory *Neighbours* **imports** *Ramsey-Bounds.Ramsey-Bounds*

begin

abbreviation *set-difference* :: $['a\ set,'a\ set] \Rightarrow 'a\ set$ (**infixl** ‹\› *65*)
where $A \setminus B \equiv A-B$

1.1 Preliminaries on graphs

context *ulgraph*
begin

The set of *undirected* edges between two sets

definition *all-edges-betw-un* :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set\ set$ **where**
all-edges-betw-un $X\ Y \equiv \{\{x,\ y\}|\ x\ y.\ x \in X \wedge y \in Y \wedge \{x,\ y\} \in E\}$

lemma *all-edges-betw-un-commute1*: *all-edges-betw-un* $X\ Y \subseteq$ *all-edges-betw-un* Y
X
by (*smt* (*verit*, *del-insts*) *Collect-mono all-edges-betw-un-def insert-commute*)

lemma *all-edges-betw-un-commute*: *all-edges-betw-un* $X\ Y =$ *all-edges-betw-un* Y
X
by (*simp add*: *all-edges-betw-un-commute1 subset-antisym*)

lemma *all-edges-betw-un-iff-mk-edge*: *all-edges-betw-un* $X\ Y =$ *mk-edge* ‘ *all-edges-between*
$X\ Y$
using *all-edges-between-set all-edges-betw-un-def* **by** *presburger*

lemma *all-uedges-betw-subset*: *all-edges-betw-un* $X\ Y \subseteq E$
by (*auto simp*: *all-edges-betw-un-def*)

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

lemma *all-edges-betw-un-subset*: *all-edges-betw-un* $X\ Y \subseteq$ *Pow* $(X \cup Y)$
by (*auto simp*: *all-edges-betw-un-def*)

lemma *all-edges-betw-un-empty* [*simp*]:
all-edges-betw-un $\{\}\ Z = \{\}$ *all-edges-betw-un* $Z\ \{\} = \{\}$
by (*auto simp*: *all-edges-betw-un-def*)

lemma *card-all-uedges-betw-le*:
assumes *finite X finite Y*
shows *card* (*all-edges-betw-un* $X\ Y$) $\leq$ *card* (*all-edges-between* $X\ Y$)
by (*simp add*: *all-edges-betw-un-iff-mk-edge assms card-image-le finite-all-edges-between*)

lemma *all-edges-betw-un-le*:
  assumes *finite X finite Y*
  shows *card* (*all-edges-betw-un X Y*) $\leq$ *card X* $*$ *card Y*
  by (*meson assms card-all-uedges-betw-le max-all-edges-between order-trans*)

lemma *all-edges-betw-un-insert1*:
  all-edges-betw-un (*insert v X*) $Y = (\{\{v, y\}|\ y.\ y \in Y\} \cap E) \cup$ *all-edges-betw-un X Y*
  by (*auto simp*: *all-edges-betw-un-def*)

lemma *all-edges-betw-un-insert2*:
  all-edges-betw-un X (*insert v Y*) $= (\{\{x, v\}|\ x.\ x \in X\} \cap E) \cup$ *all-edges-betw-un X Y*
  by (*auto simp*: *all-edges-betw-un-def*)

lemma *all-edges-betw-un-Un1*:
  all-edges-betw-un $(X \cup Y)$ Z = *all-edges-betw-un X Z* $\cup$ *all-edges-betw-un Y Z*
  by (*auto simp*: *all-edges-betw-un-def*)

lemma *all-edges-betw-un-Un2*:
  all-edges-betw-un X $(Y \cup Z)$ = *all-edges-betw-un X Y* $\cup$ *all-edges-betw-un X Z*
  by (*auto simp*: *all-edges-betw-un-def*)

lemma *finite-all-edges-betw-un*:
  assumes *finite X finite Y*
  shows *finite* (*all-edges-betw-un X Y*)
  by (*simp add*: *all-edges-betw-un-iff-mk-edge assms finite-all-edges-between*)

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

lemma *all-edges-betw-un-Union2*:
  all-edges-betw-un X (*Union* $\mathcal{Y}$) = ($\bigcup Y \in \mathcal{Y}$. *all-edges-betw-un X Y*)
  by (*auto simp*: *all-edges-betw-un-def*)

lemma *all-edges-betw-un-mono1*:
  $Y \subseteq Z \Longrightarrow$ *all-edges-betw-un Y X* $\subseteq$ *all-edges-betw-un Z X*
  by (*auto simp*: *all-edges-betw-un-def*)

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

lemma *disjnt-all-edges-betw-un*:
  assumes *disjnt X Y disjnt X Z*
  shows *disjnt* (*all-edges-betw-un X Z*) (*all-edges-betw-un Y Z*)
  using *assms* **by** (*auto simp*: *all-edges-betw-un-def disjnt-iff doubleton-eq-iff*)

end

1.2 Neighbours of a vertex

definition *Neighbours* :: *'a set set* ⇒ *'a* ⇒ *'a set* **where**
  Neighbours ≡ λ*E x*. {*y*. {*x*,*y*} ∈ *E*}

lemma *in-Neighbours-iff*: *y* ∈ *Neighbours E x* ⟷ {*x*,*y*} ∈ *E*
  by (*simp add*: *Neighbours-def*)

lemma *finite-Neighbours*:
  assumes *finite E*
  shows *finite* (*Neighbours E x*)
proof −
  have *Neighbours E x* ⊆ *Neighbours* {*X*∈*E*. *finite X*} *x*
    by (*auto simp*: *Neighbours-def*)
  also have ... ⊆ (⋃{*X*∈*E*. *finite X*})
    by (*meson Union-iff in-Neighbours-iff insert-iff subset-iff*)
  finally show *?thesis*
    using *assms finite-subset* **by** *fastforce*
qed

lemma (**in** *fin-sgraph*) *not-own-Neighbour*: *E'* ⊆ *E* ⟹ *x* ∉ *Neighbours E' x*
  by (*force simp*: *Neighbours-def singleton-not-edge*)

context *fin-sgraph*
begin

declare *singleton-not-edge* [*simp*]

"A graph on vertex set $S \cup T$ that contains all edges incident to S" (page 3). In fact, S is a clique and every vertex in T has an edge into S.

definition *book* :: *'a set* ⇒ *'a set* ⇒ *'a set set* ⇒ *bool* **where**
  book ≡ λ*S T F*. *disjnt S T* ∧ *all-edges-betw-un S* (*S*∪*T*) ⊆ *F*

Cliques of a given number of vertices; the definition of clique from Ramsey is used

definition *size-clique* :: *nat* ⇒ *'a set* ⇒ *'a set set* ⇒ *bool* **where**
  size-clique p K F ≡ *card K* = *p* ∧ *clique K F* ∧ *K* ⊆ *V*

lemma *size-clique-smaller*: ⟦*size-clique p K F*; *p'* < *p*⟧ ⟹ ∃*K'*. *size-clique p' K' F*
  unfolding *size-clique-def*
  by (*meson card-Ex-subset order.trans less-imp-le-nat smaller-clique*)

1.3 Density: for calculating the parameter p

definition *edge-card* ≡ λ*C X Y*. *card* (*C* ∩ *all-edges-betw-un X Y*)

definition *gen-density* ≡ λ*C X Y*. *edge-card C X Y* / (*card X* ∗ *card Y*)

lemma *edge-card-empty* [*simp*]: *edge-card C* {} *X* = *0* *edge-card C X* {} = *0*
  by (*auto simp*: *edge-card-def*)

lemma *edge-card-commute*: *edge-card C X Y* = *edge-card C Y X*
  using *all-edges-betw-un-commute edge-card-def* **by** *presburger*

lemma *edge-card-le*:
  assumes *finite X finite Y*
  shows *edge-card C X Y* ≤ *card X* ∗ *card Y*
proof −
  have *edge-card C X Y* ≤ *card* (*all-edges-betw-un X Y*)
    by (*simp add*: *assms card-mono edge-card-def finite-all-edges-betw-un*)
  then show *?thesis*
    by (*meson all-edges-betw-un-le assms le-trans*)
qed

the assumption that Z is disjoint from X (or Y) is necessary

lemma *edge-card-Un*:
  assumes *disjnt X Y disjnt X Z finite X finite Y*
  shows *edge-card C* (*X* ∪ *Y*) *Z* = *edge-card C X Z* + *edge-card C Y Z*
proof −
  have [*simp*]: *finite* (*all-edges-betw-un U Z*) **for** *U*
    by (*meson all-uedges-betw-subset fin-edges finite-subset*)
  have *disjnt* (*C* ∩ *all-edges-betw-un X Z*) (*C* ∩ *all-edges-betw-un Y Z*)
    using *assms* **by** (*meson Int-iff disjnt-all-edges-betw-un disjnt-iff*)
  then show *?thesis*
   by (*simp add*: *edge-card-def card-Un-disjnt all-edges-betw-un-Un1 Int-Un-distrib*)
qed

lemma *edge-card-diff*:
  assumes *Y*⊆*X disjnt X Z finite X*
  shows *edge-card C* (*X*−*Y*) *Z* = *edge-card C X Z* − *edge-card C Y Z*
proof −
  have (*X**Y*) ∪ *Y* = *X disjnt* (*X**Y*) *Y*
    by (*auto simp*: *Un-absorb2 assms disjnt-iff*)
  then show *?thesis*
   by (*metis add-diff-cancel-right′ assms disjnt-Un1 edge-card-Un finite-Diff finite-subset*)
qed

lemma *edge-card-mono*:
  assumes *Y*⊆*X* **shows** *edge-card C Y Z* ≤ *edge-card C X Z*
  unfolding *edge-card-def*
proof (*intro card-mono*)
  show *finite* (*C* ∩ *all-edges-betw-un X Z*)
    by (*meson all-uedges-betw-subset fin-edges finite-Int finite-subset*)
  show *C* ∩ *all-edges-betw-un Y Z* ⊆ *C* ∩ *all-edges-betw-un X Z*

```
    by (meson Int-mono all-edges-betw-un-mono1 assms subset-refl)
qed

lemma edge-card-eq-sum-Neighbours:
  assumes C⊆E and B: finite B disjnt A B
  shows edge-card C A B = (∑i∈B. card (Neighbours C i ∩ A))
  using B
proof (induction B)
  case empty
  then show ?case
    by (auto simp: edge-card-def)
next
  case (insert b B)
  have finite C
    using assms(1) fin-edges finite-subset by blast
  have bij: bij-betw (λe. the-elem(e−{b})) (C ∩ {{x, b} |x. x ∈ A}) (Neighbours
C b ∩ A)
    unfolding bij-betw-def
  proof
    have [simp]: the-elem ({x, b} − {b}) = x if x ∈ A for x
      using insert.prems by (simp add: disjnt-iff insert-Diff-if that)
    show inj-on (λe. the-elem (e − {b})) (C ∩ {{x, b} |x. x ∈ A})
      by (auto simp: inj-on-def)
    show (λe. the-elem (e − {b})) ` (C ∩ {{x, b} |x. x ∈ A}) = Neighbours C b
∩ A
      by (fastforce simp: Neighbours-def insert-commute image-iff Bex-def)
  qed
  have (C ∩ all-edges-betw-un A (insert b B)) = (C ∩ ({{x, b} |x. x ∈ A} ∪
all-edges-betw-un A B))
    using ‹C ⊆ E› by (auto simp: all-edges-betw-un-insert2)
  then have edge-card C A (insert b B) = card ((C ∩ ({{x,b} |x. x ∈ A}) ∪ (C
∩ all-edges-betw-un A B)))
    by (simp add: edge-card-def Int-Un-distrib)
  also have ... = card (C ∩ {{x,b} |x. x ∈ A}) + card (C ∩ all-edges-betw-un
A B)
  proof (rule card-Un-disjnt)
    show disjnt (C ∩ {{x, b} |x. x ∈ A}) (C ∩ all-edges-betw-un A B)
      using insert by (auto simp: disjnt-iff all-edges-betw-un-def doubleton-eq-iff)
  qed (use ‹finite C› in auto)
  also have ... = card (Neighbours C b ∩ A) + card (C ∩ all-edges-betw-un A B)
    using bij-betw-same-card [OF bij] by simp
  also have ... = (∑i∈insert b B. card (Neighbours C i ∩ A))
    using insert by (simp add: edge-card-def)
  finally show ?case .
qed

lemma sum-eq-card: finite A ⟹ (∑x ∈ A. if x ∈ B then 1 else 0) = card (A∩B)
  by (metis (no-types, lifting) card-eq-sum sum.cong sum.inter-restrict)
```

lemma *sum-eq-card-Neighbours*:
  assumes $x \in V$ $C \subseteq E$
  shows $(\sum y \in V\setminus\{x\}.$ *if* $\{x,y\} \in C$ *then 1 else 0*$) =$ *card* (*Neighbours C x*)
proof –
  have *Neighbours C x* $= (V \setminus \{x\}) \cap \{y.\ \{x, y\} \in C\}$
    using *assms wellformed* **by** (*auto simp*: *Neighbours-def*)
  with *finV sum-eq-card* [*of* - $\{y.\ \{x,y\}\in C\}$] **show** *?thesis* **by** *simp*
qed

lemma *Neighbours-insert-NO-MATCH*: *NO-MATCH* {} $C \Longrightarrow$ *Neighbours* (*insert e C*) $x =$ *Neighbours* $\{e\}$ $x \cup$ *Neighbours C x*
  by (*auto simp*: *Neighbours-def*)

lemma *Neighbours-sing-2*:
  assumes $e \in E$
  shows $(\sum x\in V.$ *card* (*Neighbours* $\{e\}$ x)) = *2*
proof –
  obtain *u v* **where** *uv*: $e = \{u,v\}$ $u \neq v$
    by (*meson assms card-2-iff two-edges*)
  then have $u \in V$ $v \in V$
    using *assms wellformed uv* **by** *blast+*
  have $*$: *Neighbours* $\{e\}$ $x =$ (*if* $x=u$ *then* $\{v\}$ *else if* $x=v$ *then* $\{u\}$ *else* {}) **for** x
    by (*auto simp*: *Neighbours-def uv doubleton-eq-iff*)
  show *?thesis*
    using ‹$u\neq v$›
    by (*simp add*: $*$ *if-distrib* [*of card*] *finV sum.delta-remove* ‹$u \in V$› ‹$v \in V$› *cong*: *if-cong*)
qed

lemma *sum-Neighbours-eq-card*:
  assumes *finite C* $C\subseteq E$
  shows $(\sum i\in V.$ *card* (*Neighbours C i*)) = *card C* $*$ *2*
  using *assms*
proof (*induction C*)
  case *empty*
  then show *?case*
    by (*auto simp*: *Neighbours-def*)
next
  case (*insert e C*)
  then have [*simp*]: *Neighbours* $\{e\}$ $x \cap$ *Neighbours C x* = {} **for** x
    by (*auto simp*: *Neighbours-def*)
  with *insert* **show** *?case*
   by (*auto simp*: *card-Un-disjoint finite-Neighbours Neighbours-insert-NO-MATCH sum.distrib Neighbours-sing-2*)
qed

lemma *gen-density-empty* [*simp*]: *gen-density C* {} $X = 0$ *gen-density C X* {} = *0*

by (*auto simp*: *gen-density-def*)

lemma *gen-density-commute*: *gen-density C X Y* = *gen-density C Y X*
by (*simp add*: *edge-card-commute gen-density-def*)

lemma *gen-density-ge0*: *gen-density C X Y* ≥ *0*
by (*auto simp*: *gen-density-def*)

lemma *gen-density-gt0*:
assumes *finite X finite Y* {*x*,*y*} ∈ *C x* ∈ *X y* ∈ *Y C* ⊆ *E*
shows *gen-density C X Y* > *0*
proof −
have *xy*: {*x*,*y*} ∈ *all-edges-betw-un X Y*
using *assms* **by** (*force simp*: *all-edges-betw-un-def*)
moreover have *finite* (*all-edges-betw-un X Y*)
by (*simp add*: *assms finite-all-edges-betw-un*)
ultimately have *edge-card C X Y* > *0*
by (*metis IntI assms*(*3*) *card-0-eq edge-card-def emptyE finite-Int gr0I*)
with *xy* **show** *?thesis*
using *assms gen-density-def less-eq-real-def* **by** *fastforce*
qed

lemma *gen-density-le1*: *gen-density C X Y* ≤ *1*
unfolding *gen-density-def*
by (*smt* (*verit*) *card.infinite divide-le-eq-1 edge-card-le mult-eq-0-iff of-nat-le-0-iff of-nat-mono*)

lemma *gen-density-le-1-minus*:
shows *gen-density C X Y* ≤ *1* − *gen-density* (*E*−*C*) *X Y*
proof (*cases finite X* ∧ *finite Y*)
case *True*
have *C* ∩ *all-edges-betw-un X Y* ∪ (*E* − *C*) ∩ *all-edges-betw-un X Y* = *all-edges-betw-un X Y*
by (*auto simp*: *all-edges-betw-un-def*)
with *True* **have** (*edge-card C X Y*) + (*edge-card* (*E* − *C*) *X Y*) ≤ *card* (*all-edges-betw-un X Y*)
unfolding *edge-card-def*
by (*metis Diff-Int-distrib2 Diff-disjoint card-Un-disjoint card-Un-le finite-Int finite-all-edges-betw-un*)
with *True* **show** *?thesis*
apply (*simp add*: *gen-density-def divide-simps*)
by (*smt* (*verit*) *all-edges-betw-un-le of-nat-add of-nat-mono of-nat-mult*)
qed (*auto simp*: *gen-density-def*)

lemma *gen-density-lt1*:
assumes {*x*,*y*} ∈ *E*−*C x* ∈ *X y* ∈ *Y C* ⊆ *E*
shows *gen-density C X Y* < *1*
proof (*cases finite X* ∧ *finite Y*)
case *True*

```
  then have 0 < gen-density (E − C) X Y
    using assms gen-density-gt0 by auto
  have gen-density C X Y ≤ 1 − gen-density (E − C) X Y
    by (intro gen-density-le-1-minus)
  then show ?thesis
    using ‹0 < gen-density (E − C) X Y› by linarith
qed (auto simp: gen-density-def)

lemma gen-density-le-iff:
  assumes disjnt X Z finite X Y⊆X Y ≠ {} finite Z
  shows gen-density C X Z ≤ gen-density C Y Z ⟷
        edge-card C X Z / card X ≤ edge-card C Y Z / card Y
 using assms by (simp add: gen-density-def divide-simps mult-less-0-iff zero-less-mult-iff)
```

"Removing vertices whose degree is less than the average can only increase the density from the remaining set" (page 17)

```
lemma gen-density-below-avg-ge:
  assumes disjnt X Z finite X Y⊂X finite Z
    and genY: gen-density C Y Z ≤ gen-density C X Z
  shows gen-density C (X−Y) Z ≥ gen-density C X Z
proof −
  have real (edge-card C Y Z) / card Y ≤ real (edge-card C X Z) / card X
    using assms
   by (force simp: gen-density-def divide-simps zero-less-mult-iff split: if-split-asm)
  have card Y < card X
    by (simp add: assms psubset-card-mono)
  have ∗: finite Y Y ⊆ X X≠{}
    using assms finite-subset by blast+
  then
  have card X ∗ edge-card C Y Z ≤ card Y ∗ edge-card C X Z
    using genY assms
     by (simp add: gen-density-def field-split-simps card-eq-0-iff flip: of-nat-mult
split: if-split-asm)
  with assms ∗ ‹card Y < card X› show ?thesis
    by (simp add: gen-density-le-iff field-split-simps edge-card-diff card-Diff-subset
        edge-card-mono flip: of-nat-mult)
qed

lemma edge-card-insert:
  assumes NO-MATCH {} F and e ∉ F
    shows  edge-card (insert e F) X Y = edge-card {e} X Y + edge-card F X Y
proof −
  have fin: finite (all-edges-betw-un X Y)
    by (meson all-uedges-betw-subset fin-edges finite-subset)
  have insert e F ∩ all-edges-betw-un X Y
      = {e} ∩ all-edges-betw-un X Y ∪ F ∩ all-edges-betw-un X Y
    by auto
  with ‹e∉F› show ?thesis
    by (auto simp: edge-card-def card-Un-disjoint disjoint-iff fin)
```

```
qed

lemma edge-card-sing:
  assumes e ∈ E
  shows edge-card {e} U U = (if e ⊆ U then 1 else 0)
proof (cases e ⊆ U)
  case True
  obtain x y where xy: e = {x,y} x≠y
    using assms by (metis card-2-iff two-edges)
  with True assms have {e} ∩ all-edges-betw-un U U = {e}
    by (auto simp: all-edges-betw-un-def)
  with True show ?thesis
    by (simp add: edge-card-def)
qed (auto simp: edge-card-def all-edges-betw-un-def)

lemma sum-edge-card-choose:
  assumes 2≤k C ⊆ E
  shows (∑ U∈[V]^k. edge-card C U U) = (card V − 2 choose (k−2)) ∗ card C
proof −
  have ∗: card {A ∈ [V]^k. e ⊆ A} = card V − 2 choose (k−2) if e: e ∈ C for e
  proof −
    have e ⊆ V
      using ‹C⊆E› e wellformed by force
    obtain x y where xy: e = {x,y} x≠y
      using ‹C⊆E› e by (metis in-mono card-2-iff two-edges)
    define 𝒜 where 𝒜 ≡ {A ∈ [V]^k. e ⊆ A}
    have ⋀A. A ∈ 𝒜 ⟹ A = e ∪ (A\e) ∧ A\e ∈ [V\e]^(k − 2)
      by (auto simp: 𝒜-def nsets-def xy)
    moreover have ⋀xa. ⟦xa ∈ [V \ e]^(k − 2)⟧ ⟹ e ∪ xa ∈ 𝒜
      using ‹e ⊆ V› assms
      by (auto simp: 𝒜-def nsets-def xy card-insert-if)
    ultimately have 𝒜 = (∪)e ‘ [V\e]^(k−2)
      by auto
    moreover have inj-on ((∪) e) ([V\e]^(k − 2))
      by (auto simp: inj-on-def nsets-def)
    moreover have card (V\e) = card V − 2
     by (metis ‹C⊆E› ‹e ∈ C› subsetD card-Diff-subset finV finite-subset two-edges
wellformed)
    ultimately show ?thesis
      using assms by (simp add: card-image 𝒜-def)
  qed
  have (∑ U∈[V]^k. edge-card R U U) = ((card V − 2) choose (k−2)) ∗ card R
    if finite R R ⊆ C for R
    using that
  proof (induction R)
    case empty
    then show ?case
      by (simp add: edge-card-def)
  next
```

```
    case (insert e R)
    with assms have e∈E by blast
    with insert show ?case
     by (simp add: edge-card-insert ∗ sum.distrib edge-card-sing Ramsey.finite-imp-finite-nsets

          finV flip: sum.inter-filter)
  qed
  then show ?thesis
    by (meson ‹C⊆E› fin-edges finite-subset set-eq-subset)
qed

lemma sum-nsets-Compl:
  assumes finite A k ≤ card A
  shows (∑ U∈[A]^k. f (A\U)) = (∑ U∈[A]^(card A − k). f U)
proof −
  have B ∈ (\) A ‘ [A]^k if B ∈ [A]^(card A − k) for B
  proof −
    have card (A\B) = k
      using assms that by (simp add: nsets-def card-Diff-subset)
    moreover have B = A\(A\B)
      using that by (auto simp: nsets-def)
    ultimately show ?thesis
      using assms unfolding nsets-def image-iff by blast
  qed
  then have bij-betw (λU. A\U) ([A]^k) ([A]^(card A − k))
    using assms by (auto simp: nsets-def bij-betw-def inj-on-def card-Diff-subset)
  then show ?thesis
    using sum.reindex-bij-betw by blast
qed
```

1.4 Lemma 9.2 preliminaries

Equation (45) in the text, page 30, is seemingly a huge gap. The development below relies on binomial coefficient identities.

```
definition graph-density ≡ λC. card C / card E

lemma graph-density-Un:
  assumes disjnt C D C ⊆ E D ⊆ E
  shows graph-density (C ∪ D) = graph-density C + graph-density D
proof (cases card E > 0)
  case True
  with assms obtain finite C finite D
    by (metis card-ge-0-finite finite-subset)
  with assms show ?thesis
    by (auto simp: graph-density-def card-Un-disjnt divide-simps)
qed (auto simp: graph-density-def)
```

Could be generalised to any complete graph

```
lemma density-eq-average:
```

assumes $C \subseteq E$ **and** *complete*: E = *all-edges* V
shows *graph-density* C =
real ($\sum x \in V.\ \sum y \in V\backslash\{x\}$. *if* $\{x,y\} \in C$ *then 1 else 0*) / (*card* V * (*card* V − *1*))
proof −
have *cardE*: *card* E = *card* V *choose 2*
using *card-all-edges complete finV* **by** *blast*
have *finite* C
using *assms fin-edges finite-subset* **by** *blast*
then have ∗: ($\sum x{\in}V.\ \sum y{\in}V\backslash\{x\}$. *if* $\{x, y\} \in C$ *then 1 else 0*) = *card* C * *2*
using *assms* **by** (*simp add*: *sum-eq-card-Neighbours sum-Neighbours-eq-card*)
show *?thesis*
by (*auto simp*: *graph-density-def divide-simps cardE choose-two-real* ∗)
qed

lemma *edge-card-V-V*:
assumes $C \subseteq E$ **and** *complete*: E = *all-edges* V
shows *edge-card* $C\ V\ V$ = *card* C
proof −
have $C \subseteq$ *all-edges-betw-un* $V\ V$
using *assms clique-iff complete subset-refl*
by (*metis all-uedges-betw-I all-uedges-betw-subset clique-def*)
then show *?thesis*
by (*metis Int-absorb2 edge-card-def*)
qed

Bhavik's statement; own proof

proposition *density-eq-average-partition*:
assumes k: *0* < k k < *card* V **and** $C \subseteq E$ **and** *complete*: E = *all-edges* V
shows *graph-density* C = ($\sum U{\in}[V]^k$. *gen-density* $C\ U\ (V\backslash U)$) / (*card* V *choose* k)
proof (*cases* k=*1* ∨ *gorder* = *Suc* k)
case *True*
then have [*simp*]: *gorder choose* k = *gorder* **by** *auto*
have *eq*: ($C \cap \{\{x, y\} \mid y.\ y \in V \wedge y \neq x \wedge \{x, y\} \in E\}$)
= ($\lambda y.\ \{x,y\}$) ‘ $\{y.\ \{x,y\} \in C\}$ **for** x
using ‹$C{\subseteq}E$› *wellformed* **by** *fastforce*
have $V \neq \{\}$
using *assms* **by** *force*
then have *nontriv*: $E \neq \{\}$
using *assms card-all-edges finV* **by** *force*
have ($\sum U{\in}[V]^k$. *gen-density* $C\ U\ (V \setminus U)$) = ($\sum x{\in}V$. *gen-density* $C\ \{x\}\ (V \setminus \{x\})$)
using *True*
proof
assume k = *1*
then show *?thesis*
by (*simp add*: *sum-nsets-one*)
next

assume §: *gorder* = *Suc k*
then have $V - A \neq \{\}$ **if** *card A* = *k finite A* **for** *A*
using *that*
by (*metis assms(2) card.empty card-less-sym-Diff finV less-nat-zero-code*)
then have *bij*: *bij-betw* ($\lambda x.\ V \setminus \{x\}$) *V* ($[V]^k$)
using *finV* §
by (*auto simp*: *inj-on-def bij-betw-def nsets-def image-iff*)
(*metis Diff-insert-absorb card.insert card-subset-eq insert-subset subsetI*)
moreover have $V \setminus (V \setminus \{x\}) = \{x\}$ **if** $x \in V$ **for** *x*
using *that* **by** *auto*
ultimately show *?thesis*
using *sum.reindex-bij-betw* [*OF bij*] *gen-density-commute*
by (*metis* (*no-types*, *lifting*) *sum.cong*)
qed
also have ... = ($\sum x \in V$. *real* (*edge-card C* {*x*} ($V \setminus \{x\}$))) / (*gorder* − *1*)
by (*simp add*: ‹*C⊆E*› *gen-density-def flip*: *sum-divide-distrib*)
also have ... = ($\sum i \in V$. *card* (*Neighbours C i*)) / (*gorder* − *1*)
unfolding *edge-card-def Neighbours-def all-edges-betw-un-def*
by (*simp add*: *eq card-image inj-on-def doubleton-eq-iff*)
also have ... = *graph-density C* ∗ *gorder*
using *assms density-eq-average* [*OF* ‹*C⊆E*› *complete*]
by (*simp add*: *sum-eq-card-Neighbours*)
finally show *?thesis*
using *k* **by** *simp*
next
case *False*
then have *K*: *gorder* > *Suc k* $k \geq 2$
using *assms* **by** *auto*
then have *gorder* − *Suc* (*Suc* (*gorder* − *Suc* (*Suc k*))) = *k*
using *assms* **by** *auto*
then have [*simp*]: *gorder* − *2 choose* (*gorder* − *Suc* (*Suc k*)) = (*gorder* − *2* *choose k*)
using *binomial-symmetric* [*of* (*gorder* − *Suc* (*Suc k*))]
by *simp*
have *cardE*: *card E* = *card V choose 2*
using *card-all-edges complete finV* **by** *blast*
have *card E* > *0*
using *k cardE* **by** *auto*
have *in-E-iff* [*iff*]: {*v*,*w*} ∈ *E* ⟷ $v \in V \wedge w \in V \wedge v \neq w$ **for** *v w*
by (*auto simp*: *complete all-edges-alt doubleton-eq-iff*)

have *B*: *edge-card C V V* = *edge-card C U U* + *edge-card C U* ($V \setminus U$) + *edge-card C* ($V \setminus U$) ($V \setminus U$)
(**is** *?L* = *?R*)
if $U \subseteq V$ **for** *U*
proof −
have *fin*: *finite* (*all-edges-betw-un U U'*) **for** *U'*
by (*meson all-uedges-betw-subset fin-edges finite-subset*)
have *dis*: *all-edges-betw-un U U* ∩ *all-edges-betw-un U* ($V \setminus U$) = {}

by (*auto simp*: *all-edges-betw-un-def doubleton-eq-iff*)
have *all-edges-betw-un V V* = *all-edges-betw-un U U* ∪ *all-edges-betw-un U* (*V**U*) ∪ *all-edges-betw-un* (*V**U*) (*V**U*)
by (*smt* (*verit*) *that Diff-partition Un-absorb Un-assoc all-edges-betw-un-Un2 all-edges-betw-un-commute*)
with *that* **have** *?L* = *card* (*C* ∩ *all-edges-betw-un U U* ∪ *C* ∩ *all-edges-betw-un U* (*V* \ *U*)
∪ *C* ∩ *all-edges-betw-un* (*V* \ *U*) (*V* \ *U*))
by (*simp add*: *edge-card-def Int-Un-distrib*)
also have ... = *?R*
using *fin dis* ‹*C*⊆*E*› *fin-edges finite-subset*
by ((*subst card-Un-disjoint*)*?*, *fastforce simp*: *edge-card-def all-edges-betw-un-def doubleton-eq-iff*)+
finally show *?thesis* **.**
qed
have *C*: ($\sum$ *U*∈[*V*]^k. *real* (*edge-card C U* (*V**U*)))
= (*card V choose k*) ∗ *card C* − *real*($\sum$ *U*∈[*V*]^k. *edge-card C U U* + *edge-card C* (*V**U*) (*V**U*))
(**is** *?L* = *?R*)
proof −
have *?L* = ($\sum$ *U*∈[*V*]^k. *edge-card C V V* − *real* (*edge-card C U U* + *edge-card C* (*V**U*) (*V**U*)))
unfolding *nsets-def* **by** (*rule sum.cong*) (*auto simp*: *B*)
also have ... = *?R*
using ‹*C*⊆*E*› *complete edge-card-V-V*
by (*simp add*: ‹*C*⊆*E*› *sum-subtractf edge-card-V-V*)
finally show *?thesis* **.**
qed

have (*gorder*−*2 choose k*) + (*gorder*−*2 choose* (*k*−*2*)) + *2* ∗ (*gorder*−*2 choose* (*k*−*1*)) = (*gorder choose k*)
using *assms K* **by** (*auto simp*: *choose-reduce-nat* [*of gorder*] *choose-reduce-nat* [*of gorder*−*Suc 0*] *eval-nat-numeral*)
moreover
have (*gorder* − *1*) ∗ (*gorder*−*2 choose* (*k*−*1*)) = (*gorder*−*k*) ∗ (*gorder*−*1 choose* (*k*−*1*))
by (*metis Suc-1 Suc-diff-1 binomial-absorb-comp diff-Suc-eq-diff-pred* ‹*k*>*0*›)
ultimately have *F*: (*gorder* − *1*) ∗ (*gorder*−*2 choose k*) + (*gorder* − *1*) ∗ (*gorder*−*2 choose* (*k*−*2*)) + *2* ∗ (*gorder*−*k*) ∗ (*gorder*−*1 choose* (*k*−*1*))
= (*gorder* − *1*) ∗ (*gorder choose k*)
by (*smt* (*verit*) *add-mult-distrib2 mult.assoc mult.left-commute*)

have ($\sum$ *U*∈[*V*]^k. *edge-card C U* (*V**U*) / (*real* (*card U*) ∗ *card* (*V**U*)))
= ($\sum$ *U*∈[*V*]^k. *edge-card C U* (*V**U*) / (*real k* ∗ (*card V* − *k*)))
using *card-Diff-subset* **by** (*intro sum.cong*) (*auto simp*: *nsets-def*)
also have ... = ($\sum$ *U*∈[*V*]^k. *edge-card C U* (*V**U*)) / (*k* ∗ (*card V* − *k*))
by (*simp add*: *sum-divide-distrib*)
finally have ∗: ($\sum$ *U*∈[*V*]^k. *edge-card C U* (*V**U*) / (*real* (*card U*) ∗ *card* (*V**U*)))

```
            = (∑ U∈[V]^k. edge-card C U (V\U)) / (k * (card V − k)) .

  have choose-m1: gorder * (gorder − 1 choose (k − 1)) = k * (gorder choose k)
    using ‹k>0› times-binomial-minus1-eq by presburger
  have **: (real k * (real gorder − real k) * real (gorder choose k)) =
        (real (gorder choose k) − (real (gorder − 2 choose (k − 2)) + real (gorder
− 2 choose k))) *
        real (gorder choose 2)
     using assms K arg-cong [OF F, of λu. real gorder * real u] arg-cong [OF
choose-m1, of real]
    apply (simp add: choose-two-real ring-distribs)
    by (smt (verit) distrib-right mult.assoc mult-2-right mult-of-nat-commute)
  have eq: (∑ U∈[V]^k. real (edge-card C (V\U) (V\U)))
          = (∑ U∈[V]^(gorder−k). real (edge-card C U U))
    using K finV by (subst sum-nsets-Compl, simp-all)
  show ?thesis
    unfolding graph-density-def gen-density-def
    using K ‹card E > 0› ‹C⊆E›
    apply (simp add: eq divide-simps B C sum.distrib *)
    apply (simp add: ** sum-edge-card-choose cardE flip: of-nat-sum)
    by argo
qed

lemma exists-density-edge-density:
  assumes k: 0 < k k < card V and C ⊆ E and complete: E = all-edges V
  obtains U where card U = k U⊆V graph-density C ≤ gen-density C U (V\U)
proof −
  have False if ⋀U. U ∈ [V]^k ⟹ graph-density C > gen-density C U (V\U)
  proof −
    have card([V]^k) > 0
      using assms by auto
   then have (∑ U∈[V]^k. gen-density C U (V \ U)) < card([V]^k) * graph-density
C
      by (meson sum-bounded-above-strict that)
    with density-eq-average-partition assms show False by force
  qed
  with that show thesis
    unfolding nsets-def by fastforce
qed

end

end
```

2 The book algorithm

```
theory Book imports
  Neighbours
```

HOL−Library.Disjoint-Sets HOL−Decision-Procs.Approximation HOL−Real-Asymp.Real-Asymp

begin

hide-const *Bseq*

2.1 Locales for the parameters of the construction

type-synonym *'a config = 'a set × 'a set × 'a set × 'a set*

locale *P0-min =*
fixes *p0-min :: real*
assumes *p0-min: 0 < p0-min p0-min < 1*

locale *Book-Basis = fin-sgraph + P0-min +* — building on finite simple graphs (no loops)
assumes *complete: E = all-edges V*
assumes *infinite-UNIV: infinite (UNIV::'a set)*
begin

abbreviation *nV ≡ card V*

lemma *graph-size: graph-size = (nV choose 2)*
using *card-all-edges complete finV* **by** *blast*

lemma *in-E-iff [iff]: {v,w} ∈ E ⟷ v∈V ∧ w∈V ∧ v≠w*
by *(auto simp: complete all-edges-alt doubleton-eq-iff)*

lemma *all-edges-betw-un-iff-clique: K ⊆ V ⟹ all-edges-betw-un K K ⊆ F ⟷ clique K F*
unfolding *clique-def all-edges-betw-un-def doubleton-eq-iff subset-iff*
by *blast*

lemma *clique-Un:*
assumes *clique A F clique B F all-edges-betw-un A B ⊆ F A ⊆ V B ⊆ V*
shows *clique (A ∪ B) F*
using *assms* **by** *(simp add: all-uedges-betw-I clique-Un subset-iff)*

lemma *clique-insert:*
assumes *clique A F all-edges-betw-un {x} A ⊆ F A ⊆ V x ∈ V*
shows *clique (insert x A) F*
using *assms*
by *(metis Un-subset-iff clique-def insert-is-Un insert-subset clique-Un singletonD)*

lemma *less-RN-Red-Blue:*
fixes *l k*
assumes *nV: nV < RN k l*
obtains *Red Blue :: 'a set set*

```
    where Red ⊆ E Blue = E\Red ¬ (∃K. size-clique k K Red) ¬ (∃K. size-clique
  l K Blue)
proof −
  have ¬ is-Ramsey-number k l nV
    using RN-le assms leD by blast
  then obtain f where f: f ∈ nsets {..<nV} 2 → {..<2}
           and noclique: ⋀i. i<2 ⟹ ¬ monochromatic {..<nV} ([k,l] ! i) 2 f i
    by (auto simp: partn-lst-def eval-nat-numeral)
  obtain φ where φ: bij-betw φ {..<nV} V
    using bij-betw-from-nat-into-finite finV by blast
  define ϑ where ϑ ≡ inv-into {..<nV} φ
  have ϑ: bij-betw ϑ V {..<nV}
    using φ ϑ-def bij-betw-inv-into by blast
  have emap: bij-betw (λe. φ‘e) (nsets {..<nV} 2) E
    by (metis φ bij-betw-nsets complete nsets2-eq-all-edges)
  define Red  where Red ≡  (λe. φ‘e) ‘ ((f −‘ {0}) ∩ nsets {..<nV} 2)
  define Blue where Blue ≡ (λe. φ‘e) ‘ ((f −‘ {1}) ∩ nsets {..<nV} 2)
  have f0: f (ϑ‘e) = 0 if e ∈ Red for e
   using that φ by (auto simp add: Red-def image-iff ϑ-def bij-betw-def nsets-def)
  have f1: f (ϑ‘e) = 1 if e ∈ Blue for e
   using that φ by (auto simp add: Blue-def image-iff ϑ-def bij-betw-def nsets-def)
  have Red ⊆ E
    using bij-betw-imp-surj-on[OF emap] by (auto simp: Red-def)
  have Blue = E−Red
    using emap f
    by (auto simp: Red-def Blue-def bij-betw-def inj-on-eq-iff image-iff Pi-iff)
  have no-Red-K: False if size-clique k K Red for K
  proof −
    have clique K Red and Kk: card K = k and K⊆V
      using that by (auto simp: size-clique-def)
    then have f ‘ [ϑ‘K]^2 ⊆ {0}
      unfolding clique-def image-subset-iff
       by (smt (verit, ccfv-SIG) f0 image-empty image-iff image-insert nsets2-E
  singleton-iff)
    moreover have ϑ‘K ∈ [{..<nV}]^(card K)
        by (smt (verit) ‹K⊆V› ϑ bij-betwE bij-betw-nsets finV mem-Collect-eq
  nsets-def finite-subset)
    ultimately show False
      using noclique [of 0] Kk by (simp add: size-clique-def monochromatic-def)
  qed
  have no-Blue-K: False if size-clique l K Blue for K
  proof −
    have clique K Blue and Kl: card K = l and K⊆V
      using that by (auto simp: size-clique-def)
    then have f ‘ [ϑ‘K]^2 ⊆ {1}
      unfolding clique-def image-subset-iff
       by (smt (verit, ccfv-SIG) f1 image-empty image-iff image-insert nsets2-E
  singleton-iff)
    moreover have ϑ‘K ∈ [{..<nV}]^(card K)
```

using *bij-betw-nsets* [*OF* ϑ] ‹$K \subseteq V$› *bij-betwE finV infinite-super nsets-def*
by *fastforce*
ultimately show *False*
using *noclique* [*of 1*] *Kl* **by** (*simp add*: *size-clique-def monochromatic-def*)
qed
show *thesis*
using ‹*Blue* = $E \setminus$ *Red*› ‹*Red* $\subseteq E$› *no-Blue-K no-Red-K that* **by** *presburger*
qed

end

locale *No-Cliques* = *Book-Basis* +
fixes *Red Blue* :: *'a set set*
assumes *Red-E*: *Red* $\subseteq E$
assumes *Blue-def*: *Blue* = *E*−*Red*
— the following are local to the program
fixes *l*::*nat* — blue limit
fixes *k*::*nat* — red limit
assumes *l-le-k*: $l \leq k$ — they should be "sufficiently large"
assumes *no-Red-clique*: ¬ (∃ *K*. *size-clique k K Red*)
assumes *no-Blue-clique*: ¬ (∃ *K*. *size-clique l K Blue*)

locale *Book* = *Book-Basis* + *No-Cliques* +
fixes μ::*real* — governs the big blue steps
assumes μ*01*: $0 < \mu$ $\mu < 1$
fixes *X0* :: *'a set* **and** *Y0* :: *'a set* — initial values
assumes *XY0*: *disjnt X0 Y0 X0* $\subseteq$ *V Y0* $\subseteq$ *V*
assumes *density-ge-p0-min*: *gen-density Red X0 Y0* $\geq$ *p0-min*

locale *Book'* = *Book-Basis* + *No-Cliques* +
fixes γ::*real* — governs the big blue steps
assumes γ*-def*: γ = *real l* / (*real k* + *real l*)
fixes *X0* :: *'a set* **and** *Y0* :: *'a set* — initial values
assumes *XY0*: *disjnt X0 Y0 X0* $\subseteq$ *V Y0* $\subseteq$ *V*
assumes *density-ge-p0-min*: *gen-density Red X0 Y0* $\geq$ *p0-min*

definition *eps* ≡ λ*k*. *real k powr* (−*1*/*4*)

definition *qfun-base* :: [*nat*, *nat*] ⇒ *real*
where *qfun-base* ≡ λ*k h*. ((*1* + *eps k*)^*h* − *1*) / *k*

definition *hgt-maximum* ≡ λ*k*. *2* ∗ *ln* (*real k*) / *eps k*

The first of many "bigness assumptions"

definition *Big-height-upper-bound* ≡ λ*k*. *qfun-base k* (*nat* ⌊*hgt-maximum k*⌋) > *1*

lemma *Big-height-upper-bound*:
shows $\forall^\infty$*k*. *Big-height-upper-bound k*
unfolding *Big-height-upper-bound-def hgt-maximum-def eps-def qfun-base-def*

by *real-asymp*

context *No-Cliques*
begin

abbreviation $\varepsilon \equiv$ *eps k*

lemma *eps-eq-sqrt*: ε = *1 / sqrt (sqrt (real k))*
by (*simp add*: *eps-def powr-minus-divide powr-powr flip*: *powr-half-sqrt*)

lemma *eps-ge0*: $\varepsilon \geq 0$
by (*simp add*: *eps-def*)

lemma *ln0*: *l*>*0*
using *no-Blue-clique* **by** (*force simp*: *size-clique-def clique-def*)

lemma *kn0*: *k* > *0*
using *l-le-k ln0* **by** *auto*

lemma *eps-gt0*: $\varepsilon > 0$
by (*simp add*: *eps-def kn0*)

lemma *eps-le1*: $\varepsilon \leq 1$
using *kn0 ge-one-powr-ge-zero*
by (*simp add*: *eps-def powr-minus powr-mono2 divide-simps*)

lemma *eps-less1*:
assumes *k*>*1* **shows** $\varepsilon < 1$
by (*smt* (*verit*) *assms eps-def less-imp-of-nat-less of-nat-1 powr-less-one zero-le-divide-iff*)

lemma *Blue-E*: *Blue* ⊆ *E*
by (*simp add*: *Blue-def*)

lemma *disjnt-Red-Blue*: *disjnt Red Blue*
by (*simp add*: *Blue-def disjnt-def*)

lemma *Red-Blue-all*: *Red* ∪ *Blue* = *all-edges V*
using *Blue-def Red-E complete* **by** *blast*

lemma *Blue-eq*: *Blue* = *all-edges V* − *Red*
using *Blue-def complete* **by** *auto*

lemma *Red-eq*: *Red* = *all-edges V* − *Blue*
using *Blue-eq Red-Blue-all* **by** *blast*

lemma *disjnt-Red-Blue-Neighbours*: *disjnt* (*Neighbours Red x* ∩ *X*) (*Neighbours Blue x* ∩ *X'*)
using *disjnt-Red-Blue* **by** (*auto simp*: *disjnt-def Neighbours-def*)

lemma *indep-Red-iff-clique-Blue*: $K \subseteq V \Longrightarrow$ *indep K Red* $\longleftrightarrow$ *clique K Blue*
using *Blue-eq* **by** *auto*

lemma *Red-Blue-RN*:
fixes X :: *'a set*
assumes *card X* $\geq$ *RN m n X*$\subseteq$*V*
shows $\exists K \subseteq X$. *size-clique m K Red* $\vee$ *size-clique n K Blue*
using *partn-lst-imp-is-clique-RN* [*OF is-Ramsey-number-RN* [*of m n*]] *assms indep-Red-iff-clique-Blue*
unfolding *is-clique-RN-def size-clique-def clique-indep-def*
by (*metis finV finite-subset subset-eq*)

end

context *Book*
begin

lemma *Red-edges-XY0*: *Red* $\cap$ *all-edges-betw-un X0 Y0* $\neq$ {}
using *density-ge-p0-min p0-min*
by (*auto simp*: *gen-density-def edge-card-def*)

lemma *finite-X0*: *finite X0* **and** *finite-Y0*: *finite Y0*
using *XY0 finV finite-subset* **by** *blast+*

lemma *Red-nonempty*: *Red* $\neq$ {}
using *Red-edges-XY0* **by** *blast*

lemma *gorder-ge2*: *gorder* $\geq$ *2*
using *Red-nonempty*
by (*metis Red-E card-mono equals0I finV subset-empty two-edges wellformed*)

lemma *nontriv*: $E \neq$ {}
using *Red-E Red-nonempty* **by** *force*

lemma *no-singleton-Blue* [*simp*]: {*a*} $\notin$ *Blue*
using *Blue-E* **by** *auto*

lemma *no-singleton-Red* [*simp*]: {*a*} $\notin$ *Red*
using *Red-E* **by** *auto*

lemma *not-Red-Neighbour* [*simp*]: $x \notin$ *Neighbours Red x* **and** *not-Blue-Neighbour* [*simp*]: $x \notin$ *Neighbours Blue x*
using *Red-E Blue-E not-own-Neighbour* **by** *auto*

lemma *Neighbours-RB*:
assumes $a \in V$ $X \subseteq V$
shows *Neighbours Red a* $\cap X \cup$ *Neighbours Blue a* $\cap X = X - \{a\}$
using *assms Red-Blue-all complete singleton-not-edge*
by (*fastforce simp*: *Neighbours-def*)

lemma *Neighbours-Red-Blue*:
  assumes $x \in V$
  shows *Neighbours Red x* = *V* − *insert x* (*Neighbours Blue x*)
  using *Red-E assms* **by** (*auto simp*: *Blue-eq Neighbours-def complete all-edges-def*)

abbreviation *red-density X Y* ≡ *gen-density Red X Y*
abbreviation *blue-density X Y* ≡ *gen-density Blue X Y*

definition *Weight* :: ['*a set*, '*a set*, '*a*, '*a*] ⇒ *real* **where**
  Weight ≡ λ*X Y x y*. *inverse* (*card Y*) ∗ (*card* (*Neighbours Red x* ∩ *Neighbours Red y* ∩ *Y*)
          − *red-density X Y* ∗ *card* (*Neighbours Red x* ∩ *Y*))

definition *weight* :: '*a set* ⇒ '*a set* ⇒ '*a* ⇒ *real* **where**
  weight ≡ λ*X Y x*. ∑ *y* ∈ *X*−{*x*}. *Weight X Y x y*

definition *p0* :: *real*
  where *p0* ≡ *red-density X0 Y0*

definition *qfun* :: *nat* ⇒ *real*
  where *qfun* ≡ λ*h*. *p0* + *qfun-base k h*

lemma *qfun-eq*: *qfun* ≡ λ*h*. *p0* + ((*1* + ε) ^*h* − *1*) / *k*
  by (*simp add*: *qfun-def qfun-base-def eps-def eps-def*)

definition *hgt* :: *real* ⇒ *nat*
  where *hgt* ≡ λ*p*. *LEAST h*. *p* ≤ *qfun h* ∧ *h*>*0*

lemma *qfun0* [*simp*]: *qfun 0* = *p0*
  by (*simp add*: *qfun-eq*)

lemma *p0-ge*: *p0* ≥ *p0-min*
  using *density-ge-p0-min* **by** (*simp add*: *p0-def*)

lemma *card-XY0*: *card X0* > *0 card Y0* > *0*
  using *Red-edges-XY0 finite-X0 finite-Y0* **by** *force*+

lemma *finite-Red* [*simp*]: *finite Red*
  by (*metis Red-Blue-all complete fin-edges finite-Un*)

lemma *finite-Blue* [*simp*]: *finite Blue*
  using *Blue-E fin-edges finite-subset* **by** *blast*

lemma *Red-edges-nonzero*: *edge-card Red X0 Y0* > *0*
  using *Red-edges-XY0*
  using *Red-E edge-card-def fin-edges finite-subset* **by** *fastforce*

lemma *p0-01*: *0* < *p0 p0* ≤ *1*

proof −
  show *0* < *p0*
    using *Red-edges-nonzero card-XY0*
    by (*auto simp*: *p0-def gen-density-def divide-simps mult-less-0-iff*)
  show *p0* ≤ *1*
    by (*simp add*: *gen-density-le1 p0-def*)
qed

lemma *qfun-strict-mono*: *h'*<*h* ⟹ *qfun h'* < *qfun h*
  by (*simp add*: *divide-strict-right-mono eps-gt0 kn0 qfun-eq*)

lemma *qfun-mono*: *h'*≤*h* ⟹ *qfun h'* ≤ *qfun h*
  by (*metis less-eq-real-def nat-less-le qfun-strict-mono*)

lemma *q-Suc-diff*: *qfun* (*Suc h*) − *qfun h* = ε ∗ (*1* + ε)^*h* / *k*
  by (*simp add*: *qfun-eq field-split-simps*)

lemma *height-exists'*:
  obtains *h* **where** *p* ≤ *qfun-base k h* ∧ *h*>*0*
proof −
  have *1*: *1* + ε ≥ *1*
    by (*auto simp*: *eps-def*)
  have ∀∞*h*. *p* ≤ *real h* ∗ ε / *real k*
    using *p0-01 kn0* **unfolding** *eps-def* **by** *real-asymp*
  then obtain *h* **where** *p* ≤ *real h* ∗ ε / *real k*
    by (*meson eventually-sequentially order.refl*)
  also have ... ≤ ((*1* + ε) ^ *h* − *1*) / *real k*
    using *linear-plus-1-le-power* [*of* ε *h*]
    by (*intro divide-right-mono add-mono*) (*auto simp*: *eps-def add-ac*)
  also have ... ≤ ((*1* + ε) ^ *Suc h* − *1*) / *real k*
    using *power-increasing* [*OF le-SucI* [*OF order-refl*] *1*]
    by (*simp add*: *divide-right-mono*)
  finally have *p* ≤ *qfun-base k* (*Suc h*)
    unfolding *qfun-base-def eps-def eps-def* **using** *p0-01* **by** *blast*
  then show *thesis*
    using *that* **by** *blast*
qed

lemma *height-exists*:
  obtains *h* **where** *p* ≤ *qfun h h*>*0*
proof −
  obtain *h'* **where** *p* ≤ *qfun-base k h'* ∧ *h'*>*0*
    using *height-exists'* **by** *blast*
  then show *thesis*
    using *p0-01 qfun-def that*
    by (*metis add-strict-increasing less-eq-real-def*)
qed

lemma *hgt-gt0*: *hgt p* $>$ *0*
  unfolding *hgt-def*
  by (*smt* (*verit*, *best*) *LeastI height-exists kn0*)

lemma *hgt-works*: *p* $\leq$ *qfun* (*hgt p*)
  by (*metis* (*no-types*, *lifting*) *LeastI height-exists hgt-def*)

lemma *hgt-Least*:
  assumes *0*$<$*h p* $\leq$ *qfun h*
  shows *hgt p* $\leq$ *h*
  by (*simp add*: *Suc-leI assms hgt-def Least-le*)

lemma *real-hgt-Least*:
  assumes *real h* $\leq$ *r 0*$<$*h p* $\leq$ *qfun h*
  shows *real* (*hgt p*) $\leq$ *r*
  using *assms* **by** (*meson assms order.trans hgt-Least of-nat-mono*)

lemma *hgt-greater*:
  assumes *p* $>$ *qfun h*
  shows *hgt p* $>$ *h*
  by (*meson assms hgt-works kn0 not-less order.trans qfun-mono*)

lemma *hgt-less-imp-qfun-less*:
  assumes *0*$<$*h h* $<$ *hgt p*
  shows *p* $>$ *qfun h*
  by (*metis assms hgt-Least not-le*)

lemma *hgt-le-imp-qfun-ge*:
  assumes *hgt p* $\leq$ *h*
  shows *p* $\leq$ *qfun h*
  by (*meson assms hgt-greater not-less*)

This gives us an upper bound for heights, namely *hgt 1*, but it's not explicit.

lemma *hgt-mono*:
  assumes *p* $\leq$ *q*
  shows *hgt p* $\leq$ *hgt q*
  by (*meson assms order.trans hgt-Least hgt-gt0 hgt-works*)

lemma *hgt-mono'*:
  assumes *hgt p* $<$ *hgt q*
  shows *p* $<$ *q*
  by (*smt* (*verit*) *assms hgt-mono leD*)

The upper bound of the height $h(p)$ appears just below (5) on page 9. Although we can bound all Heights by monotonicity (since $p \leq 1$), we need to exhibit a specific $o(k)$ function.

lemma *height-upper-bound*:
  assumes *p* $\leq$ *1* **and** *big*: *Big-height-upper-bound k*

shows *hgt p* ≤ *2* ∗ *ln k* / ε
using *assms real-hgt-Least big nat-floor-neg not-gr0 of-nat-floor*
unfolding *Big-height-upper-bound-def hgt-maximum-def*
by (*smt* (*verit*) *eps-def hgt-Least of-nat-mono p0-01*(*1*) *qfun0 qfun-def*)

definition *alpha* :: *nat* ⇒ *real* **where** *alpha* ≡ λ*h*. *qfun h* − *qfun* (*h*−*1*)

lemma *alpha-ge0*: *alpha h* ≥ *0*
by (*simp add*: *alpha-def qfun-eq divide-le-cancel eps-gt0*)

lemma *alpha-Suc-ge*: *alpha* (*Suc h*) ≥ ε / *k*
proof −
have (*1* + ε) ^ *h* ≥ *1*
by (*simp add*: *eps-def*)
then show *?thesis*
by (*simp add*: *alpha-def qfun-eq eps-gt0 field-split-simps*)
qed

lemma *alpha-ge*: *h*>*0* ⟹ *alpha h* ≥ ε / *k*
by (*metis Suc-pred alpha-Suc-ge*)

lemma *alpha-gt0*: *h*>*0* ⟹ *alpha h* > *0*
by (*metis alpha-ge alpha-ge0 eps-gt0 kn0 nle-le not-le of-nat-0-less-iff zero-less-divide-iff*)

lemma *alpha-Suc-eq*: *alpha* (*Suc h*) = ε ∗ (*1* + ε) ^ *h* / *k*
by (*simp add*: *alpha-def q-Suc-diff*)

lemma *alpha-eq*:
assumes *h*>*0* **shows** *alpha h* = ε ∗ (*1* + ε) ^ (*h*−*1*) / *k*
by (*metis Suc-pred′ alpha-Suc-eq assms*)

lemma *alpha-hgt-eq*: *alpha* (*hgt p*) = ε ∗ (*1* + ε) ^ (*hgt p* −*1*) / *k*
using *alpha-eq hgt-gt0* **by** *presburger*

lemma *alpha-mono*: ⟦*h′* ≤ *h*; *0* < *h*⟧ ⟹ *alpha h′* ≤ *alpha h*
by (*simp add*: *alpha-eq eps-ge0 divide-right-mono mult-left-mono power-increasing*)

definition *all-incident-edges* :: *′a set* ⇒ *′a set set* **where**
all-incident-edges ≡ λ*A*. ⋃*v*∈*A*. *incident-edges v*

lemma *all-incident-edges-Un* [*simp*]: *all-incident-edges* (*A*∪*B*) = *all-incident-edges* *A* ∪ *all-incident-edges B*
by (*auto simp*: *all-incident-edges-def*)

end

context *Book*
begin

2.2 State invariants

definition *V-state* ≡ λ(*X*, *Y*, *A*, *B*). *X* ⊆ *V* ∧ *Y* ⊆ *V* ∧ *A* ⊆ *V* ∧ *B* ⊆ *V*

definition *disjoint-state* ≡ λ(*X*, *Y*, *A*, *B*). *disjnt X Y* ∧ *disjnt X A* ∧ *disjnt X B* ∧ *disjnt Y A* ∧ *disjnt Y B* ∧ *disjnt A B*

previously had all edges incident to A, B

definition *RB-state* ≡ λ(*X*, *Y*, *A*, *B*). *all-edges-betw-un A A* ⊆ *Red* ∧ *all-edges-betw-un A* (*X* ∪ *Y*) ⊆ *Red*
∧ *all-edges-betw-un B* (*B* ∪ *X*) ⊆ *Blue*

definition *valid-state* ≡ λ*U*. *V-state U* ∧ *disjoint-state U* ∧ *RB-state U*

definition *termination-condition* ≡ λ*X Y*. *card X* ≤ *RN k* (*nat* ⌈*real l powr* (*3*/*4*)⌉) ∨ *red-density X Y* ≤ *1*/*k*

lemma
assumes *V-state*(*X*, *Y*, *A*, *B*)
shows *finX*: *finite X* **and** *finY*: *finite Y* **and** *finA*: *finite A* **and** *finB*: *finite B*
using *V-state-def assms finV finite-subset* **by** *auto*

lemma
assumes *valid-state*(*X*, *Y*, *A*, *B*)
shows *A-Red-clique*: *clique A Red* **and** *B-Blue-clique*: *clique B Blue*
using *assms*
by (*auto simp*: *valid-state-def V-state-def RB-state-def all-edges-betw-un-iff-clique all-edges-betw-un-Un2*)

lemma *A-less-k*:
assumes *valid*: *valid-state*(*X*, *Y*, *A*, *B*)
shows *card A* < *k*
using *assms A-Red-clique*[*OF valid*] *no-Red-clique* **unfolding** *valid-state-def V-state-def*
by (*metis nat-neq-iff prod.case size-clique-def size-clique-smaller*)

lemma *B-less-l*:
assumes *valid*: *valid-state*(*X*, *Y*, *A*, *B*)
shows *card B* < *l*
using *assms B-Blue-clique*[*OF valid*] *no-Blue-clique* **unfolding** *valid-state-def V-state-def*
by (*metis nat-neq-iff prod.case size-clique-def size-clique-smaller*)

2.3 Degree regularisation

definition *red-dense* ≡ λ*Y p x*. *card* (*Neighbours Red x* ∩ *Y*) ≥ (*p* − ε *powr* (−*1*/*2*) ∗ *alpha* (*hgt p*)) ∗ *card Y*

definition *X-degree-reg* ≡ λ*X Y*. {*x* ∈ *X*. *red-dense Y* (*red-density X Y*) *x*}

definition *degree-reg* ≡ λ(*X*,*Y*,*A*,*B*). (*X-degree-reg X Y*, *Y*, *A*, *B*)

lemma *X-degree-reg-subset*: *X-degree-reg X Y* ⊆ *X*
  by (*auto simp*: *X-degree-reg-def*)

lemma *degree-reg-V-state*: *V-state U* ⟹ *V-state* (*degree-reg U*)
  by (*auto simp*: *degree-reg-def X-degree-reg-def V-state-def*)

lemma *degree-reg-disjoint-state*: *disjoint-state U* ⟹ *disjoint-state* (*degree-reg U*)
  by (*auto simp*: *degree-reg-def X-degree-reg-def disjoint-state-def disjnt-iff*)

lemma *degree-reg-RB-state*: *RB-state U* ⟹ *RB-state* (*degree-reg U*)
 apply (*simp add*: *degree-reg-def RB-state-def all-edges-betw-un-Un2 split*: *prod.split prod.split-asm*)
  by (*meson X-degree-reg-subset all-edges-betw-un-mono2 order.trans*)

lemma *degree-reg-valid-state*: *valid-state U* ⟹ *valid-state* (*degree-reg U*)
 by (*simp add*: *degree-reg-RB-state degree-reg-V-state degree-reg-disjoint-state valid-state-def*)

lemma *not-red-dense-sum-less*:
  assumes ⋀*x*. *x* ∈ *X* ⟹ ¬ *red-dense Y p x* **and** *X*≠{} *finite X*
  shows (∑*x*∈*X*. *card* (*Neighbours Red x* ∩ *Y*)) < *p* ∗ *real* (*card Y*) ∗ *card X*
proof −
  have ⋀*x*. *x* ∈ *X* ⟹ *card* (*Neighbours Red x* ∩ *Y*) < *p* ∗ *real* (*card Y*)
    using *assms*
    unfolding *red-dense-def*
   by (*smt* (*verit*) *alpha-ge0 mult-right-mono of-nat-0-le-iff powr-ge-zero zero-le-mult-iff*)
  with ‹*X*≠{}› **show** *?thesis*
    by (*smt* (*verit*) ‹*finite X*› *of-nat-sum sum-strict-mono mult-of-nat-commute sum-constant*)
qed

lemma *red-density-X-degree-reg-ge*:
  assumes *disjnt X Y*
  shows *red-density* (*X-degree-reg X Y*) *Y* ≥ *red-density X Y*
proof (*cases X*={} ∨ *infinite X* ∨ *infinite Y*)
  case *True*
  then show *?thesis*
    by (*force simp*: *gen-density-def X-degree-reg-def*)
next
  case *False*
  then have *finite X finite Y*
    by *auto*
  { assume ⋀*x*. *x* ∈ *X* ⟹ ¬ *red-dense Y* (*red-density X Y*) *x*
    with *False* **have** (∑*x*∈*X*. *card* (*Neighbours Red x* ∩ *Y*)) < *red-density X Y* ∗ *real* (*card Y*) ∗ *card X*
      using ‹*finite X*› *not-red-dense-sum-less* **by** *blast*
    with *Red-E* **have** *edge-card Red Y X* < (*red-density X Y* ∗ *real* (*card Y*)) ∗ *card X*

```
      by (metis False assms disjnt-sym edge-card-eq-sum-Neighbours)
    then have False
      by (simp add: gen-density-def edge-card-commute split: if-split-asm)
  }
  then obtain x where x: x ∈ X red-dense Y (red-density X Y) x
    by blast
  define X' where X' ≡ {x ∈ X. ¬ red-dense Y (red-density X Y) x}
  have X': finite X' disjnt Y X'
    using assms ‹finite X› by (auto simp: X'-def disjnt-iff)
  have eq: X-degree-reg X Y = X − X'
    by (auto simp: X-degree-reg-def X'-def)
  show ?thesis
  proof (cases X'={})
    case True
    then show ?thesis
      by (simp add: eq)
  next
    case False
    show ?thesis
      unfolding eq
    proof (rule gen-density-below-avg-ge)
     have (∑x∈X'. card (Neighbours Red x ∩ Y)) < red-density X Y ∗ real (card
Y) ∗ card X'
      proof (intro not-red-dense-sum-less)
        fix x
        assume x ∈ X'
        show ¬ red-dense Y (red-density X Y) x
          using ‹x ∈ X'› by (simp add: X'-def)
      qed (use False X' in auto)
       then have card X ∗ (∑x∈X'. card (Neighbours Red x ∩ Y)) < card X' ∗
edge-card Red Y X
       by (simp add: gen-density-def mult.commute divide-simps edge-card-commute
            flip: of-nat-sum of-nat-mult split: if-split-asm)
       then have card X ∗ (∑x∈X'. card (Neighbours Red x ∩ Y)) ≤ card X' ∗
(∑x∈X. card (Neighbours Red x ∩ Y))
        using assms Red-E
        by (metis ‹finite X› disjnt-sym edge-card-eq-sum-Neighbours nless-le)
      then have red-density Y X' ≤ red-density Y X
        using assms X' False ‹finite X›
      apply (simp add: gen-density-def edge-card-eq-sum-Neighbours disjnt-commute
Red-E)
        apply (simp add: X'-def field-split-simps flip: of-nat-sum of-nat-mult)
        done
      then show red-density X' Y ≤ red-density X Y
        by (simp add: X'-def gen-density-commute)
    qed (use assms x ‹finite X› ‹finite Y› X'-def in auto)
  qed
qed
```

2.4 Big blue steps: code

```
definition bluish :: ['a set,'a] ⇒ bool where
  bluish ≡ λX x. card (Neighbours Blue x ∩ X) ≥ μ * real (card X)

definition many-bluish :: 'a set ⇒ bool where
  many-bluish ≡ λX. card {x∈X. bluish X x} ≥ RN k (nat ⌈l powr (2/3)⌉)

definition good-blue-book :: ['a set, 'a set × 'a set] ⇒ bool where
  good-blue-book ≡ λX. λ(S,T). book S T Blue ∧ S⊆X ∧ T⊆X ∧ card T ≥ (μ ^
card S) * card X / 2

lemma ex-good-blue-book: good-blue-book X ({}, X)
  by (simp add: good-blue-book-def book-def)

lemma bounded-good-blue-book: ⟦good-blue-book X (S,T); finite X⟧ ⟹ card S ≤
card X
  by (simp add: card-mono finX good-blue-book-def)

definition best-blue-book-card :: 'a set ⇒ nat where
  best-blue-book-card ≡ λX. GREATEST s. ∃S T. good-blue-book X (S,T) ∧ s =
card S

lemma best-blue-book-is-best: ⟦good-blue-book X (S,T); finite X⟧ ⟹ card S ≤
best-blue-book-card X
  unfolding best-blue-book-card-def
  by (smt (verit) Greatest-le-nat bounded-good-blue-book)

lemma ex-best-blue-book: finite X ⟹ ∃S T. good-blue-book X (S,T) ∧ card S =
best-blue-book-card X
  unfolding best-blue-book-card-def
  by (smt (verit) GreatestI-ex-nat bounded-good-blue-book ex-good-blue-book)

definition choose-blue-book ≡ λ(X,Y,A,B). @(S,T). good-blue-book X (S,T) ∧
card S = best-blue-book-card X

lemma choose-blue-book-works:
  ⟦finite X; (S,T) = choose-blue-book (X,Y,A,B)⟧
  ⟹ good-blue-book X (S,T) ∧ card S = best-blue-book-card X
  unfolding choose-blue-book-def
  using someI-ex [OF ex-best-blue-book]
  by (metis (mono-tags, lifting) case-prod-conv someI-ex)

lemma choose-blue-book-subset:
  ⟦finite X; (S,T) = choose-blue-book (X,Y,A,B)⟧ ⟹ S ⊆ X ∧ T ⊆ X ∧ disjnt
S T
  using choose-blue-book-works good-blue-book-def book-def by fastforce
```

expressing the complicated preconditions inductively

```
inductive big-blue
```

where ⟦*many-bluish X*; *good-blue-book X* (*S*,*T*); *card S* = *best-blue-book-card X*⟧
⟹ *big-blue* (*X*,*Y*,*A*,*B*) (*T*, *Y*, *A*, *B*∪*S*)

lemma *big-blue-V-state*: ⟦*big-blue U U'*; *V-state U*⟧ ⟹ *V-state U'*
by (*force simp*: *good-blue-book-def V-state-def elim*!: *big-blue.cases*)

lemma *big-blue-disjoint-state*: ⟦*big-blue U U'*; *disjoint-state U*⟧ ⟹ *disjoint-state U'*
by (*force simp*: *book-def disjnt-iff good-blue-book-def disjoint-state-def elim*!: *big-blue.cases*)

lemma *big-blue-RB-state*: ⟦*big-blue U U'*; *RB-state U*⟧ ⟹ *RB-state U'*
apply (*clarsimp simp add*: *good-blue-book-def book-def RB-state-def all-edges-betw-un-Un1 all-edges-betw-un-Un2 elim*!: *big-blue.cases*)
by (*metis all-edges-betw-un-commute all-edges-betw-un-mono1 le-supI2 sup.orderE*)

lemma *big-blue-valid-state*: ⟦*big-blue U U'*; *valid-state U*⟧ ⟹ *valid-state U'*
by (*meson big-blue-RB-state big-blue-V-state big-blue-disjoint-state valid-state-def*)

2.5 The central vertex

definition *central-vertex* :: [*'a set*,*'a*] ⇒ *bool* **where**
central-vertex ≡ λ*X x*. *x* ∈ *X* ∧ *card* (*Neighbours Blue x* ∩ *X*) ≤ μ ∗ *real* (*card X*)

lemma *ex-central-vertex*:
assumes ¬ *termination-condition X Y* ¬ *many-bluish X*
shows ∃ *x*. *central-vertex X x*
proof −
have *l* ≠ *0*
using *linorder-not-less assms* **unfolding** *many-bluish-def* **by** *force*
then have ∗: *real l powr* (*2*/*3*) ≤ *real l powr* (*3*/*4*)
using *powr-mono* **by** *force*
then have *card* {*x* ∈ *X*. *bluish X x*} < *card X*
using *assms RN-mono*
unfolding *termination-condition-def many-bluish-def not-le*
by (*smt* (*verit*, *ccfv-SIG*) *linorder-not-le nat-ceiling-le-eq of-nat-le-iff*)
then obtain *x* **where** *x* ∈ *X* ¬ *bluish X x*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq nat-neq-iff subsetI subset-antisym*)
then show *?thesis*
by (*meson bluish-def central-vertex-def linorder-linear*)
qed

lemma *finite-central-vertex-set*: *finite X* ⟹ *finite* {*x*. *central-vertex X x*}
by (*simp add*: *central-vertex-def*)

definition *max-central-vx* :: [*'a set*,*'a set*] ⇒ *real* **where**
max-central-vx ≡ λ*X Y*. *Max* (*weight X Y* ` {*x*. *central-vertex X x*})

lemma *central-vx-is-best*:

$[\![$*central-vertex X x*; *finite X*$]\!] \Longrightarrow$ *weight X Y x* $\leq$ *max-central-vx X Y*
unfolding *max-central-vx-def* **by** (*simp add*: *finite-central-vertex-set*)

lemma *ex-best-central-vx*:
$[\![\neg$ *termination-condition X Y*; $\neg$ *many-bluish X*; *finite X*$]\!]$
$\Longrightarrow \exists\, x$. *central-vertex X x* $\wedge$ *weight X Y x* = *max-central-vx X Y*
unfolding *max-central-vx-def*
by (*metis empty-iff ex-central-vertex finite-central-vertex-set mem-Collect-eq obtains-MAX*)

it's necessary to make a specific choice; a relational treatment might allow different vertices to be chosen, making a nonsense of the choice between steps 4 and 5

definition *choose-central-vx* $\equiv \lambda(X,Y,A,B)$. @*x*. *central-vertex X x* $\wedge$ *weight X Y x* = *max-central-vx X Y*

lemma *choose-central-vx-works*:
$[\![\neg$ *termination-condition X Y*; $\neg$ *many-bluish X*; *finite X*$]\!]$
$\Longrightarrow$ *central-vertex X* (*choose-central-vx* (X,Y,A,B)) $\wedge$ *weight X Y* (*choose-central-vx* (X,Y,A,B)) = *max-central-vx X Y*
unfolding *choose-central-vx-def*
using *someI-ex* [*OF ex-best-central-vx*] **by** *force*

lemma *choose-central-vx-X*:
$[\![\neg$ *many-bluish X*; $\neg$ *termination-condition X Y*; *finite X*$]\!] \Longrightarrow$ *choose-central-vx* $(X,Y,A,B) \in X$
using *central-vertex-def choose-central-vx-works* **by** *fastforce*

2.6 Red step

definition *reddish* $\equiv \lambda k\ X\ Y\ p\ x$. *red-density* (*Neighbours Red x* $\cap X$) (*Neighbours Red x* $\cap Y$) $\geq p -$ *alpha* (*hgt p*)

inductive *red-step*
where $[\![$*reddish k X Y* (*red-density X Y*) *x*; *x* = *choose-central-vx* $(X,Y,A,B)]\!]$
$\Longrightarrow$ *red-step* (X,Y,A,B) (*Neighbours Red x* $\cap X$, *Neighbours Red x* $\cap Y$, *insert x A*, *B*)

lemma *red-step-V-state*:
assumes *red-step* (X,Y,A,B) U' $\neg$ *termination-condition X Y*
$\neg$ *many-bluish X* *V-state* (X,Y,A,B)
shows *V-state* U'
proof −
have $X \subseteq V$
using *assms* **by** (*auto simp*: *V-state-def*)
then have *choose-central-vx* $(X, Y, A, B) \in V$
using *assms choose-central-vx-X* **by** (*fastforce simp*: *finX*)
with *assms* **show** *?thesis*
by (*auto simp*: *V-state-def elim!*: *red-step.cases*)
qed

lemma *red-step-disjoint-state*:
  assumes *red-step* (X,Y,A,B) U' $\neg$ *termination-condition* X Y
    $\neg$ *many-bluish* X *V-state* (X,Y,A,B) *disjoint-state* (X,Y,A,B)
  shows *disjoint-state* U'
proof –
  have *choose-central-vx* $(X, Y, A, B) \in X$
    using *assms* **by** (*metis choose-central-vx-X finX*)
  with *assms* **show** *?thesis*
   by (*auto simp*: *disjoint-state-def disjnt-iff not-own-Neighbour elim*!: *red-step.cases*)
qed

lemma *red-step-RB-state*:
  assumes *red-step* (X,Y,A,B) U' $\neg$ *termination-condition* X Y
    $\neg$ *many-bluish* X *V-state* (X,Y,A,B) *RB-state* (X,Y,A,B)
  shows *RB-state* U'
proof –
  define x **where** $x \equiv$ *choose-central-vx* (X, Y, A, B)
  have [*simp*]: *finite* X
    using *assms* **by** (*simp add*: *finX*)
  have $x \in X$
    using *assms choose-central-vx-X* **by** (*metis* ‹*finite* X› *x-def*)
  have *A*: *all-edges-betw-un* (*insert* x A) (*insert* x A) $\subseteq$ *Red*
    if *all-edges-betw-un* A $A \subseteq$ *Red all-edges-betw-un* A $(X \cup Y) \subseteq$ *Red*
    using *that* ‹$x \in X$› *all-edges-betw-un-commute*
   by (*auto simp*: *all-edges-betw-un-insert2 all-edges-betw-un-Un2 intro*!: *all-uedges-betw-I*)
  have *B1*: *all-edges-betw-un* (*insert* x A) (*Neighbours Red* $x \cap X$) $\subseteq$ *Red*
    if *all-edges-betw-un* A $X \subseteq$ *Red*
    using *that* ‹$x \in X$› **by** (*force simp*: *all-edges-betw-un-def in-Neighbours-iff*)
  have *B2*: *all-edges-betw-un* (*insert* x A) (*Neighbours Red* $x \cap Y$) $\subseteq$ *Red*
    if *all-edges-betw-un* A $Y \subseteq$ *Red*
    using *that* ‹$x \in X$› **by** (*force simp*: *all-edges-betw-un-def in-Neighbours-iff*)
  from *assms A B1 B2* **show** *?thesis*
    apply (*clarsimp simp*: *RB-state-def simp flip*: *x-def elim*!: *red-step.cases*)
    by (*metis Int-Un-eq*(*2*) *Un-subset-iff all-edges-betw-un-Un2*)
qed

lemma *red-step-valid-state*:
  assumes *red-step* (X,Y,A,B) U' $\neg$ *termination-condition* X Y
    $\neg$ *many-bluish* X *valid-state* (X,Y,A,B)
  shows *valid-state* U'
 by (*meson assms red-step-RB-state red-step-V-state red-step-disjoint-state valid-state-def*)

2.7 Density-boost step

inductive *density-boost*
  where $[\![\neg$ *reddish* k X Y (*red-density* X Y) x; $x =$ *choose-central-vx* $(X,Y,A,B)]\!]$

    $\Longrightarrow$ *density-boost* (X,Y,A,B) (*Neighbours Blue* $x \cap X$, *Neighbours Red* x

```
∩ Y, A, insert x B)

lemma density-boost-V-state:
  assumes density-boost (X,Y,A,B) U′ ¬ termination-condition X Y
        ¬ many-bluish X V-state (X,Y,A,B)
  shows V-state U′
proof –
  have X ⊆ V
    using assms by (auto simp: V-state-def)
  then have choose-central-vx (X, Y, A, B) ∈ V
    using assms choose-central-vx-X finX by fastforce
  with assms show ?thesis
    by (auto simp: V-state-def elim!: density-boost.cases)
qed

lemma density-boost-disjoint-state:
  assumes density-boost (X,Y,A,B) U′ ¬ termination-condition X Y
        ¬ many-bluish X V-state (X,Y,A,B) disjoint-state (X,Y,A,B)
  shows disjoint-state U′
proof –
  have X ⊆ V
    using assms by (auto simp: V-state-def)
  then have choose-central-vx (X, Y, A, B) ∈ X
    using assms by (metis choose-central-vx-X finX)
  with assms show ?thesis
   by (auto simp: disjoint-state-def disjnt-iff not-own-Neighbour elim!: density-boost.cases)
qed

lemma density-boost-RB-state:
 assumes density-boost (X,Y,A,B) U′ ¬ termination-condition X Y ¬ many-bluish
X V-state (X,Y,A,B)
    and rb: RB-state (X,Y,A,B)
  shows RB-state U′
proof –
  define x where x ≡ choose-central-vx (X, Y, A, B)
  have x ∈ X
    using assms by (metis choose-central-vx-X finX x-def)
  have all-edges-betw-un A (Neighbours Blue x ∩ X ∪ Neighbours Red x ∩ Y) ⊆
Red
    if all-edges-betw-un A (X ∪ Y) ⊆ Red
    using that by (metis Int-Un-eq(4) Un-subset-iff all-edges-betw-un-Un2)
  moreover
  have all-edges-betw-un (insert x B) (insert x B) ⊆ Blue
    if all-edges-betw-un B (B ∪ X) ⊆ Blue
    using that ‹x ∈ X› by (auto simp: subset-iff set-eq-iff all-edges-betw-un-def)
  moreover
  have all-edges-betw-un (insert x B) (Neighbours Blue x ∩ X) ⊆ Blue
    if all-edges-betw-un B (B ∪ X) ⊆ Blue
   using ‹x ∈ X› that by (auto simp: all-edges-betw-un-def subset-iff in-Neighbours-iff)
```

ultimately show *?thesis*
using *assms*
by (*auto simp*: *RB-state-def all-edges-betw-un-Un2 x-def* [*symmetric*] *elim*!: *density-boost.cases*)
qed

lemma *density-boost-valid-state*:
assumes *density-boost* (*X*,*Y*,*A*,*B*) *U'* ¬ *termination-condition X Y* ¬ *many-bluish X valid-state* (*X*,*Y*,*A*,*B*)
shows *valid-state U'*
by (*meson assms density-boost-RB-state density-boost-V-state density-boost-disjoint-state valid-state-def*)

2.8 Execution steps 2–5 as a function

definition *next-state* :: *'a config* ⇒ *'a config* **where**
next-state ≡ λ(*X*,*Y*,*A*,*B*).
if many-bluish X
then let (*S*,*T*) = *choose-blue-book* (*X*,*Y*,*A*,*B*) *in* (*T*, *Y*, *A*, *B*∪*S*)
else let x = *choose-central-vx* (*X*,*Y*,*A*,*B*) *in*
if reddish k X Y (*red-density X Y*) *x*
then (*Neighbours Red x* ∩ *X*, *Neighbours Red x* ∩ *Y*, *insert x A*, *B*)
else (*Neighbours Blue x* ∩ *X*, *Neighbours Red x* ∩ *Y*, *A*, *insert x B*)

lemma *next-state-valid*:
assumes *valid-state* (*X*,*Y*,*A*,*B*) ¬ *termination-condition X Y*
shows *valid-state* (*next-state* (*X*,*Y*,*A*,*B*))
proof (*cases many-bluish X*)
case *True*
with *finX* **have** *big-blue* (*X*,*Y*,*A*,*B*) (*next-state* (*X*,*Y*,*A*,*B*))
apply (*simp add*: *next-state-def split*: *prod.split*)
by (*metis assms*(*1*) *big-blue.intros choose-blue-book-works valid-state-def*)
then show *?thesis*
using *assms big-blue-valid-state* **by** *blast*
next
case *non-bluish*: *False*
define *x* **where** *x* = *choose-central-vx* (*X*,*Y*,*A*,*B*)
show *?thesis*
proof (*cases reddish k X Y* (*red-density X Y*) *x*)
case *True*
with *non-bluish* **have** *red-step* (*X*,*Y*,*A*,*B*) (*next-state* (*X*,*Y*,*A*,*B*))
by (*simp add*: *next-state-def Let-def x-def red-step.intros split*: *prod.split*)
then show *?thesis*
using *assms non-bluish red-step-valid-state* **by** *blast*
next
case *False*
with *non-bluish* **have** *density-boost* (*X*,*Y*,*A*,*B*) (*next-state* (*X*,*Y*,*A*,*B*))
by (*simp add*: *next-state-def Let-def x-def density-boost.intros split*: *prod.split*)
then show *?thesis*

```
    using assms density-boost-valid-state non-bluish by blast
  qed
qed

primrec stepper :: nat ⇒ 'a config where
  stepper 0 = (X0,Y0,{},{})
| stepper (Suc n) =
     (let (X,Y,A,B) = stepper n in
      if termination-condition X Y then (X,Y,A,B)
      else if even n then degree-reg (X,Y,A,B) else next-state (X,Y,A,B))

lemma degree-reg-subset:
  assumes degree-reg (X,Y,A,B) = (X',Y',A',B')
  shows X' ⊆ X ∧ Y' ⊆ Y
  using assms by (auto simp: degree-reg-def X-degree-reg-def)

lemma next-state-subset:
  assumes next-state (X,Y,A,B) = (X',Y',A',B') finite X
  shows X' ⊆ X ∧ Y' ⊆ Y
  using assms choose-blue-book-subset
   apply (clarsimp simp: next-state-def valid-state-def Let-def split: if-split-asm
prod.split-asm)
  by (smt (verit) choose-blue-book-subset subset-eq)

lemma valid-state0: valid-state (X0, Y0, {}, {})
 using XY0 by (simp add: valid-state-def V-state-def disjoint-state-def RB-state-def)

lemma valid-state-stepper [simp]: valid-state (stepper n)
proof (induction n)
  case 0
  then show ?case
    by (simp add: stepper-def valid-state0)
next
  case (Suc n)
  then show ?case
    by (force simp: next-state-valid degree-reg-valid-state split: prod.split)
qed

lemma V-state-stepper: V-state (stepper n)
  using valid-state-def valid-state-stepper by force

lemma RB-state-stepper: RB-state (stepper n)
  using valid-state-def valid-state-stepper by force

lemma
  assumes stepper n = (X,Y,A,B)
  shows stepper-A: clique A Red ∧ A⊆V and stepper-B: clique B Blue ∧ B⊆V
proof −
  have A⊆V B⊆V
```

using *V-state-stepper*[*of n*] *assms* **by** (*auto simp*: *V-state-def*)
moreover
have *all-edges-betw-un A A* ⊆ *Red all-edges-betw-un B B* ⊆ *Blue*
using *RB-state-stepper*[*of n*] *assms* **by** (*auto simp*: *RB-state-def all-edges-betw-un-Un2*)
ultimately show *clique A Red* ∧ *A*⊆*V clique B Blue* ∧ *B*⊆*V*
using *all-edges-betw-un-iff-clique* **by** *auto*
qed

lemma *card-B-limit*:
assumes *stepper n* = (*X*, *Y*,*A*,*B*) **shows** *card B* < *l*
by (*metis B-less-l assms valid-state-stepper*)

definition *Xseq* ≡ (λ(*X*, *Y*,*A*,*B*). *X*) ∘ *stepper*
definition *Yseq* ≡ (λ(*X*, *Y*,*A*,*B*). *Y*) ∘ *stepper*
definition *Aseq* ≡ (λ(*X*, *Y*,*A*,*B*). *A*) ∘ *stepper*
definition *Bseq* ≡ (λ(*X*, *Y*,*A*,*B*). *B*) ∘ *stepper*
definition *pseq* ≡ λ*i*. *red-density* (*Xseq i*) (*Yseq i*)

lemma *Xseq-0* [*simp*]: *Xseq 0* = *X0*
by (*simp add*: *Xseq-def*)

lemma *Xseq-Suc-subset*: *Xseq* (*Suc i*) ⊆ *Xseq i* **and** *Yseq-Suc-subset*: *Yseq* (*Suc i*) ⊆ *Yseq i*
apply (*simp-all add*: *Xseq-def Yseq-def split*: *if-split-asm prod.split*)
by (*metis V-state-stepper degree-reg-subset finX next-state-subset*)+

lemma *Xseq-antimono*: *j* ≤ *i* ⟹ *Xseq i* ⊆ *Xseq j*
by (*simp add*: *Xseq-Suc-subset lift-Suc-antimono-le*)

lemma *Xseq-subset-V*: *Xseq i* ⊆ *V*
using *XY0 Xseq-0 Xseq-antimono* **by** *blast*

lemma *finite-Xseq*: *finite* (*Xseq i*)
by (*meson Xseq-subset-V finV finite-subset*)

lemma *Yseq-0* [*simp*]: *Yseq 0* = *Y0*
by (*simp add*: *Yseq-def*)

lemma *Yseq-antimono*: *j* ≤ *i* ⟹ *Yseq i* ⊆ *Yseq j*
by (*simp add*: *Yseq-Suc-subset lift-Suc-antimono-le*)

lemma *Yseq-subset-V*: *Yseq i* ⊆ *V*
using *XY0 Yseq-0 Yseq-antimono* **by** *blast*

lemma *finite-Yseq*: *finite* (*Yseq i*)
by (*meson Yseq-subset-V finV finite-subset*)

lemma *Xseq-Yseq-disjnt*: *disjnt* (*Xseq i*) (*Yseq i*)
by (*metis XY0*(*1*) *Xseq-0 Xseq-antimono Yseq-0 Yseq-antimono disjnt-subset1*

disjnt-sym zero-le)

lemma *edge-card-eq-pee*:
edge-card Red (*Xseq i*) (*Yseq i*) = *pseq i* ∗ *card* (*Xseq i*) ∗ *card* (*Yseq i*)
by (*simp add*: *pseq-def gen-density-def finite-Xseq finite-Yseq*)

lemma *valid-state-seq*: *valid-state*(*Xseq i*, *Yseq i*, *Aseq i*, *Bseq i*)
using *valid-state-stepper*[*of i*]
by (*force simp*: *Xseq-def Yseq-def Aseq-def Bseq-def simp del*: *valid-state-stepper split*: *prod.split*)

lemma *Aseq-less-k*: *card* (*Aseq i*) < *k*
by (*meson A-less-k valid-state-seq*)

lemma *Aseq-0* [*simp*]: *Aseq 0* = {}
by (*simp add*: *Aseq-def*)

lemma *Aseq-Suc-subset*: *Aseq i* ⊆ *Aseq* (*Suc i*) **and** *Bseq-Suc-subset*: *Bseq i* ⊆ *Bseq* (*Suc i*)
by (*auto simp*: *Aseq-def Bseq-def next-state-def degree-reg-def Let-def split*: *prod.split*)

lemma
assumes *j* ≤ *i*
shows *Aseq-mono*: *Aseq j* ⊆ *Aseq i* **and** *Bseq-mono*: *Bseq j* ⊆ *Bseq i*
using *assms* **by** (*auto simp*: *Aseq-Suc-subset Bseq-Suc-subset lift-Suc-mono-le*)

lemma *Aseq-subset-V*: *Aseq i* ⊆ *V*
using *stepper-A*[*of i*] **by** (*simp add*: *Aseq-def split*: *prod.split*)

lemma *Bseq-subset-V*: *Bseq i* ⊆ *V*
using *stepper-B*[*of i*] **by** (*simp add*: *Bseq-def split*: *prod.split*)

lemma *finite-Aseq*: *finite* (*Aseq i*) **and** *finite-Bseq*: *finite* (*Bseq i*)
by (*meson Aseq-subset-V Bseq-subset-V finV finite-subset*)+

lemma *Bseq-less-l*: *card* (*Bseq i*) < *l*
by (*meson B-less-l valid-state-seq*)

lemma *Bseq-0* [*simp*]: *Bseq 0* = {}
by (*simp add*: *Bseq-def*)

lemma *pee-eq-p0*: *pseq 0* = *p0*
by (*simp add*: *pseq-def p0-def*)

lemma *pee-ge0*: *pseq i* ≥ *0*
by (*simp add*: *gen-density-ge0 pseq-def*)

lemma *pee-le1*: *pseq i* ≤ *1*
using *gen-density-le1 pseq-def* **by** *presburger*

lemma *pseq-0*: *p0* = *pseq 0*
  by (*simp add*: *p0-def pseq-def Xseq-def Yseq-def*)

The central vertex at each step (though only defined in some cases), *x-i* in the paper

definition *cvx* ≡ λ*i*. *choose-central-vx* (*stepper i*)

the indexing of *beta* is as in the paper — and different from that of *Xseq*

definition
  beta ≡ λ*i*. *let* (*X*,*Y*,*A*,*B*) = *stepper i in card*(*Neighbours Blue* (*cvx i*) ∩ *X*) / *card X*

lemma *beta-eq*: *beta i* = *card*(*Neighbours Blue* (*cvx i*) ∩ *Xseq i*) / *card* (*Xseq i*)
  by (*simp add*: *beta-def cvx-def Xseq-def split*: *prod.split*)

lemma *beta-ge0*: *beta i* ≥ *0*
  by (*simp add*: *beta-eq*)

2.9 The classes of execution steps

For R, B, S, D

datatype *stepkind* = *red-step* | *bblue-step* | *dboost-step* | *dreg-step* | *halted*

definition *next-state-kind* :: *'a config* ⇒ *stepkind* **where**
  next-state-kind ≡ λ(*X*,*Y*,*A*,*B*).
    if many-bluish X then bblue-step
    else let x = *choose-central-vx* (*X*,*Y*,*A*,*B*) *in*
      if reddish k X Y (*red-density X Y*) *x then red-step*
      else dboost-step

definition *stepper-kind* :: *nat* ⇒ *stepkind* **where**
  stepper-kind i =
    (*let* (*X*,*Y*,*A*,*B*) = *stepper i in*
     if termination-condition X Y then halted
     else if even i then dreg-step else next-state-kind (*X*,*Y*,*A*,*B*))

definition *Step-class* ≡ λ*knd*. {*n*. *stepper-kind n* ∈ *knd*}

lemma *subset-Step-class*: ⟦*i* ∈ *Step-class K'*; *K'* ⊆ *K*⟧ ⟹ *i* ∈ *Step-class K*
  by (*auto simp*: *Step-class-def*)

lemma *Step-class-Un*: *Step-class* (*K'* ∪ *K*) = *Step-class K'* ∪ *Step-class K*
  by (*auto simp*: *Step-class-def*)

lemma *Step-class-insert*: *Step-class* (*insert knd K*) = (*Step-class* {*knd*}) ∪ (*Step-class K*)
  by (*auto simp*: *Step-class-def*)

lemma *Step-class-insert-NO-MATCH*:
NO-MATCH {} *K* ⟹ *Step-class* (*insert knd K*) = (*Step-class* {*knd*}) ∪ (*Step-class* *K*)
by (*auto simp*: *Step-class-def*)

lemma *Step-class-UNIV*: *Step-class* {*red-step*,*bblue-step*,*dboost-step*,*dreg-step*,*halted*} = *UNIV*
using *Step-class-def stepkind.exhaust* **by** *auto*

lemma *Step-class-cases*:
i ∈ *Step-class* {*stepkind.red-step*} ∨ *i* ∈ *Step-class* {*bblue-step*} ∨
i ∈ *Step-class* {*dboost-step*} ∨ *i* ∈ *Step-class* {*dreg-step*} ∨
i ∈ *Step-class* {*halted*}
using *Step-class-def stepkind.exhaust* **by** *auto*

lemmas *step-kind-defs* = *Step-class-def stepper-kind-def next-state-kind-def*
Xseq-def Yseq-def Aseq-def Bseq-def cvx-def Let-def

lemma *disjnt-Step-class*:
disjnt knd knd' ⟹ *disjnt* (*Step-class knd*) (*Step-class knd'*)
by (*auto simp*: *Step-class-def disjnt-iff*)

lemma *halted-imp-next-halted*: *stepper-kind i* = *halted* ⟹ *stepper-kind* (*Suc i*) = *halted*
by (*auto simp*: *step-kind-defs split*: *prod.split if-split-asm*)

lemma *halted-imp-ge-halted*: *stepper-kind i* = *halted* ⟹ *stepper-kind* (*i*+*n*) = *halted*
by (*induction n*) (*auto simp*: *halted-imp-next-halted*)

lemma *Step-class-halted-forever*: ⟦*i* ∈ *Step-class* {*halted*}; *i*≤*j*⟧ ⟹ *j* ∈ *Step-class* {*halted*}
by (*simp add*: *Step-class-def*) (*metis halted-imp-ge-halted le-iff-add*)

lemma *Step-class-not-halted*: ⟦*i* ∉ *Step-class* {*halted*}; *i*≥*j*⟧ ⟹ *j* ∉ *Step-class* {*halted*}
using *Step-class-halted-forever* **by** *blast*

lemma
assumes *i* ∉ *Step-class* {*halted*}
shows *not-halted-pee-gt*: *pseq i* > *1*/*k*
and *Xseq-gt0*: *card* (*Xseq i*) > *0*
and *Xseq-gt-RN*: *card* (*Xseq i*) > *RN k* (*nat* ⌈*real l powr* (*3*/*4*)⌉)
and *not-termination-condition*: ¬ *termination-condition* (*Xseq i*) (*Yseq i*)
using *assms*
by (*auto simp*: *step-kind-defs termination-condition-def pseq-def split*: *if-split-asm prod.split-asm*)

lemma *not-halted-pee-gt0*:

assumes *i* ∉ *Step-class* {*halted*}
shows *pseq i* > *0*
using *not-halted-pee-gt* [*OF assms*] *linorder-not-le order-less-le-trans* **by** *fastforce*

lemma *Yseq-gt0*:
assumes *i* ∉ *Step-class* {*halted*}
shows *card* (*Yseq i*) > *0*
using *not-halted-pee-gt* [*OF assms*]
using *card-gt-0-iff finite-Yseq pseq-def* **by** *fastforce*

lemma *step-odd*: *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*} ⟹ *odd i*
by (*auto simp*: *Step-class-def stepper-kind-def split*: *if-split-asm prod.split-asm*)

lemma *step-even*: *i* ∈ *Step-class* {*dreg-step*} ⟹ *even i*
by (*auto simp*: *Step-class-def stepper-kind-def next-state-kind-def split*: *if-split-asm prod.split-asm*)

lemma *not-halted-odd-RBS*: ⟦*i* ∉ *Step-class* {*halted*}; *odd i*⟧ ⟹ *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
by (*auto simp*: *Step-class-def stepper-kind-def next-state-kind-def split*: *prod.split-asm*)

lemma *not-halted-even-dreg*: ⟦*i* ∉ *Step-class* {*halted*}; *even i*⟧ ⟹ *i* ∈ *Step-class* {*dreg-step*}
by (*auto simp*: *Step-class-def stepper-kind-def next-state-kind-def split*: *prod.split-asm*)

lemma *step-before-dreg*:
assumes *Suc i* ∈ *Step-class* {*dreg-step*}
shows *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
using *assms* **by** (*auto simp*: *step-kind-defs split*: *if-split-asm prod.split-asm*)

lemma *dreg-before-step*:
assumes *Suc i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
shows *i* ∈ *Step-class* {*dreg-step*}
using *assms* **by** (*auto simp*: *Step-class-def stepper-kind-def split*: *if-split-asm prod.split-asm*)

lemma
assumes *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
shows *dreg-before-step'*: *i* − *Suc 0* ∈ *Step-class* {*dreg-step*}
and *dreg-before-gt0*: *i*>*0*
proof −
show *i*>*0*
using *assms gr0I step-odd* **by** *force*
then show *i* − *Suc 0* ∈ *Step-class* {*dreg-step*}
using *assms dreg-before-step Suc-pred* **by** *force*
qed

lemma *dreg-before-step1*:
assumes *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}

shows *i−1* ∈ *Step-class* {*dreg-step*}
using *dreg-before-step′* [*OF assms*] **by** *auto*

lemma *step-odd-minus2*:
assumes *i* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*} *i*>*1*
shows *i−2* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
by (*metis Suc-1 Suc-diff-Suc assms dreg-before-step1 step-before-dreg*)

lemma *Step-class-iterates*:
assumes *finite* (*Step-class* {*knd*})
obtains *n* **where** *Step-class* {*knd*} = {*m*. *m*<*n* ∧ *stepper-kind m* = *knd*}
proof −
have *eq*: (*Step-class* {*knd*}) = (⋃*i*. {*m*. *m*<*i* ∧ *stepper-kind m* = *knd*})
by (*auto simp*: *Step-class-def*)
then obtain *n* **where** *n*: (*Step-class* {*knd*}) = (⋃*i*<*n*. {*m*. *m*<*i* ∧ *stepper-kind m* = *knd*})
using *finite-countable-equals*[*OF assms*] **by** *blast*
with *Step-class-def*
have {*m*. *m*<*n* ∧ *stepper-kind m* = *knd*} = (⋃*i*<*n*. {*m*. *m*<*i* ∧ *stepper-kind m* = *knd*})
by *auto*
then show *?thesis*
by (*metis n that*)
qed

lemma *step-non-terminating-iff*:
i ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*,*dreg-step*}
⟷ ¬ *termination-condition* (*Xseq i*) (*Yseq i*)
by (*auto simp*: *step-kind-defs split*: *if-split-asm prod.split-asm*)

lemma *step-terminating-iff*:
i ∈ *Step-class* {*halted*} ⟷ *termination-condition* (*Xseq i*) (*Yseq i*)
by (*auto simp*: *step-kind-defs split*: *if-split-asm prod.split-asm*)

lemma *not-many-bluish*:
assumes *i* ∈ *Step-class* {*red-step*,*dboost-step*}
shows ¬ *many-bluish* (*Xseq i*)
using *assms*
by (*simp add*: *step-kind-defs split*: *if-split-asm prod.split-asm*)

lemma *stepper-XYseq*: *stepper i* = (*X*,*Y*,*A*,*B*) ⟹ *X* = *Xseq i* ∧ *Y* = *Yseq i*
using *Xseq-def Yseq-def* **by** *fastforce*

lemma *cvx-works*:
assumes *i* ∈ *Step-class* {*red-step*,*dboost-step*}
shows *central-vertex* (*Xseq i*) (*cvx i*)
∧ *weight* (*Xseq i*) (*Yseq i*) (*cvx i*) = *max-central-vx* (*Xseq i*) (*Yseq i*)
proof −
have ¬ *termination-condition* (*Xseq i*) (*Yseq i*)

using *Step-class-def assms step-non-terminating-iff* **by** *fastforce*
then show *?thesis*
using *assms not-many-bluish*[*OF assms*]
apply (*simp add*: *Step-class-def Xseq-def cvx-def Yseq-def split*: *prod.split prod.split-asm*)
by (*metis V-state-stepper choose-central-vx-works finX*)
qed

lemma *cvx-in-Xseq*:
assumes $i \in$ *Step-class* {*red-step*,*dboost-step*}
shows *cvx i* $\in$ *Xseq i*
using *assms cvx-works*[*OF assms*]
by (*simp add*: *Xseq-def central-vertex-def cvx-def split*: *prod.split-asm*)

lemma *card-Xseq-pos*:
assumes $i \in$ *Step-class* {*red-step*,*dboost-step*}
shows *card* (*Xseq i*) > *0*
by (*metis assms card-0-eq cvx-in-Xseq empty-iff finite-Xseq gr0I*)

lemma *beta-le*:
assumes $i \in$ *Step-class* {*red-step*,*dboost-step*}
shows *beta i* $\le \mu$
using *assms cvx-works*[*OF assms*] μ*01*
by (*simp add*: *beta-def central-vertex-def Xseq-def divide-simps split*: *prod.split-asm*)

2.10 Termination proof

Each step decreases the size of X

lemma *ex-nonempty-blue-book*:
assumes *mb*: *many-bluish X*
shows $\exists x \in X$. *good-blue-book X* ({*x*}, *Neighbours Blue x* $\cap$ *X*)
proof −
have *RN k* (*nat* ⌈*real l powr* (*2* / *3*)⌉) > *0*
by (*metis kn0 ln0 RN-eq-0-iff gr0I of-nat-ceiling of-nat-eq-0-iff powr-nonneg-iff*)
then obtain *x* **where** *x*$\in$*X* **and** *x*: *bluish X x*
using *mb* **unfolding** *many-bluish-def*
by (*smt* (*verit*) *card-eq-0-iff empty-iff equalityI less-le-not-le mem-Collect-eq subset-iff*)
have *book* {*x*} (*Neighbours Blue x* $\cap$ *X*) *Blue*
by (*force simp*: *book-def all-edges-betw-un-def in-Neighbours-iff*)
with *x* **show** *?thesis*
by (*auto simp*: *bluish-def good-blue-book-def* ‹*x* $\in$ *X*›)
qed

lemma *choose-blue-book-psubset*:
assumes *many-bluish X* **and** *ST*: *choose-blue-book* (*X*,*Y*,*A*,*B*) = (*S*,*T*)
and *finite X*
shows *T* $\neq$ *X*
proof −

```
  obtain x where x∈X and x: good-blue-book X ({x}, Neighbours Blue x ∩ X)
    using ex-nonempty-blue-book assms by blast
  with ‹finite X› have best-blue-book-card X ≠ 0
    unfolding valid-state-def
   by (metis best-blue-book-is-best card.empty card-seteq empty-not-insert finite.intros
singleton-insert-inj-eq)
  then have S ≠ {}
    by (metis ‹finite X› ST choose-blue-book-works card.empty)
  with ‹finite X› ST show ?thesis
   by (metis (no-types, opaque-lifting) choose-blue-book-subset disjnt-iff empty-subsetI
equalityI subset-eq)
qed

lemma next-state-smaller:
  assumes next-state (X,Y,A,B) = (X',Y',A',B')
    and finite X and nont: ¬ termination-condition X Y
  shows X' ⊂ X
proof −
  have X' ⊆ X
    using assms next-state-subset by auto
  moreover have X' ≠ X
  proof −
    have *: ¬ X ⊆ Neighbours rb x ∩ X if x ∈ X rb ⊆ E for x rb
      using that by (auto simp: Neighbours-def subset-iff)
    show ?thesis
    proof (cases many-bluish X)
      case True
      with assms show ?thesis
        by (auto simp: next-state-def split: if-split-asm prod.split-asm
            dest!: choose-blue-book-psubset [OF True])
    next
      case False
      then have choose-central-vx (X,Y,A,B) ∈ X
        by (simp add: ‹finite X› choose-central-vx-X nont)
      with assms *[of - Red] *[of - Blue] ‹X' ⊆ X› Red-E Blue-E False
      choose-central-vx-X [OF False nont]
      show ?thesis
        by (fastforce simp: next-state-def Let-def split: if-split-asm prod.split-asm)
    qed
  qed
  ultimately show ?thesis
    by auto
qed

lemma do-next-state:
  assumes odd i ¬ termination-condition (Xseq i) (Yseq i)
  obtains A B A' B' where next-state (Xseq i, Yseq i, A, B)
                    = (Xseq (Suc i), Yseq (Suc i), A',B')
  using assms
```

by (*force simp*: *Xseq-def Yseq-def split*: *if-split-asm prod.split-asm prod.split*)

lemma *step-bound*:
assumes *i*: *Suc* (*2***i*) ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
shows *card* (*Xseq* (*Suc* (*2***i*))) + *i* ≤ *card X0*
using *i*
proof (*induction i*)
case *0*
then show *?case*
by (*metis Xseq-0 Xseq-Suc-subset add-0-right mult-0-right card-mono finite-X0*)
next
case (*Suc i*)
then have *nt*: ¬ *termination-condition* (*Xseq* (*Suc* (*2***i*))) (*Yseq* (*Suc* (*2***i*)))
unfolding *step-non-terminating-iff* [*symmetric*]
by (*metis Step-class-insert Suc-1 Un-iff dreg-before-step mult-Suc-right plus-1-eq-Suc plus-nat.simps*(*2*) *step-before-dreg*)
obtain *A B A′ B′* **where** *2*:
next-state (*Xseq* (*Suc* (*2***i*)), *Yseq* (*Suc* (*2***i*)), *A*, *B*) = (*Xseq* (*Suc* (*Suc* (*2***i*))), *Yseq* (*Suc* (*Suc* (*2***i*))), *A′*,*B′*)
by (*meson nt Suc-double-not-eq-double do-next-state evenE*)
have *Xseq* (*Suc* (*Suc* (*2***i*))) ⊂ *Xseq* (*Suc* (*2***i*))
by (*meson 2 finite-Xseq assms next-state-smaller nt*)
then have *card* (*Xseq* (*Suc* (*Suc* (*Suc* (*2***i*))))) < *card* (*Xseq* (*Suc* (*2***i*)))
by (*smt* (*verit*, *best*) *Xseq-Suc-subset card-seteq order.trans finite-Xseq leD not-le*)
moreover have *card* (*Xseq* (*Suc* (*2***i*))) + *i* ≤ *card X0*
using *Suc dreg-before-step step-before-dreg* **by** *force*
ultimately show *?case* **by** *auto*
qed

lemma *Step-class-halted-nonempty*: *Step-class* {*halted*} ≠ {}
proof −
define *i* **where** *i* ≡ *Suc* (*2* * *Suc* (*card X0*))
have *odd i*
by (*auto simp*: *i-def*)
then have *i* ∉ *Step-class* {*dreg-step*}
using *step-even* **by** *blast*
moreover have *i* ∉ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
unfolding *i-def* **using** *step-bound le-add2 not-less-eq-eq* **by** *blast*
ultimately show *?thesis*
using ‹*odd i*› *not-halted-odd-RBS* **by** *blast*
qed

definition *halted-point* ≡ *Inf* (*Step-class* {*halted*})

lemma *halted-point-halted*: *halted-point* ∈ *Step-class* {*halted*}
using *Step-class-halted-nonempty Inf-nat-def1*
by (*auto simp*: *halted-point-def*)

lemma *halted-point-minimal*:
  shows $i \notin$ *Step-class* {*halted*} $\longleftrightarrow$ $i <$ *halted-point*
  using *Step-class-halted-nonempty*
 by (*metis wellorder-Inf-le1 Inf-nat-def1 Step-class-not-halted halted-point-def less-le-not-le nle-le*)

lemma *halted-point-minimal'*: *stepper-kind i* $\neq$ *halted* $\longleftrightarrow$ $i <$ *halted-point*
  by (*simp add*: *Step-class-def flip*: *halted-point-minimal*)

lemma *halted-eq-Compl*:
  Step-class {*dreg-step*,*red-step*,*bblue-step*,*dboost-step*} = − *Step-class* {*halted*}
  using *Step-class-UNIV* [*of*] **by** (*auto simp*: *Step-class-def*)

lemma *before-halted-eq*:
  shows {..<*halted-point*} = *Step-class* {*dreg-step*,*red-step*,*bblue-step*,*dboost-step*}
  using *halted-point-minimal* **by** (*force simp*: *halted-eq-Compl*)

lemma *finite-components*:
  shows *finite* (*Step-class* {*dreg-step*,*red-step*,*bblue-step*,*dboost-step*})
  by (*metis before-halted-eq finite-lessThan*)

lemma
  shows *dreg-step-finite* [*simp*]: *finite* (*Step-class* {*dreg-step*})
    and *red-step-finite* [*simp*]: *finite* (*Step-class* {*red-step*})
    and *bblue-step-finite* [*simp*]: *finite* (*Step-class* {*bblue-step*})
    and *dboost-step-finite*[*simp*]: *finite* (*Step-class* {*dboost-step*})
  using *finite-components* **by** (*auto simp*: *Step-class-insert-NO-MATCH*)

lemma *halted-stepper-add-eq*: *stepper* (*halted-point* + *i*) = *stepper* (*halted-point*)
proof (*induction i*)
  case *0*
  then show *?case*
    by *auto*
next
  case (*Suc i*)
  have *hlt*: *stepper-kind* (*halted-point*) = *halted*
    using *Step-class-def halted-point-halted* **by** *force*
  obtain *X Y A B* **where** ∗: *stepper* (*halted-point*) = (*X*, *Y*, *A*, *B*)
    by (*metis surj-pair*)
  with *hlt* **have** *termination-condition X Y*
    by (*simp add*: *stepper-kind-def next-state-kind-def split*: *if-split-asm*)
  with ∗ **show** *?case*
    by (*simp add*: *Suc*)
qed

lemma *halted-stepper-eq*:
  assumes *i*: $i \geq$ *halted-point*
  shows *stepper i* = *stepper* (*halted-point*)
  using *μ01* **by** (*metis assms halted-stepper-add-eq le-iff-add*)

lemma *below-halted-point-cardX*:
  assumes $i < $ *halted-point*
  shows *card* (*Xseq i*) > 0
  using *Xseq-gt0 assms halted-point-minimal halted-stepper-eq* μ*01*
  by *blast*

end

sublocale *Book'* $\subseteq$ *Book* **where** $\mu=\gamma$
proof
  show $0 < \gamma$ $\gamma < 1$
    using *ln0 kn0* **by** (*auto simp*: γ*-def*)
qed (*use XY0 density-ge-p0-min* **in** *auto*)

lemma (**in** *Book*) *Book'*:
  assumes $\gamma =$ *real l* / (*real k* + *real l*)
  shows *Book' V E p0-min Red Blue l k* γ *X0 Y0*
proof qed (*use assms XY0 density-ge-p0-min* **in** *auto*)

end

3 Big Blue Steps: theorems

theory *Big-Blue-Steps* **imports** *Book*

begin

lemma *gbinomial-is-prod*: (*a gchoose k*) = ($\prod$ *i*<*k*. (*a* − *of-nat i*) / (*1* + *of-nat i*))
  unfolding *gbinomial-prod-rev*
  by (*induction k*; *simp add*: *divide-simps*)

3.1 Preliminaries

A bounded increasing sequence of finite sets eventually terminates

lemma *Union-incseq-finite*:
  assumes *fin*: $\bigwedge$*n*. *finite* (*A n*) **and** *N*: $\bigwedge$*n*. *card* (*A n*) < *N* **and** *incseq A*
  shows $\forall_F$ *k in sequentially*. $\bigcup$ (*range A*) = *A k*
proof (*rule ccontr*)
  assume ¬ *?thesis*
  then have $\forall$*k*. $\exists$*l*≥*k*. $\bigcup$ (*range A*) ≠ *A l*
    using *eventually-sequentially* **by** *force*
  then have $\forall$*k*. $\exists$*l*≥*k*. $\exists$*m*≥*l*. *A m* ≠ *A l*
    by (*smt* (*verit*, *ccfv-threshold*) ‹*incseq A*› *cSup-eq-maximum image-iff monotoneD nle-le rangeI*)
  then have $\forall$*k*. $\exists$*l*≥*k*. *A l* − *A k* ≠ {}

by (*metis* ‹*incseq A*› *diff-shunt-var monotoneD nat-le-linear subset-antisym*)
then obtain *f* **where** *f*: $\bigwedge k.\ f\ k \geq k \wedge A\ (f\ k) - A\ k \neq \{\}$
by *metis*
have *card* $(A\ ((f\hat{}\hat{}i)0)) \geq i$ **for** *i*
proof (*induction i*)
case *0*
then show *?case*
by *auto*
next
case (*Suc i*)
have *card* $(A\ ((f\ \hat{}\hat{}\ i)\ 0)) <$ *card* $(A\ (f\ ((f\ \hat{}\hat{}\ i)\ 0)))$
by (*metis Diff-cancel* ‹*incseq A*› *card-seteq f fin leI monotoneD*)
then show *?case*
using *Suc* **by** *simp*
qed
with *N* **show** *False*
using *linorder-not-less* **by** *auto*
qed

Two lemmas for proving "bigness lemmas" over a closed interval

lemma *eventually-all-geI0*:
assumes $\forall_F\ l$ *in sequentially*. $P\ a\ l$
$\bigwedge l\ x.\ [\![P\ a\ l;\ a{\leq}x;\ x{\leq}b;\ l \geq L]\!] \Longrightarrow P\ x\ l$
shows $\forall_F\ l$ *in sequentially*. $\forall x.\ a \leq x \wedge x \leq b \longrightarrow P\ x\ l$
by (*smt* (*verit*, *del-insts*) *assms eventually-sequentially eventually-elim2*)

lemma *eventually-all-geI1*:
assumes $\forall_F\ l$ *in sequentially*. $P\ b\ l$
$\bigwedge l\ x.\ [\![P\ b\ l;\ a{\leq}x;\ x{\leq}b;\ l \geq L]\!] \Longrightarrow P\ x\ l$
shows $\forall_F\ l$ *in sequentially*. $\forall x.\ a \leq x \wedge x \leq b \longrightarrow P\ x\ l$
by (*smt* (*verit*, *del-insts*) *assms eventually-sequentially eventually-elim2*)

Mehta's binomial function: convex on the entire real line and coinciding with gchoose under weak conditions

definition *mfact* $\equiv \lambda a\ k.$ *if* $a <$ *real* $k - 1$ *then 0 else prod* $(\lambda i.\ a -$ *of-nat* $i)$ $\{0..{<}k\}$

Mehta's special rule for convexity, my proof

lemma *convex-on-extend*:
fixes *f* :: *real* ⇒ *real*
assumes *cf*: *convex-on* {*k*..} *f* **and** *mon*: *mono-on* {*k*..} *f*
and *fk*: $\bigwedge x.\ x{<}k \Longrightarrow f\ x = f\ k$
shows *convex-on UNIV f*
proof (*intro convex-on-linorderI*)
fix *t x y* :: *real*
assume *t*: $0 < t$ $t < 1$ **and** $x < y$
let $?u = ((1 - t) *_R x + t *_R y)$
show $f\ ?u \leq (1 - t) * f\ x + t * f\ y$
proof (*cases* $k \leq x$)

```
    case True
    with ‹x < y› t show ?thesis
      by (intro convex-onD [OF cf]) auto
  next
    case False
    then have x < k and fxk: f x = f k by (auto simp: fk)
    show ?thesis
    proof (cases k ≤ y)
      case True
      then have f y ≥ f k
        using mon mono-onD by auto
      have kle: k ≤ (1 − t) ∗ k + t ∗ y
        using True segment-bound-lemma t by auto
      have fle: f ((1 − t) ∗R k + t ∗R y) ≤ (1 − t) ∗ f k + t ∗ f y
        using t True by (intro convex-onD [OF cf]) auto
      with False
      show ?thesis
      proof (cases ?u < k)
        case True
        then show ?thesis
          using ‹f k ≤ f y› fxk fk segment-bound-lemma t by auto
      next
        case False
        have f ?u ≤ f ((1 − t) ∗R k + t ∗R y)
          using kle ‹x < k› False t by (intro mono-onD [OF mon]) auto
        then show ?thesis
          using fle fxk by auto
      qed
    next
      case False
      with ‹x < k› show ?thesis
        by (simp add: fk convex-bound-lt order-less-imp-le segment-bound-lemma t)
    qed
  qed
qed auto

lemma convex-mfact:
  assumes k>0
  shows convex-on UNIV (λa. mfact a k)
  unfolding mfact-def
proof (rule convex-on-extend)
  show convex-on {real (k − 1)..} (λa. if a < real k − 1 then 0 else ∏ i = 0..<k.
a − real i)
    using convex-gchoose-aux [of k] assms
    apply (simp add: convex-on-def Ball-def)
    by (smt (verit, del-insts) distrib-right mult-cancel-right2 mult-left-mono)
  show mono-on {real (k − 1)..} (λa. if a < real k − 1 then 0 else ∏ i = 0..<k.
a − real i)
    using ‹k > 0› by (auto simp: mono-on-def intro!: prod-mono)
```

qed (*use assms gr0-conv-Suc* **in** *force*)

definition *mbinomial* :: *real* ⇒ *nat* ⇒ *real*
where *mbinomial* ≡ λ*a k*. *mfact a k* / *fact k*

lemma *convex-mbinomial*: *k*>*0* ⟹ *convex-on UNIV* (λ*x*. *mbinomial x k*)
by (*simp add*: *mbinomial-def convex-mfact convex-on-cdiv*)

lemma *mbinomial-eq-choose* [*simp*]: *mbinomial* (*real n*) *k* = *n choose k*
by (*simp add*: *binomial-gbinomial gbinomial-prod-rev mbinomial-def mfact-def*)

lemma *mbinomial-eq-gchoose* [*simp*]: *k* ≤ *a* ⟹ *mbinomial a k* = *a gchoose k*
by (*simp add*: *gbinomial-prod-rev mbinomial-def mfact-def*)

3.2 Preliminaries: Fact D1

from appendix D, page 55

lemma *Fact-D1-73-aux*:
fixes σ::*real* **and** *m b*::*nat*
assumes σ: *0*<σ **and** *bm*: *real b* < *real m*
shows ((σ∗*m*) *gchoose b*) ∗ *inverse* (*m gchoose b*) = σ ^*b* ∗ (∏ *i*<*b*. *1* − ((*1*−σ)∗*i*) / (σ ∗ (*real m* − *real i*)))
proof −
have ((σ∗*m*) *gchoose b*) ∗ *inverse* (*m gchoose b*) = (∏ *i*<*b*. (σ∗*m* − *i*) / (*real m* − *real i*))
using *bm* **by** (*simp add*: *gbinomial-prod-rev prod-dividef atLeast0LessThan*)
also have ... = σ ^*b* ∗ (∏ *i*<*b*. *1* − ((*1*−σ)∗*i*) / (σ ∗ (*real m* − *real i*)))
using *bm* σ **by** (*induction b*) (*auto simp*: *field-simps*)
finally show *?thesis* **.**
qed

This is fact 4.2 (page 11) as well as equation (73), page 55.

lemma *Fact-D1-73*:
fixes σ::*real* **and** *m b*::*nat*
assumes σ: *0*<σ σ≤*1* **and** *b*: *real b* ≤ σ ∗ *m* / *2*
shows (σ∗*m*) *gchoose b* ∈ {σ ^*b* ∗ (*real m gchoose b*) ∗ *exp* (− (*real b* ^ *2*) / (σ∗*m*)) .. σ ^*b* ∗ (*m gchoose b*)}
proof (*cases m*=*0* ∨ *b*=*0*)
case *True*
then show *?thesis*
using *True assms* **by** *auto*
next
case *False*
then have σ ∗ *m* / *2* < *real m*
using σ **by** *auto*
with *b* σ *False* **have** *bm*: *real b* < *real m*
by *linarith*
then have *nonz*: *m gchoose b* ≠ *0*
by (*simp add*: *flip*: *binomial-gbinomial*)

have *EQ*: $((\sigma*m)$ *gchoose* $b) * $ *inverse* $(m$ *gchoose* $b) = \sigma$ ^$b * (\prod i<b.\ 1 - ((1-\sigma)*i) / (\sigma * (real\ m - real\ i)))$
  using *Fact-D1-73-aux* ‹$0<\sigma$› *bm* **by** *blast*
also have ... $\leq \sigma$ ^ $b * 1$
proof (*intro mult-left-mono prod-le-1 conjI*)
  fix *i* **assume** $i \in \{..<b\}$
  with *b σ bm* **show** $0 \leq 1 - (1 - \sigma) * i / (\sigma * (real\ m - i))$
    by (*simp add*: *field-split-simps*)
qed (*use σ bm* **in** *auto*)
finally have *upper*: $(\sigma*m)$ *gchoose* $b \leq \sigma$ ^$b * (m$ *gchoose* $b)$
  using *nonz* **by** (*simp add*: *divide-simps flip*: *binomial-gbinomial*)
have $*$: $exp\ (-2 * real\ i / (\sigma*m)) \leq 1 - ((1-\sigma)*i) / (\sigma * (real\ m - real\ i))$ **if** $i<b$ **for** *i*
proof −
  have $i \leq m$
    using *bm that* **by** *linarith*
  have *exp-le*: $1-x \geq exp\ (-2 * x)$ **if** $0 \leq x\ x \leq 1/2$ **for** *x*::*real*
  proof −
    have $exp\ (-2 * x) \leq inverse\ (1 + 2*x)$
      using *exp-ge-add-one-self that* **by** (*simp add*: *exp-minus*)
    also have ... $\leq 1-x$
      using *that* **by** (*simp add*: *mult-left-le field-simps*)
    finally show *?thesis* **.**
  qed
  have $exp\ (-2 * real\ i / (\sigma*m)) = exp\ (-2 * (i / (\sigma*m)))$
    by *simp*
  also have ... $\leq 1 - i/(\sigma * m)$
  using *b that* **by** (*intro exp-le*) *auto*
  also have ... $\leq 1 - ((1-\sigma)*i) / (\sigma * (real\ m - real\ i))$
    using *σ b that* ‹$i \leq m$› **by** (*simp add*: *field-split-simps*)
  finally show *?thesis* **.**
qed
have $sum\ real\ \{..<b\} \leq real\ b$ ^ $2 / 2$
  by (*induction b*) (*auto simp*: *power2-eq-square algebra-simps*)
with *σ* **have** $exp\ (- (real\ b$ ^ $2) / (\sigma*m)) \leq exp\ (- (2 * (\sum i<b.\ i) / (\sigma*m)))$
  by (*simp add*: *mult-less-0-iff divide-simps*)
also have ... $= exp\ (\sum i<b.\ -2 * real\ i / (\sigma*m))$
  by (*simp add*: *sum-negf sum-distrib-left sum-divide-distrib*)
also have ... $= (\prod i<b.\ exp\ (-2 * real\ i / (\sigma*m)))$
  using *exp-sum* **by** *blast*
also have ... $\leq (\prod i<b.\ 1 - ((1-\sigma)*i) / (\sigma * (real\ m - real\ i)))$
  using $*$ **by** (*force intro*: *prod-mono*)
finally have $exp\ (- (real\ b)^2 / (\sigma * m)) \leq (\prod i<b.\ 1 - (1 - \sigma) * i / (\sigma * (real\ m - real\ i)))$ **.**
with *EQ* **have** σ ^$b * exp\ (- (real\ b$ ^ $2) / (\sigma*m)) \leq ((\sigma*m)$ *gchoose* $b) * inverse\ (real\ m$ *gchoose* $b)$
  by (*simp add*: *σ*)
with *σ bm* **have** *lower*: σ ^$b * (real\ m$ *gchoose* $b) * exp\ (- (real\ b$ ^ $2) / (\sigma*m)) \leq (\sigma*m)$ *gchoose* b

```
    by (simp add: field-split-simps flip: binomial-gbinomial)
  with upper show ?thesis
    by simp
qed
```

Exact at zero, so cannot be done using the approximation method

```
lemma exp-inequality-17:
  fixes x::real
  assumes 0 ≤ x x ≤ 1/7
  shows 1 − 4∗x/3 ≥ exp (−3∗x/2)
proof (cases x ≤ 1/12)
  case True
  have exp (−3∗x/2) ≤ 1/(1 + (3∗x)/2)
    using exp-ge-add-one-self [of 3∗x/2] assms
    by (simp add: exp-minus divide-simps)
  also have ... ≤ 1 − 4∗x/3
    using assms True mult-left-le [of x∗12] by (simp add: field-simps)
  finally show ?thesis .
next
  case False
  with assms have x ∈ {1/12..1/7}
    by auto
  then show ?thesis
    by (approximation 12 splitting: x=5)
qed
```

additional part

```
lemma Fact-D1-75:
  fixes σ::real and m b::nat
  assumes σ: 0<σ σ<1 and b: real b ≤ σ ∗ m / 2 and b': b ≤ m/7 and σ': σ
≥ 7/15
  shows (σ∗m) gchoose b ≥ exp (− (3 ∗ real b ^ 2) / (4∗m)) ∗ σ^b ∗ (m gchoose
b)
proof (cases m=0 ∨ b=0)
  case True
  then show ?thesis
    using True assms by auto
next
  case False
  with b b' σ have bm: real b < real m
    by linarith
  have ∗: exp (− 3 ∗ real i / (2∗m)) ≤ 1 − ((1−σ)∗i) / (σ ∗ (real m − real i))
if i<b for i
  proof −
    have im: 0 ≤ i/m i/m ≤ 1/7
      using b' that by auto
    have exp (− 3∗ real i / (2∗m)) ≤ 1 − 4∗i / (3∗m)
      using exp-inequality-17 [OF im] by (simp add: mult.commute)
    also have ... ≤ 1 − 8∗i / (7 ∗ (real m − real b))
```

proof −
have *real i* ∗ (*real b* ∗ *7*) ≤ *real i* ∗ *real m*
using *b′* **by** (*simp add*: *mult-left-mono*)
then show *?thesis*
using *b′* **by** (*simp add*: *field-split-simps*)
qed
also have ... ≤ *1* − ((*1*−σ)∗*i*) / (σ ∗ (*real m* − *real i*))
proof −
have *1*: (*1* − σ) / σ ≤ *8*/*7*
using σ σ′ *that*
by (*simp add*: *field-split-simps*)
have *2*: *1* / (*real m* − *real i*) ≤ *1* / (*real m* − *real b*)
using σ σ′ *b′* *that* **by** (*simp add*: *field-split-simps*)
have §: (*1* − σ) / (σ ∗ (*real m* − *real i*)) ≤ *8* / (*7* ∗ (*real m* − *real b*))
using *mult-mono* [*OF 1 2*] *b′ that* **by** *auto*
show *?thesis*
using *mult-left-mono* [*OF* §, *of i*]
by (*simp add*: *mult-of-nat-commute*)
qed
finally show *?thesis* **.**
qed
have *EQ*: ((σ∗*m*) *gchoose b*) ∗ *inverse* (*m gchoose b*) = σ ^*b* ∗ (∏*i*<*b*. *1* − ((*1*−σ)∗*i*) / (σ ∗ (*real m* − *real i*)))
using *Fact-D1-73-aux* ‹*0*<σ› *bm* **by** *blast*
have *sum real* {..<*b*} ≤ *real b* ^ *2* / *2*
by (*induction b*) (*auto simp*: *power2-eq-square algebra-simps*)
with σ **have** *exp* (− (*3* ∗ *real b* ^ *2*) / (*4*∗*m*)) ≤ *exp* (− (*3* ∗ (∑*i*<*b*. *i*) / (*2*∗*m*)))
by (*simp add*: *mult-less-0-iff divide-simps*)
also have ... = *exp* (∑*i*<*b*. −*3* ∗ *real i* / (*2*∗*m*))
by (*simp add*: *sum-negf sum-distrib-left sum-divide-distrib*)
also have ... = (∏*i*<*b*. *exp* (−*3* ∗ *real i* / (*2*∗*m*)))
using *exp-sum* **by** *blast*
also have ... ≤ (∏*i*<*b*. *1* − ((*1*−σ)∗*i*) / (σ ∗ (*real m* − *real i*)))
using ∗ **by** (*force intro*: *prod-mono*)
finally have *exp* (− (*3* ∗ *real b* ^ *2*) / (*4*∗*m*)) ≤ (∏*i*<*b*. *1* − (*1*−σ) ∗ *i* / (σ ∗ (*real m* − *real i*))) **.**
with *EQ* **have** σ ^*b* ∗ *exp* (− (*3* ∗ *real b* ^ *2*) / (*4*∗*m*)) ≤ ((σ∗*m*) *gchoose b*) / (*m gchoose b*)
by (*simp add*: *assms field-simps*)
with σ *bm* **show** *?thesis*
by (*simp add*: *field-split-simps flip*: *binomial-gbinomial*)
qed

lemma *power2-12*: *m* ≥ *12* ⟹ *25* ∗ m^2 ≤ *2*^*m*
proof (*induction m*)
case *0*
then show *?case* **by** *auto*
next

```
  case (Suc m)
  then consider m=11 | m≥12
    by linarith
  then show ?case
  proof cases
    case 1
    then show ?thesis
      by auto
  next
    case 2
    then have Suc(m+m) ≤ m*3 m≥3
      using Suc by auto
    then have 25 * Suc (m+m) ≤ 25 * (m*m)
      by (metis le-trans mult-le-mono2)
    with Suc show ?thesis
      by (auto simp: power2-eq-square algebra-simps 2)
  qed
qed
```

How b and m are obtained from l

```
definition b-of where b-of ≡ λl::nat. nat⌈l powr (1/4)⌉
definition m-of where m-of ≡ λl::nat. nat⌈l powr (2/3)⌉

definition Big-Blue-4-1 ≡
      λμ l. m-of l ≥ 12 ∧ l ≥ (6/μ) powr (12/5) ∧ l ≥ 15
              ∧ 1 ≤ 5/4 * exp (− real((b-of l)^2) / ((μ − 2/l) * m-of l)) ∧ μ > 2/l
               ∧ 2/l ≤ (μ − 2/l) * ((5/4) powr (1/b-of l) − 1)
```

Establishing the size requirements for 4.1. NOTE: it doesn't become clear until SECTION 9 that all bounds involving the parameter μ must hold for a RANGE of values

```
lemma Big-Blue-4-1:
  assumes 0<μ0
  shows ∀∞l. ∀μ. μ ∈ {μ0..μ1} ⟶ Big-Blue-4-1 μ l
proof −
  have 3: 3 / μ0 > 0
    using assms by force
  have 2: μ0 * nat ⌈3 / μ0⌉ > 2
    by (smt (verit, best) mult.commute assms of-nat-ceiling pos-less-divide-eq)
  have ∀∞l. 12 ≤ m-of l
    unfolding m-of-def by real-asymp
  moreover have ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ (6 / μ) powr (12 / 5) ≤ l
    using assms
    apply (intro eventually-all-geI0, real-asymp)
    by (smt (verit, ccfv-SIG) divide-pos-pos frac-le powr-mono2)
  moreover have ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ 4 ≤ 5 * exp (− ((real (b-of
l))^2 / ((μ − 2/l) * m-of l)))
  proof (intro eventually-all-geI0 [where L = nat ⌈3/μ0⌉])
    show ∀∞l. 4 ≤ 5 * exp (− ((real (b-of l))^2 / ((μ0 − 2/l) * m-of l)))
```

```
    unfolding b-of-def m-of-def using assms by real-asymp
  next
    fix l μ
    assume §: 4 ≤ 5 ∗ exp (− ((real (b-of l))² / ((μ0 − 2/l) ∗ m-of l)))
      and μ0 ≤ μ μ ≤ μ1 and lel: nat ⌈3 / μ0⌉ ≤ l
    then have 0: m-of l > 0
      using 3 of-nat-0-eq-iff by (fastforce simp: m-of-def)
    have μ0 > 2/l
      using lel assms by (auto simp: divide-simps mult.commute)
    then show 4 ≤ 5 ∗ exp (− ((real (b-of l))² / ((μ − 2/l) ∗ m-of l)))
      using order-trans [OF §] by (simp add: 0 ‹μ0 ≤ μ› frac-le)
  qed
  moreover have ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ 2/l < μ
    using assms by (intro eventually-all-geI0, real-asymp, linarith)
  moreover have ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ 2/l ≤ (μ − 2/l) ∗ ((5 / 4)
powr (1 / real (b-of l)) − 1)
  proof −
    have ⋀l μ. μ0 ≤ μ ⟹ μ0 − 2/l ≤ μ − 2/l
      by (auto simp: divide-simps ge-one-powr-ge-zero mult.commute)
    show ?thesis
      using assms
      unfolding b-of-def
      apply (intro eventually-all-geI0, real-asymp)
        by (smt (verit, best) divide-le-eq-1 ge-one-powr-ge-zero mult-right-mono
of-nat-0-le-iff zero-le-divide-1-iff)
  qed
  ultimately show ?thesis
    by (auto simp: Big-Blue-4-1-def eventually-conj-iff all-imp-conj-distrib)
qed

context Book
begin

lemma Blue-4-1:
  assumes X⊆V and manyb: many-bluish X and big: Big-Blue-4-1 μ l
  shows ∃S T. good-blue-book X (S,T) ∧ card S ≥ l powr (1/4)
proof −
  have lpowr0[simp]: 0 ≤ ⌈l powr r⌉ for r
    by (metis ceiling-mono ceiling-zero powr-ge-zero)
  define b where b ≡ b-of l
  define W where W ≡ {x∈X. bluish X x}
  define m where m ≡ m-of l
  have m>0 m ≥ 6 m ≥ 12 b>0
    using big by (auto simp: Big-Blue-4-1-def m-def b-def b-of-def)
  have Wbig: card W ≥ RN k m
    using manyb by (simp add: W-def m-def m-of-def many-bluish-def)
  with Red-Blue-RN obtain U where U ⊆ W and U-m-Blue: size-clique m U
Blue
    by (metis W-def ‹X ⊆ V› mem-Collect-eq no-Red-clique subset-eq)
```

then obtain *card U = m* **and** *clique U Blue* **and** *U ⊆ V finite U*
by (*simp add*: *finV finite-subset size-clique-def*)
have *finite X*
using ‹*X⊆V*› *finV finite-subset* **by** *auto*
have *k ≤ RN k m*
using ‹*m≥12*› **by** (*simp add*: *RN-3plus'*)
moreover have *card W ≤ card X*
by (*simp add*: *W-def* ‹*finite X*› *card-mono*)
ultimately have *card X ≥ l*
using *Wbig l-le-k* **by** *linarith*
then have *U ≠ X*
by (*metis U-m-Blue* ‹*card U = m*› *le-eq-less-or-eq no-Blue-clique size-clique-smaller*)
then have *U ⊂ X*
using *W-def* ‹*U ⊆ W*› **by** *blast*
then have *cardU-less-X*: *card U < card X*
by (*meson* ‹*X⊆V*› *finV finite-subset psubset-card-mono*)
with ‹*X⊆V*› **have** *cardXU*: *card* (*X−U*) = *card X − card U*
by (*meson* ‹*U ⊂ X*› *card-Diff-subset finV finite-subset psubset-imp-subset*)
then have *real-cardXU*: *real* (*card* (*X−U*)) = *real* (*card X*) − *m*
using ‹*card U = m*› *cardU-less-X* **by** *linarith*
have [*simp*]: *m ≤ card X*
using ‹*card U = m*› *cardU-less-X nless-le* **by** *blast*
have *lpowr23*: *real l powr* (*2/3*) ≤ *real l powr 1*
using *ln0* **by** (*intro powr-mono*) *auto*
then have *m ≤ l m≤k*
using *l-le-k* **by** (*auto simp*: *m-def m-of-def*)
then have *m < RN k m*
using ‹*12 ≤ m*› *RN-gt2* **by** *auto*
also have *cX*: *RN k m ≤ card X*
using *Wbig* ‹*card W ≤ card X*› **by** *linarith*
finally have *card U < card X*
using ‹*card U = m*› **by** *blast*

First part of (10)

have *card U* ∗ (*μ* ∗ *card X − card U*) = *m* ∗ (*μ* ∗ (*card X − card U*)) − (*1−μ*) ∗ m^2
using *cardU-less-X* **by** (*simp add*: ‹*card U = m*› *algebra-simps numeral-2-eq-2*)
also have ... ≤ *real* (*card* (*Blue ∩ all-edges-betw-un U* (*X−U*)))
proof −
have *dfam*: *disjoint-family-on* (*λu. Blue ∩ all-edges-betw-un* {*u*} (*X−U*)) *U*
by (*auto simp*: *disjoint-family-on-def all-edges-betw-un-def*)
have *μ* ∗ (*card X − card U*) ≤ *card* (*Blue ∩ all-edges-betw-un* {*u*} (*X−U*)) + (*1−μ*) ∗ *m*
if *u ∈ U* **for** *u*
proof −
have *NBU*: *Neighbours Blue u ∩ U = U* − {*u*}
using ‹*clique U Blue*› *Red-Blue-all singleton-not-edge that*
by (*force simp*: *Neighbours-def clique-def*)
then have *NBX-split*: (*Neighbours Blue u ∩ X*) = (*Neighbours Blue u ∩*

$(X-U)) \cup (U - \{u\})$
using ‹$U \subset X$› **by** *blast*
moreover have *Neighbours Blue u* $\cap (X-U) \cap (U - \{u\}) = \{\}$
by *blast*
ultimately have *card*(*Neighbours Blue u* $\cap X$) = *card*(*Neighbours Blue u* $\cap$ $(X-U)) + (m - Suc\ 0)$
by (*simp add*: *card-Un-disjoint finite-Neighbours* ‹*finite U*› ‹*card U = m*› *that*)
then have $\mu *$ (*card X*) $\leq$ *real* (*card* (*Neighbours Blue u* $\cap (X-U)$)) + *real* $(m - Suc\ 0)$
using *W-def* ‹$U \subseteq W$› *bluish-def that* **by** *force*
then have $\mu *$ (*card X* − *card U*)
$\leq$ *card* (*Neighbours Blue u* $\cap (X-U)$) + *real* $(m - Suc\ 0) - \mu *$*card* *U*
by (*smt* (*verit*) *cardU-less-X nless-le of-nat-diff right-diff-distrib'*)
then have ∗: $\mu *$ (*card X* − *card U*) $\leq$ *real* (*card* (*Neighbours Blue u* $\cap$ $(X-U))) + (1-\mu)*m$
using *assms* **by** (*simp add*: ‹*card U = m*› *left-diff-distrib*)
have *inj-on* ($\lambda x.\ \{u,x\}$) (*Neighbours Blue u* $\cap X$)
by (*simp add*: *doubleton-eq-iff inj-on-def*)
moreover have ($\lambda x.\ \{u,x\}$) ‘ (*Neighbours Blue u* $\cap (X-U)$) $\subseteq$ *Blue* $\cap$ *all-edges-betw-un* $\{u\}$ $(X-U)$
using *Blue-E* **by** (*auto simp*: *Neighbours-def all-edges-betw-un-def*)
ultimately have *card* (*Neighbours Blue u* $\cap (X-U)$) $\leq$ *card* (*Blue* $\cap$ *all-edges-betw-un* $\{u\}$ $(X-U)$)
by (*metis NBX-split card-inj-on-le finite-Blue finite-Int inj-on-Un*)
with ∗ **show** *?thesis*
by *auto*
qed
then have (*card U*) $* (\mu *$ *real* (*card X* − *card U*))
$\leq (\sum x \in U.$ *card* (*Blue* $\cap$ *all-edges-betw-un* $\{x\}$ $(X-U)) + (1-\mu) * m)$
by (*meson sum-bounded-below*)
then have $m * (\mu *$ (*card X* − *card U*))
$\leq (\sum x \in U.$ *card* (*Blue* $\cap$ *all-edges-betw-un* $\{x\}$ $(X-U))) + (1-\mu) *$ m^2
by (*simp add*: *sum.distrib power2-eq-square* ‹*card U = m*› *mult-ac*)
also have … $\leq$ *card* ($\bigcup u \in U.$ *Blue* $\cap$ *all-edges-betw-un* $\{u\}$ $(X-U)) + (1-\mu)$ $* m^2$
by (*simp add*: *dfam card-UN-disjoint'* ‹*finite U*› *flip*: *UN-simps*)
finally have $m * (\mu *$ (*card X* − *card U*))
$\leq$ *card* ($\bigcup u \in U.$ *Blue* $\cap$ *all-edges-betw-un* $\{u\}$ $(X-U)) + (1-\mu) *$ m^2 **.**
moreover have ($\bigcup u \in U.$ *Blue* $\cap$ *all-edges-betw-un* $\{u\}$ $(X-U)$) = (*Blue* $\cap$ *all-edges-betw-un* U $(X-U)$)
by (*auto simp*: *all-edges-betw-un-def*)
ultimately show *?thesis*
by *simp*
qed
also have … $\leq$ *edge-card Blue U* $(X-U)$

by (*simp add*: *edge-card-def*)
finally have *edge-card-XU*: *edge-card Blue U* (*X−U*) ≥ *card U* ∗ (*μ* ∗ *card X* − *card U*) **.**
define *σ* **where** *σ* ≡ *blue-density U* (*X−U*)
then have *σ* ≥ *0* **by** (*simp add*: *gen-density-ge0*)
have *σ* ≤ *1*
by (*simp add*: *σ-def gen-density-le1*)
have *6*: *real* (*6*∗*k*) ≤ *real* (*2* + *k*∗*m*)
by (*metis mult.commute* ‹*6*≤*m*› *mult-le-mono2 of-nat-mono trans-le-add2*)
then have *km*: *k* + *m* ≤ *Suc* (*k* ∗ *m*)
using *big l-le-k* ‹*m* ≤ *l*› **by** *linarith*
have *m*/*2* ∗ (*2* + *real k* ∗ (*1*−*μ*)) ≤ *m*/*2* ∗ (*2* + *real k*)
using *assms μ01* **by** (*simp add*: *algebra-simps*)
also have ... ≤ (*k* − *1*) ∗ (*m* − *1*)
using *big l-le-k 6* ‹*m*≤*k*› **by** (*simp add*: *Big-Blue-4-1-def algebra-simps add-divide-distrib km*)
finally have (*m*/*2*) ∗ (*2* + *k* ∗ (*1*−*μ*)) ≤ *RN k m*
using *RN-times-lower′* [*of k m*] **by** *linarith*
then have *μ* − *2*/*k* ≤ (*μ* ∗ *card X* − *card U*) / (*card X* − *card U*)
using *kn0 assms cardU-less-X* ‹*card U* = *m*› *cX* **by** (*simp add*: *field-simps*)
also have ... ≤ *σ*
using ‹*m*>*0*› ‹*card U* = *m*› *cardU-less-X cardXU edge-card-XU*
by (*simp add*: *σ-def gen-density-def divide-simps mult-ac*)
finally have *eq10*: *μ* − *2*/*k* ≤ *σ* **.**
have *2* ∗ *b* / *m* ≤ *μ* − *2*/*k*
proof −
have *512*: *5*/*12* ≤ (*1*::*real*)
by *simp*
with *big* **have** *l powr* (*5*/*12*) ≥ ((*6*/*μ*) *powr* (*12*/*5*)) *powr* (*5*/*12*)
by (*simp add*: *Big-Blue-4-1-def powr-mono2*)
then have *lge*: *l powr* (*5*/*12*) ≥ *6*/*μ*
using *assms μ01 powr-powr* **by** *force*
have *2* ∗ *b* ≤ *2* ∗ (*l powr* (*1*/*4*) + *1*)
by (*simp add*: *b-def b-of-def del*: *zero-le-ceiling distrib-left-numeral*)
then have *2*∗*b* / *m* + *2*/*l* ≤ *2* ∗ (*l powr* (*1*/*4*) + *1*) / *l powr* (*2*/*3*) + *2*/*l*
by (*simp add*: *m-def m-of-def frac-le ln0 del*: *zero-le-ceiling distrib-left-numeral*)
also have ... ≤ (*2* ∗ *l powr* (*1*/*4*) + *4*) / *l powr* (*2*/*3*)
using *ln0 lpowr23* **by** (*simp add*: *pos-le-divide-eq pos-divide-le-eq add-divide-distrib algebra-simps*)
also have ... ≤ (*2* ∗ *l powr* (*1*/*4*) + *4* ∗ *l powr* (*1*/*4*)) / *l powr* (*2*/*3*)
using *big* **by** (*simp add*: *Big-Blue-4-1-def divide-right-mono ge-one-powr-ge-zero*)
also have ... = *6* / *l powr* (*5*/*12*)
by (*simp add*: *divide-simps flip*: *powr-add*)
also have ... ≤ *μ*
using *lge assms μ01* **by** (*simp add*: *divide-le-eq mult.commute*)
finally have *2*∗*b* / *m* + *2*/*l* ≤ *μ* **.**
then show *?thesis*
using *l-le-k* ‹*m*>*0*› *ln0*
by (*smt* (*verit*, *best*) *frac-le of-nat-0-less-iff of-nat-mono*)

qed
with *eq10* **have** $2\ /\ (m/b) \le \sigma$
by *simp*
moreover have $l\ powr\ (2/3) \le nat\ \lceil real\ l\ powr\ (2/3)\rceil$
using *of-nat-ceiling* **by** *blast*
ultimately have *ble*: $b \le \sigma * m\ /\ 2$
using *mult-left-mono* ‹$\sigma \ge 0$› *big kn0 l-le-k*
by (*simp add*: *Big-Blue-4-1-def powr-diff b-def m-def divide-simps*)
then have $\sigma > 0$
using ‹$0 < b$› ‹$0 \le \sigma$› *less-eq-real-def* **by** *force*

define Φ **where** $\Phi \equiv \sum v \in X-U.\ card\ (Neighbours\ Blue\ v \cap U)\ choose\ b$

now for the material between (10) and (11)

have $\sigma * real\ m\ /\ 2 \le m$
using ‹$\sigma \le 1$› ‹$m>0$› **by** *auto*
with *ble* **have** $b \le m$
by *linarith*
have $\mu\hat{}b * 1 * card\ X \le (5/4 * \sigma\hat{}b) * (5/4 * exp(-\ real(b^2)\ /\ (\sigma*m))) * (5/4 * (card\ X - m))$
proof (*intro mult-mono*)
have *2*: $2/k \le 2/l$
by (*simp add*: *l-le-k frac-le ln0*)
also have $\ldots \le (\mu - 2/l) * ((5/4)\ powr\ (1/b) - 1)$
using *big* **by** (*simp add*: *Big-Blue-4-1-def b-def*)
also have $\ldots \le \sigma * ((5/4)\ powr\ (1/b) - 1)$
using *2* ‹$0 < b$› *eq10* **by** *auto*
finally have $2\ /\ real\ k \le \sigma * ((5/4)\ powr\ (1/b) - 1)$ **.**
then have *1*: $\mu \le (5/4) powr(1/b) * \sigma$
using *eq10* ‹$b>0$› **by** (*simp add*: *algebra-simps*)
show $\mu\ \hat{}\ b \le 5/4 * \sigma\ \hat{}\ b$
using *power-mono*[*OF 1*, *of b*] *assms* ‹$\sigma>0$› ‹$b>0$› *μ01*
by (*simp add*: *powr-mult powr-powr flip*: *powr-realpow*)
have $\mu - 2/l \le \sigma$
using *2 eq10* **by** *linarith*
moreover have $2/l < \mu$
using *big* **by** (*auto simp*: *Big-Blue-4-1-def*)
ultimately have $exp\ (-\ real(b^2)\ /\ ((\mu - 2/l) * m)) \le exp\ (-\ real\ (b^2)\ /\ (\sigma * m))$
using ‹$\sigma>0$› ‹$m>0$› **by** (*simp add*: *frac-le*)
then show $1 \le 5/4 * exp\ (-\ real(b^2)\ /\ (\sigma * real\ m))$
using *big* **unfolding** *Big-Blue-4-1-def b-def m-def*
by (*smt* (*verit*, *best*) *divide-minus-left frac-le mult-left-mono*)
have $25 * (real\ m * real\ m) \le 2\ powr\ m$
using *of-nat-mono* [*OF power2-12* [*OF* ‹$12 \le m$›]] **by** (*simp add*: *power2-eq-square powr-realpow*)
then have $real\ (5 * m) \le 2\ powr\ (real\ m\ /\ 2)$
by (*simp add*: *powr-half-sqrt-powr power2-eq-square real-le-rsqrt*)
moreover

have *card X > 2 powr (m/2)*
by (*metis RN-commute RN-lower-nodiag* ‹*6 ≤ m*› ‹*m≤k*› *add-leE less-le-trans cX numeral-Bit0 of-nat-mono*)
ultimately have *5 ∗ m ≤ real* (*card X*)
by *linarith*
then show *card X ≤ 5/4 ∗* (*card X − m*)
using ‹*card U = m*› *cardU-less-X* **by** *simp*
qed (*use* ‹*0 ≤ σ*› **in** *auto*)
also have ... = (*125/64*) ∗ (*σ ^b*) ∗ *exp*(− (*real b*)^2 / (*σ∗m*)) ∗ (*card X − m*)
by *simp*
also have ... ≤ *2* ∗ (*σ ^b*) ∗ *exp*(− (*real b*)^2 / (*σ∗m*)) ∗ (*card X − m*)
by (*intro mult-right-mono*) (*auto simp*: ‹*0 ≤ σ*›)
finally have *μ ^b/2 ∗ card X ≤ σ ^b ∗ exp*(− *of-nat* (b^2) / (*σ∗m*)) ∗ *card* (*X−U*)
by (*simp add*: ‹*card U = m*› *cardXU real-cardXU*)
also have ... ≤ *1*/(*m choose b*) ∗ ((*σ∗m*) *gchoose b*) ∗ *card* (*X−U*)
proof (*intro mult-right-mono*)
have *0 < real m gchoose b*
by (*metis* ‹*b ≤ m*› *binomial-gbinomial of-nat-0-less-iff zero-less-binomial-iff*)
then have *σ ^ b* ∗ ((*real m gchoose b*) ∗ *exp* (− ((*real b*)^2 / (*σ ∗ real m*)))) ≤ *σ ∗ real m gchoose b*
using *Fact-D1-73* [*OF* ‹*σ>0*› ‹*σ≤1*› *ble*] ‹*b≤m*› *cardU-less-X* ‹*0 < σ*›
by (*simp add*: *field-split-simps binomial-gbinomial*)
then show *σ ^b ∗ exp* (− *real* (b^2) / (*σ ∗ m*)) ≤ *1*/(*m choose b*) ∗ (*σ ∗ m gchoose b*)
using ‹*b≤m*› *cardU-less-X* ‹*0 < σ*› ‹*0 < m gchoose b*›
by (*simp add*: *field-split-simps binomial-gbinomial*)
qed *auto*
also have ... ≤ *1*/(*m choose b*) ∗ Φ
unfolding *mult.assoc*
proof (*intro mult-left-mono*)
have *eeq*: *edge-card Blue U* (*X−U*) = (∑*i∈X−U*. *card* (*Neighbours Blue i ∩ U*))
proof (*intro edge-card-eq-sum-Neighbours*)
show *finite* (*X−U*)
by (*meson* ‹*X⊆V*› *finV finite-Diff finite-subset*)
qed (*use disjnt-def Blue-E* **in** *auto*)
have (∑*i∈X − U*. *card* (*Neighbours Blue i ∩ U*)) / (*real* (*card X*) − *m*) = *blue-density U* (*X−U*) ∗ *m*
using ‹*m>0*› **by** (*simp add*: *gen-density-def real-cardXU* ‹*card U = m*› *eeq divide-simps*)
then have ∗: (∑*i∈X − U*. *real* (*card* (*Neighbours Blue i ∩ U*)) $/_R$ *real* (*card* (*X−U*))) = *σ ∗ m*
by (*simp add*: *σ-def divide-inverse-commute real-cardXU flip*: *sum-distrib-left*)
have *mbinomial* (∑*i∈X − U*. *real* (*card* (*Neighbours Blue i ∩ U*)) $/_R$ (*card* (*X−U*))) *b*
≤ (∑*i∈X − U*. *inverse* (*real* (*card* (*X−U*))) ∗ *mbinomial* (*card* (*Neighbours Blue i ∩ U*)) *b*)
proof (*rule convex-on-sum*)
show *finite* (*X−U*)

using *cardU-less-X zero-less-diff* **by** *fastforce*
show *convex-on UNIV* (*λa. mbinomial a b*)
by (*simp add*: ‹*0* < *b*› *convex-mbinomial*)
show (∑*i*∈*X* − *U*. *inverse* (*card* (*X*−*U*))) = *1*
using *cardU-less-X cardXU* **by** *force*
qed (*use* ‹*U* ⊂ *X*› **in** *auto*)
with *ble*
show (*σ*∗*m gchoose b*) ∗ *card* (*X*−*U*) ≤ Φ
unfolding ∗ *Φ-def*
by (*simp add*: *cardU-less-X cardXU binomial-gbinomial divide-simps flip*: *sum-distrib-left sum-divide-distrib*)
qed *auto*
finally have *11*: *μ*ˆ*b* / *2* ∗ *card X* ≤ Φ / (*m choose b*)
by *simp*

define Ω **where** Ω ≡ *nsets U b* — Choose a random subset of size *b*
have *card*Ω: *card* Ω = *m choose b*
by (*simp add*: *Ω-def* ‹*card U* = *m*›)
then have *fin*Ω: *finite* Ω **and** Ω ≠ {} **and** *card* Ω > *0*
using ‹*b* ≤ *m*› *not-less* **by** *fastforce*+
define *M* **where** *M* ≡ *uniform-count-measure* Ω
interpret *P*: *prob-space M*
using *M-def* ‹*b* ≤ *m*› *card*Ω *fin*Ω *prob-space-uniform-count-measure* **by** *force*
have *measure-eq*: *measure M C* = (*if C* ⊆ Ω *then card C* / *card* Ω *else 0*) **for** *C*
by (*simp add*: *M-def fin*Ω *measure-uniform-count-measure-if*)

define *Int-NB* **where** *Int-NB* ≡ *λS*. ⋂*v*∈*S*. *Neighbours Blue v* ∩ (*X*−*U*)
have *sum-card-NB*: (∑*A*∈Ω. *card* (⋂(*Neighbours Blue* ‘ *A*) ∩ *Y*))
= (∑*v*∈*Y*. *card* (*Neighbours Blue v* ∩ *U*) *choose b*)
if *finite Y Y* ⊆ *X*−*U* **for** *Y*
using *that*
proof (*induction Y*)
case (*insert y Y*)
have ∗: Ω ∩ {*A*. ∀*x*∈*A*. *y* ∈ *Neighbours Blue x*} = *nsets* (*Neighbours Blue y* ∩ *U*) *b*
Ω ∩ − {*A*. ∀*x*∈*A*. *y* ∈ *Neighbours Blue x*} = Ω − *nsets* (*Neighbours Blue y* ∩ *U*) *b*
[*Neighbours Blue y* ∩ *U*]^b ⊆ Ω
using *insert.prems* **by** (*auto simp*: *Ω-def nsets-def in-Neighbours-iff insert-commute*)
then show *?case*
using *insert fin*Ω
by (*simp add*: *Int-insert-right sum-Suc sum.If-cases if-distrib* [*of card*]
sum.subset-diff flip: *insert.IH*)
qed *auto*

have (∑*x*∈Ω. *card* (*if x* = {} *then UNIV else* ⋂ (*Neighbours Blue* ‘ *x*) ∩ (*X*−*U*)))
= (∑*x*∈Ω. *card* (⋂ (*Neighbours Blue* ‘ *x*) ∩ (*X*−*U*)))
unfolding *Ω-def nsets-def* **using** ‹*0* < *b*› **by** (*force intro*: *sum.cong*)

also have ... = ($\sum v{\in}X - U$. *card* (*Neighbours Blue* $v \cap U$) *choose* b)
by (*metis sum-card-NB* ‹$X \subseteq V$› *dual-order.refl finV finite-Diff rev-finite-subset*)
finally have *sum* (*card o Int-NB*) $\Omega = \Phi$
by (*simp add*: Ω-*def* Φ-*def Int-NB-def*)
moreover
have *ennreal* (*P.expectation* (λS. *card* (*Int-NB S*))) = *sum* (*card o Int-NB*) Ω / (*card* Ω)
using *integral-uniform-count-measure M-def fin*Ω **by** *fastforce*
ultimately have *P*: *P.expectation* (λS. *card* (*Int-NB S*)) = Φ / (*m choose b*)
by (*metis Bochner-Integration.integral-nonneg card*Ω *divide-nonneg-nonneg ennreal-inj of-nat-0-le-iff*)
have *False* **if** $\bigwedge S.\ S \in \Omega \Longrightarrow$ *card* (*Int-NB S*) < Φ / (*m choose b*)
proof −
define *L* **where** $L \equiv$ (λS. Φ / *real* (*m choose b*) − *card* (*Int-NB S*)) ‘ Ω
have *finite L* $L \neq \{\}$
using *L-def fin*Ω ‹$\Omega \neq \{\}$› **by** *blast+*
define ε **where** $\varepsilon \equiv$ *Min L*
have $\varepsilon > 0$
using *that fin*Ω ‹$\Omega \neq \{\}$› **by** (*simp add*: *L-def* ε-*def*)
then have $\bigwedge S.\ S \in \Omega \Longrightarrow$ *card* (*Int-NB S*) $\leq \Phi$ / (*m choose b*) − ε
using *Min-le* [*OF* ‹*finite L*›] **by** (*fastforce simp*: *algebra-simps* ε-*def L-def*)
then have *P.expectation* (λS. *card* (*Int-NB S*)) $\leq \Phi$ / (*m choose b*) − ε
using *P P.not-empty not-integrable-integral-eq* ‹$\varepsilon > 0$›
by (*intro P.integral-le-const*) (*fastforce simp*: *M-def space-uniform-count-measure*)+
then show *False*
using *P* ‹$0 < \varepsilon$› **by** *auto*
qed
then obtain *S* **where** $S \in \Omega$ **and** *Sge*: *card* (*Int-NB S*) $\geq \Phi$ / (*m choose b*)
using *linorder-not-le* **by** *blast*
then have $S \subseteq U$
by (*simp add*: Ω-*def nsets-def subset-iff*)
have *card S* = *b clique S Blue*
using ‹$S \in \Omega$› ‹$U \subseteq V$› ‹*clique U Blue*› *smaller-clique*
unfolding Ω-*def nsets-def size-clique-def* **by** *auto*
have Φ / (*m choose b*) $\geq \mu$^*b* ∗ *card X* / *2*
using *11* **by** *simp*
then have *S*: *card* (*Int-NB S*) $\geq \mu$^*b* ∗ *card X* / *2*
using *Sge* **by** *linarith*
obtain *v* **where** $v \in S$
using ‹$0 < b$› ‹*card S* = *b*› **by** *fastforce*
have *all-edges-betw-un S* ($S \cup$ *Int-NB S*) $\subseteq$ *Blue*
using ‹*clique S Blue*›
unfolding *all-edges-betw-un-def Neighbours-def clique-def Int-NB-def* **by** *fastforce*
then have *good-blue-book X* (*S*, *Int-NB S*)
using ‹$S \subseteq U$› ‹$v \in S$› ‹$U \subset X$› *S* ‹*card S* = *b*›
unfolding *good-blue-book-def book-def size-clique-def Int-NB-def disjnt-iff*
by *blast*
then show *?thesis*
by (*metis* ‹*card S* = *b*› *b-def b-of-def of-nat-ceiling*)

qed

Lemma 4.3

proposition *bblue-step-limit*:
  assumes *big*: *Big-Blue-4-1 μ l*
  shows *card* (*Step-class* {*bblue-step*}) ≤ *l powr* (*3*/*4*)
proof −
   define *BBLUES* **where** *BBLUES* ≡ λ*r*. {*m*. *m* < *r* ∧ *stepper-kind m* = *bblue-step*}
  have *cardB-ge*: *card* (*Bseq n*) ≥ *b-of l* ∗ *card*(*BBLUES n*)
    for *n*
  proof (*induction n*)
    case *0* **then show** *?case* **by** (*auto simp*: *BBLUES-def*)
  next
    case (*Suc n*)
    show *?case*
    proof (*cases stepper-kind n* = *bblue-step*)
      case *True*
      have [*simp*]: *card* (*insert n* (*BBLUES n*)) = *Suc* (*card* (*BBLUES n*))
        by (*simp add*: *BBLUES-def*)
      have *card-B'*: *card* (*Bseq* (*Suc n*)) ≥ *b-of l* ∗ *card* (*BBLUES n*)
        using *Suc.IH*
        by (*meson Bseq-Suc-subset card-mono finite-Bseq le-trans*)

      define *S* **where** *S* ≡ *fst* (*choose-blue-book* (*Xseq n*, *Yseq n*, *Aseq n*, *Bseq n*))
      have *BSuc*: *Bseq* (*Suc n*) = *Bseq n* ∪ *S*
        and *manyb*: *many-bluish* (*Xseq n*)
         and *cbb*: *choose-blue-book* (*Xseq n*, *Yseq n*, *Aseq n*, *Bseq n*) = (*S*, *Xseq* (*Suc n*))
        and *same*: *Aseq* (*Suc n*) = *Aseq n* *Yseq* (*Suc n*) = *Yseq n*
        using *True*
      by (*force simp*: *S-def step-kind-defs next-state-def split*: *prod.split if-split-asm*)+

      have *l14*: *l powr* (*1*/*4*) ≤ *card S*
        using *Blue-4-1* [*OF Xseq-subset-V manyb big*]
           by (*smt* (*verit*, *best*) *choose-blue-book-works best-blue-book-is-best cbb finite-Xseq of-nat-mono*)
      then have *ble*: *b-of l* ≤ *card S*
        using *b-of-def nat-ceiling-le-eq* **by** *presburger*
      have *S*: *good-blue-book* (*Xseq n*) (*S*, *Xseq* (*Suc n*))
        by (*metis cbb choose-blue-book-works finite-Xseq*)
      then have *card S* ≤ *best-blue-book-card* (*Xseq n*)
        by (*simp add*: *best-blue-book-is-best finite-Xseq*)
      have *finS*: *finite S*
        using *ln0 l14 card.infinite* **by** *force*
      have *disjnt* (*Bseq n*) (*Xseq n*)
        using *valid-state-seq* [*of n*]
         by (*auto simp*: *Bseq-def Xseq-def valid-state-def disjoint-state-def disjnt-iff*

split: *prod.split-asm*)
    then have *dBS*: *disjnt* (*Bseq n*) *S*
      using *S cbb* **by** (*force simp*: *good-blue-book-def book-def disjnt-iff*)
    have *eq*: *BBLUES*(*Suc n*) = *insert n* (*BBLUES n*)
      using *less-Suc-eq True* **unfolding** *BBLUES-def* **by** *blast*
    then have *b-of l* ∗ *card* (*BBLUES* (*Suc n*)) = *b-of l* + *b-of l* ∗ *card* (*BBLUES n*)
      by *auto*
    also have . . . ≤ *card* (*Bseq n*) + *card S*
      using *ble card-B′ Suc.IH* **by** *linarith*
    also have . . . ≤ *card* (*Bseq n* ∪ *S*)
      using *ble dBS* **by** (*simp add*: *card-Un-disjnt finS finite-Bseq*)
    finally have ∗∗: *b-of l* ∗ *card* (*BBLUES* (*Suc n*)) ≤ *card* (*Bseq* (*Suc n*))
      using *order.trans BSuc* **by** *argo*
    then show *?thesis*
      by (*simp add*: *BBLUES-def*)
  next
    case *False*
    then have *BBLUES*(*Suc n*) = *BBLUES n*
      using *less-Suc-eq* **by** (*auto simp*: *BBLUES-def*)
    then show *?thesis*
      by (*metis Bseq-Suc-subset Suc.IH card-mono finite-Bseq le-trans*)
  qed
qed
{ **assume** §: *card* (*Step-class* {*bblue-step*}) > *l powr* (*3/4*)
  then have *fin*: *finite* (*Step-class* {*bblue-step*})
    using *card.infinite* **by** *fastforce*
  then obtain *n* **where** *n*: (*Step-class* {*bblue-step*}) = {*m*. *m*<*n* ∧ *stepper-kind m* = *bblue-step*}
    using *Step-class-iterates* **by** *blast*
  with § **have** *card-gt*: *card*{*m*. *m*<*n* ∧ *stepper-kind m* = *bblue-step*} > *l powr* (*3/4*)
    by (*simp add*: *n*)
  have *l* = *l powr* (*1/4*) ∗ *l powr* (*3/4*)
    by (*simp flip*: *powr-add*)
  also have . . . ≤ *b-of l* ∗ *l powr* (*3/4*)
    by (*simp add*: *b-of-def mult-mono′*)
  also have . . . ≤ *b-of l* ∗ *card*{*m*. *m*<*n* ∧ *stepper-kind m* = *bblue-step*}
    using *card-gt less-eq-real-def* **by** *fastforce*
  also have . . . ≤ *card* (*Bseq n*)
    using *cardB-ge step of-nat-mono* **unfolding** *BBLUES-def* **by** *blast*
  also have . . . < *l*
    by (*simp add*: *Bseq-less-l*)
  finally have *False*
    by *simp*
}
then show *?thesis* **by** *force*
qed

lemma *red-steps-eq-A*:
  defines *REDS* ≡ *λr*. {*i*. *i* < *r* ∧ *stepper-kind i* = *red-step*}
  shows *card*(*REDS n*) = *card* (*Aseq n*)
proof (*induction n*)
  case *0*
  then show *?case*
    by (*auto simp*: *REDS-def*)
next
  case (*Suc n*)
  show *?case*
  proof (*cases stepper-kind n* = *red-step*)
    case *True*
    then have [*simp*]: *REDS* (*Suc n*) = *insert n* (*REDS n*) *card* (*insert n* (*REDS n*)) = *Suc* (*card* (*REDS n*))
      by (*auto simp*: *REDS-def*)
    have *Aeq*: *Aseq* (*Suc n*) = *insert* (*choose-central-vx* (*Xseq n*,*Yseq n*,*Aseq n*,*Bseq n*)) (*Aseq n*)
      using *Suc.prems True*
      by (*auto simp*: *step-kind-defs next-state-def split*: *if-split-asm prod.split*)
    have *finite* (*Xseq n*)
      using *finite-Xseq* **by** *presburger*
    then have *choose-central-vx* (*Xseq n*,*Yseq n*,*Aseq n*,*Bseq n*) ∈ *Xseq n*
      using *True*
      by (*simp add*: *step-kind-defs choose-central-vx-X split*: *if-split-asm prod.split-asm*)
    moreover have *disjnt* (*Xseq n*) (*Aseq n*)
      using *valid-state-seq* **by** (*simp add*: *valid-state-def disjoint-state-def*)
    ultimately have *choose-central-vx* (*Xseq n*,*Yseq n*,*Aseq n*,*Bseq n*) ∉ *Aseq n*
      by (*simp add*: *disjnt-iff*)
    then show *?thesis*
      by (*simp add*: *Aeq Suc.IH finite-Aseq*)
  next
    case *False*
    then have *REDS*(*Suc n*) = *REDS n*
      using *less-Suc-eq* **unfolding** *REDS-def* **by** *blast*
    moreover have *Aseq* (*Suc n*) = *Aseq n*
      using *False*
      by (*auto simp*: *step-kind-defs degree-reg-def next-state-def split*: *prod.split*)
    ultimately show *?thesis*
      using *Suc.IH* **by** *presburger*
  qed
qed

proposition *red-step-eq-Aseq*: *card* (*Step-class* {*red-step*}) = *card* (*Aseq halted-point*)
proof −
  have *card*{*i*. *i* < *halted-point* ∧ *stepper-kind i* = *red-step*} = *card* (*Aseq halted-point*)
    by (*rule red-steps-eq-A*)
  moreover have (*Step-class* {*red-step*}) = {*i*. *i* < *halted-point* ∧ *stepper-kind i* = *red-step*}

using *halted-point-minimal′* **by** (*fastforce simp*: *Step-class-def*)
ultimately show *?thesis*
by *argo*
qed

proposition *red-step-limit*: *card* (*Step-class* {*red-step*}) < *k*
using *Aseq-less-k red-step-eq-Aseq* **by** *presburger*

proposition *bblue-dboost-step-limit*:
assumes *big*: *Big-Blue-4-1* μ *l*
shows *card* (*Step-class* {*bblue-step*}) + *card* (*Step-class* {*dboost-step*}) < *l*
proof −
define *BDB* **where** *BDB* ≡ λ*r*. {*i*. *i* < *r* ∧ *stepper-kind i* ∈ {*bblue-step*,*dboost-step*}}
have ∗: *card*(*BDB n*) ≤ *card B* — looks clunky but gives access to all state components
if *stepper n* = (*X*,*Y*,*A*,*B*) **for** *n X Y A B*
using *that*
proof (*induction n arbitrary*: *X Y A B*)
case *0*
then show *?case*
by (*auto simp*: *BDB-def*)
next
case (*Suc n*)
obtain *X′ Y′ A′ B′* **where** *step-n*: *stepper n* = (*X′*,*Y′*,*A′*,*B′*)
by (*metis surj-pair*)
then obtain *valid-state* (*X′*,*Y′*,*A′*,*B′*) **and** *V-state* (*X′*,*Y′*,*A′*,*B′*)
and *disjst*: *disjoint-state*(*X′*,*Y′*,*A′*,*B′*) **and** *finite X′*
by (*metis finX valid-state-def valid-state-stepper*)
have *B′* ⊆ *B*
using *Suc.prems* **by** (*auto simp*: *next-state-def Let-def degree-reg-def step-n split*: *prod.split-asm if-split-asm*)
show *?case*
proof (*cases stepper-kind n* ∈ {*bblue-step*,*dboost-step*})
case *True*
then have *BDB* (*Suc n*) = *insert n* (*BDB n*)
by (*auto simp*: *BDB-def*)
moreover have *card* (*insert n* (*BDB n*)) = *Suc* (*card* (*BDB n*))
by (*simp add*: *BDB-def*)
ultimately have *card-Suc*[*simp*]: *card* (*BDB* (*Suc n*)) = *Suc* (*card* (*BDB n*))
by *presburger*
have *card-B′*: *card* (*BDB n*) ≤ *card B′*
using *step-n BDB-def Suc.IH* **by** *blast*
consider *stepper-kind n* = *bblue-step* | *stepper-kind n* = *dboost-step*
using *True* **by** *force*
then have *Bigger*: *B′* ⊂ *B*
proof *cases*
case *1*
then have ¬ *termination-condition X′ Y′*

```
          by (auto simp: stepper-kind-def step-n)
        with 1 obtain S where A' = A Y' = Y and manyb: many-bluish X'
          and cbb: choose-blue-book (X',Y,A,B') = (S,X) and le-cardB: B = B' ∪
S
          using Suc.prems
            by (auto simp: step-kind-defs next-state-def step-n split: prod.split-asm
if-split-asm)
        then obtain X' ⊆ V finite X'
          using Xseq-subset-V ‹finite X'› step-n stepper-XYseq by blast
        then have l powr (1/4) ≤ real (card S)
          using Blue-4-1 [OF - manyb big]
       by (smt (verit, best) of-nat-mono best-blue-book-is-best cbb choose-blue-book-works)
        then have S ≠ {}
          using ln0 by fastforce
        moreover have disjnt B' S
          using choose-blue-book-subset [OF ‹finite X'›] disjst cbb
          unfolding disjoint-state-def
          by (smt (verit) in-mono ‹A' = A› ‹Y' = Y› disjnt-iff old.prod.case)
        ultimately show ?thesis
          by (metis ‹B' ⊆ B› disjnt-Un1 disjnt-self-iff-empty le-cardB psubsetI)
      next
        case 2
        then have choose-central-vx (X',Y',A',B') ∈ X'
          unfolding step-kind-defs
          by (auto simp: ‹finite X'› choose-central-vx-X step-n split: if-split-asm)
        moreover have disjnt B' X'
          using disjst disjnt-sym by (force simp: disjoint-state-def)
        ultimately have choose-central-vx (X',Y',A',B') ∉ B'
          by (meson disjnt-iff)
        then show ?thesis
          using 2 Suc.prems
          by (auto simp: step-kind-defs next-state-def step-n split: if-split-asm)
      qed
      moreover have finite B
        by (metis Suc.prems V-state-stepper finB)
      ultimately show ?thesis
        by (metis card-B' card-Suc card-seteq le-trans not-less-eq-eq psubset-eq)
    next
      case False
      then have BDB (Suc n) = BDB n
        using less-Suc-eq unfolding BDB-def by blast
      with ‹B' ⊆ B› Suc show ?thesis
        by (metis V-state-stepper card-mono finB le-trans step-n)
    qed
  qed
  have less-l: card (BDB n) < l for n
    by (meson card-B-limit ∗ order.trans linorder-not-le prod-cases4)
  moreover have fin: ⋀n. finite (BDB n) incseq BDB
    by (auto simp: BDB-def incseq-def)
```

ultimately have ∗∗: ∀∞n. ⋃ (*range BDB*) = *BDB n*
using *Union-incseq-finite* **by** *blast*
then have *finite* (⋃ (*range BDB*))
using *BDB-def eventually-sequentially* **by** *force*
moreover have *Uneq*: ⋃ (*range BDB*) = *Step-class* {*bblue-step*,*dboost-step*}
by (*auto simp*: *Step-class-def BDB-def*)
ultimately have *fin*: *finite* (*Step-class* {*bblue-step*,*dboost-step*})
by *fastforce*
obtain *n* **where** ⋃ (*range BDB*) = *BDB n*
using ∗∗ **by** *force*
then have *card* (*BDB n*) = *card* (*Step-class* {*bblue-step*} ∪ *Step-class* {*dboost-step*})
by (*metis Step-class-insert Uneq*)
also have ... = *card* (*Step-class* {*bblue-step*}) + *card* (*Step-class* {*dboost-step*})
by (*simp add*: *card-Un-disjnt disjnt-Step-class*)
finally show *?thesis*
by (*metis less-l*)
qed

end

end

4 Red Steps: theorems

theory *Red-Steps* **imports** *Big-Blue-Steps*

begin

Bhavik Mehta: choose-free Ramsey lower bound that's okay for very small *p*

lemma *Ramsey-number-lower-simple*:
fixes *p*::*real*
assumes *n*: *n*^*k* ∗ *p powr* (*k*^*2* / *4*) + *n*^*l* ∗ *exp* (−*p* ∗ *l*^*2* / *4*) < *1*
assumes *p01*: *0*<*p p*<*1* **and** *k*>*1 l*>*1*
shows ¬ *is-Ramsey-number k l n*
proof (*rule Ramsey-number-lower-gen*)
have (*n choose k*) ∗ *p*^(*k choose 2*) ≤ *n*^*k* ∗ *p powr* (*real k*^*2* / *4*)
proof −
have (*n choose k*) ∗ *p*^(*k choose 2*) ≤ *real* (*Suc n* − *k*)^*k* ∗ *p*^(*k choose 2*)
using *choose-le-power p01* **by** *simp*
also have ... = *real* (*Suc n* − *k*)^*k* ∗ *p powr* (*k* ∗ (*real k* − *1*) / *2*)
by (*metis choose-two-real p01*(*1*) *powr-realpow*)
also have ... ≤ *n*^*k* ∗ *p powr* (*real k*^*2* / *4*)
using *p01* ‹*k*>*1*› **by** (*intro mult-mono powr-mono'*) (*auto simp*: *power2-eq-square*)
finally show *?thesis* **.**
qed
moreover
have *real* (*n choose l*) ∗ (*1* − *p*)^(*l choose 2*) ≤ *n*^*l* ∗ *exp* (−*p* ∗ *real l*^*2* / *4*)
proof −

show *?thesis*
proof (*intro mult-mono*)
show *real* (*n choose l*) ≤ *n^l*
by (*metis binomial-eq-0-iff binomial-le-pow not-le of-nat-le-iff zero-le*)
have *l* ∗ *p* ≤ *2* ∗ (*1* − *real l*) ∗ −*p*
using *assms* **by** (*auto simp*: *algebra-simps*)
also have ... ≤ *2* ∗ (*1* − *real l*) ∗ *ln* (*1*−*p*)
using *p01* ‹*l*>*1*› *ln-add-one-self-le-self2* [*of* −*p*]
by (*intro mult-left-mono-neg*) *auto*
finally have *real l* ∗ (*real l* ∗ *p*) ≤ *real l* ∗ (*2* ∗ (*1* − *real l*) ∗ *ln* (*1*−*p*))
using *mult-left-mono* ‹*l*>*1*› **by** *fastforce*
with *p01* **show** (*1* − *p*) ^(*l choose 2*) ≤ *exp* (− *p* ∗ (*real l*)^2 / *4*)
by (*simp add*: *field-simps power2-eq-square powr-def choose-two-real flip*: *powr-realpow*)
qed (*use p01* **in** *auto*)
qed
ultimately
show *real* (*n choose k*) ∗ *p* ^(*k choose 2*) + *real* (*n choose l*) ∗ (*1* − *p*) ^(*l choose 2*) < *1*
using *n* **by** *auto*
qed (*use p01* **in** *auto*)

context *Book*
begin

4.1 Density-boost steps

4.1.1 Observation 5.5

lemma *sum-Weight-ge0*:
assumes *X* ⊆ *V* *Y* ⊆ *V* *disjnt X Y*
shows (∑*x*∈*X*. ∑*x'*∈*X*. *Weight X Y x x'*) ≥ *0*
proof −
have *finite X finite Y*
using *assms finV finite-subset* **by** *blast*+
with *Red-E* **have** *EXY*: *edge-card Red X Y* = (∑*x*∈*X*. *card* (*Neighbours Red x* ∩ *Y*))
by (*metis* ‹*disjnt X Y*› *disjnt-sym edge-card-commute edge-card-eq-sum-Neighbours*)
have (∑*x*∈*X*. ∑*x'*∈*X*. *red-density X Y* ∗ *card* (*Neighbours Red x* ∩ *Y*))
= *red-density X Y* ∗ *card X* ∗ *edge-card Red X Y*
using *assms Red-E*
by (*simp add*: *EXY power2-eq-square edge-card-eq-sum-Neighbours flip*: *sum-distrib-left*)
also have ... = *red-density X Y^2* ∗ *card X^2* ∗ *card Y*
by (*simp add*: *power2-eq-square gen-density-def*)
also have ... = ((∑*i*∈*Y*. *card* (*Neighbours Red i* ∩ *X*)) / (*real* (*card X*) ∗ *real* (*card Y*)))^2 ∗ (*card X*)^2 ∗ *card Y*
using *Red-E* ‹*finite Y*› *assms*
by (*simp add*: *psubset-eq gen-density-def edge-card-eq-sum-Neighbours*)
also have ... ≤ (∑*y*∈*Y*. *real* ((*card* (*Neighbours Red y* ∩ *X*))^2))

proof (*cases card* $Y = 0$)
case *False*
then have $(\sum x{\in}Y.\ real\ (card\ (Neighbours\ Red\ x \cap X)))^2$
$\leq (\sum y{\in}Y.\ (real\ (card\ (Neighbours\ Red\ y \cap X)))^2) * card\ Y$
using ‹*finite Y*› *assms* **by** (*intro sum-squared-le-sum-of-squares*) *auto*
then show *?thesis*
using *assms False* **by** (*simp add*: *divide-simps power2-eq-square sum-nonneg*)
qed (*auto simp*: *sum-nonneg*)
also have ... $= (\sum x{\in}X.\ \sum x'{\in}X.\ real\ (card\ (Neighbours\ Red\ x \cap Neighbours\ Red\ x' \cap Y)))$
proof −
define $f{::}'a \times 'a \times 'a \Rightarrow 'a \times 'a \times 'a$ **where** $f \equiv \lambda(y,(x,x')).\ (x,\ (x',\ y))$
have f: *bij-betw* f (*SIGMA* $y{:}Y.$ (*Neighbours Red* $y \cap X$) $\times$ (*Neighbours Red* $y \cap X$))
(*SIGMA* $x{:}X.$ *SIGMA* $x'{:}X.$ *Neighbours Red* $x \cap$ *Neighbours Red* $x' \cap Y$)
by (*auto simp*: *f-def bij-betw-def inj-on-def image-iff in-Neighbours-iff doubleton-eq-iff insert-commute*)
have $(\sum y{\in}Y.\ (card\ (Neighbours\ Red\ y \cap X))^2) = card(SIGMA\ y{:}Y.\ (Neighbours\ Red\ y \cap X) \times (Neighbours\ Red\ y \cap X))$
by (*simp add*: ‹*finite Y*› *finite-Neighbours power2-eq-square*)
also have ... $= card(Sigma\ X\ (\lambda x.\ Sigma\ X\ (\lambda x'.\ Neighbours\ Red\ x \cap Neighbours\ Red\ x' \cap Y)))$
using *bij-betw-same-card f* **by** *blast*
also have ... $= (\sum x{\in}X.\ \sum x'{\in}X.\ card\ (Neighbours\ Red\ x \cap Neighbours\ Red\ x' \cap Y))$
by (*simp add*: ‹*finite X*› *finite-Neighbours power2-eq-square*)
finally
have $(\sum y{\in}Y.\ (card\ (Neighbours\ Red\ y \cap X))^2) =$
$(\sum x{\in}X.\ \sum x'{\in}X.\ card\ (Neighbours\ Red\ x \cap Neighbours\ Red\ x' \cap Y))$ **.**
then show *?thesis*
by (*simp flip*: *of-nat-sum of-nat-power*)
qed
finally have $(\sum x{\in}X.\ \sum y{\in}X.\ red\text{-}density\ X\ Y * card\ (Neighbours\ Red\ x \cap Y))$
$\leq (\sum x{\in}X.\ \sum y{\in}X.\ real\ (card\ (Neighbours\ Red\ x \cap Neighbours\ Red\ y \cap Y)))$
.
then show *?thesis*
by (*simp add*: *Weight-def sum-subtractf inverse-eq-divide flip*: *sum-divide-distrib*)
qed

end

4.1.2 Lemma 5.6

definition *Big-Red-5-6-Ramsey* $\equiv$
$\lambda c\ l.\ nat\ \lceil real\ l\ powr\ (3/4)\rceil \geq 3$
$\wedge\ (l\ powr\ (3/4) * (c - 1/32) \leq -1)$
$\wedge\ (\forall k{\geq}l.\ k * (c * l\ powr\ (3/4) * ln\ k - k\ powr\ (7/8)\ /\ 4) \leq -1)$

establishing the size requirements for 5.6

```
lemma Big-Red-5-6-Ramsey:
  assumes 0<c c<1/32
  shows ∀∞l. Big-Red-5-6-Ramsey c l
proof −
  have D34: ⋀l k. l ≤ k ⟹ c ∗ real l powr (3/4) ≤ c ∗ real k powr (3/4)
    by (simp add: assms powr-mono2)
  have D0: ∀∞l. l ∗ (c ∗ l powr (3/4) ∗ ln l − l powr (7/8) / 4) ≤ −1
    using ‹c>0› by real-asymp
  have ⋀l k. l ≤ k ⟹ c ∗ real l powr (3/4) ∗ ln k ≤ c ∗ real k powr (3/4) ∗ ln k
    using D34 le-eq-less-or-eq mult-right-mono by fastforce
  then have D: ∀∞l. ∀k≥l. k ∗ (c ∗ l powr (3/4) ∗ ln k − real k powr (7/8) /
4) ≤ −1
    using eventually-mono [OF eventually-all-ge-at-top [OF D0]]
    by (smt (verit, ccfv-SIG) mult-left-mono of-nat-0-le-iff)
  show ?thesis
    using assms
    unfolding Big-Red-5-6-Ramsey-def eventually-conj-iff m-of-def
    by (intro conjI eventually-all-ge-at-top D; real-asymp)
qed

lemma Red-5-6-Ramsey:
  assumes 0<c c<1/32 and l≤k and big: Big-Red-5-6-Ramsey c l
  shows exp (c ∗ l powr (3/4) ∗ ln k) ≤ RN k (nat⌈l powr (3/4)⌉)
proof −
  define r where r ≡ nat ⌊exp (c ∗ l powr (3/4) ∗ ln k)⌋
  define s where s ≡ nat ⌈l powr (3/4)⌉
  have l≠0
    using big by (force simp: Big-Red-5-6-Ramsey-def)
  have 3 ≤ s
    using assms by (auto simp: Big-Red-5-6-Ramsey-def s-def)
  also have ... ≤ l
    using powr-mono [of 3/4 1] ‹l ≠ 0› by (simp add: s-def)
  finally have 3 ≤ l .
  then have k≥3 ‹k>0› ‹l>0›
    using assms by auto
  define p where p ≡ k powr (−1/8)
  have p01: 0 < p p < 1
    using ‹k≥3› powr-less-one by (auto simp: p-def)
  have r-le: r ≤ k powr (c ∗ l powr (3/4))
    using p01 ‹k≥3› unfolding r-def powr-def by force

  have left: r^s ∗ p powr ((real s)² / 4) < 1/2
  proof −
    have A: r powr s ≤ k powr (s ∗ c ∗ l powr (3/4))
     using r-le by (smt (verit) mult.commute of-nat-0-le-iff powr-mono2 powr-powr)
    have B: p powr ((real s)² / 4) ≤ k powr (−(real s)² / 32)
      by (simp add: powr-powr p-def power2-eq-square)
    have C: (c ∗ l powr (3/4) − s/32) ≤ −1
```

using *big* **by** (*simp add*: *Big-Red-5-6-Ramsey-def s-def algebra-simps*) *linarith*
have $r\hat{}s * p$ *powr* ((*real s*)^2 / *4*) ≤ *k powr* (*s* ∗ (*c* ∗ *l powr* (*3/4*) − *s* / *32*))
using *mult-mono* [*OF A B*] ‹*s*≥*3*›
by (*simp add*: *power2-eq-square algebra-simps powr-realpow′ flip*: *powr-add*)
also have ... ≤ *k powr* − *real s*
using *C* ‹*s*≥*3*› *mult-left-mono* ‹*k*≥*3*› **by** *fastforce*
also have ... ≤ *k powr* −*3*
using ‹*k*≥*3*› ‹*s*≥*3*› **by** (*simp add*: *powr-minus powr-realpow*)
also have ... ≤ *3 powr* −*3*
using ‹*k*≥*3*› **by** (*intro powr-mono2′*) *auto*
also have ... < *1/2*
by *auto*
finally show *?thesis* **.**
qed
have *right*: $r\hat{}k * exp\ (- p * (real\ k)^2\ /\ 4) < 1/2$
proof −
have *A*: $r\hat{}k$ ≤ *exp* (*c* ∗ *l powr* (*3/4*) ∗ *ln k* ∗ *k*)
using *r-le* ‹*0* < *k*› ‹*0* < *l*› **by** (*simp add*: *powr-def exp-of-nat2-mult*)
have *B*: *exp* (− *p* ∗ (*real k*)^2 / *4*) ≤ *exp* (− *k* ∗ *k powr* (*7/8*) / *4*)
using ‹*k*>*0*› **by** (*simp add*: *p-def mult-ac power2-eq-square powr-mult-base*)
have $r\hat{}k$ ∗ *exp* (− *p* ∗ (*real k*)^2 / *4*) ≤ *exp* (*k* ∗ (*c* ∗ *l powr* (*3/4*) ∗ *ln k* − *k powr* (*7/8*) / *4*))
using *mult-mono* [*OF A B*] **by** (*simp add*: *algebra-simps s-def flip*: *exp-add*)
also have ... ≤ *exp* (−*1*)
using *assms* **unfolding** *Big-Red-5-6-Ramsey-def* **by** *blast*
also have ... < *1/2*
by (*approximation 5*)
finally show *?thesis* **.**
qed
have ¬ *is-Ramsey-number* (*nat*⌈*l powr* (*3/4*)⌉) *k* (*nat* ⌊*exp* (*c* ∗ *l powr* (*3/4*) ∗ *ln k*)⌋)
using *Ramsey-number-lower-simple* [*OF - p01*] *left right* ‹*k*≥*3*› ‹*l*≥*3*›
unfolding *r-def s-def* **by** *force*
then show *?thesis*
by (*smt* (*verit*) *RN-commute is-Ramsey-number-RN le-nat-floor partn-lst-greater-resource*)
qed

definition *ineq-Red-5-6* ≡ λ*c l*. ∀ *k*. *l* ≤ *k* ⟶ *exp* (*c* ∗ *real l powr* (*3/4*) ∗ *ln k*) ≤ *RN k* (*nat*⌈*l powr* (*3/4*)⌉)

definition *Big-Red-5-6* ≡
λ*l*. *6* + *m-of l* ≤ (*1/128*) ∗ *l powr* (*3/4*) ∧ *ineq-Red-5-6* (*1/128*) *l*

establishing the size requirements for 5.6

lemma *Big-Red-5-6*: ∀$^\infty$*l*. *Big-Red-5-6 l*
proof −
define *c*::*real* **where** *c* ≡ *1/128*
have *0* < *c c* < *1/32*
by (*auto simp*: *c-def*)

```
  then have ∀∞l. ineq-Red-5-6 c l
   unfolding ineq-Red-5-6-def using Red-5-6-Ramsey Big-Red-5-6-Ramsey exp-gt-zero
    by (smt (verit, del-insts) eventually-sequentially)
  then show ?thesis
    unfolding Big-Red-5-6-def eventually-conj-iff m-of-def
    by (simp add: c-def; real-asymp)
qed

lemma (in Book) Red-5-6:
  assumes big: Big-Red-5-6 l
  shows RN k (nat⌈l powr (3/4)⌉) ≥ k^6 ∗ RN k (m-of l)
proof −
  define c::real where c ≡ 1/128
  have RN k (m-of l) ≤ k^(m-of l)
   by (metis RN-le-argpower' RN-mono diff-add-inverse diff-le-self le-refl le-trans)
  also have … ≤ exp (m-of l ∗ ln k)
    using kn0 by (simp add: exp-of-nat-mult)
  finally have RN k (m-of l) ≤ exp (m-of l ∗ ln k)
    by force
  then have k^6 ∗ RN k (m-of l) ≤ real k^6 ∗ exp (m-of l ∗ ln k)
    by (simp add: kn0)
  also have … ≤ exp (c ∗ l powr (3/4) ∗ ln k)
  proof −
    have (6 + real (m-of l)) ∗ ln (real k) ≤ (c ∗ l powr (3/4)) ∗ ln (real k)
      unfolding mult-le-cancel-right
      using big kn0 by (auto simp: c-def Big-Red-5-6-def)
    then have ln (real k^6 ∗ exp (m-of l ∗ ln k)) ≤ ln (exp (c ∗ l powr (3/4) ∗
ln k))
      using kn0 by (simp add: ln-mult ln-powr algebra-simps flip: powr-numeral)
    then show ?thesis
      by (smt (verit) exp-gt-zero ln-le-cancel-iff)
  qed
  also have … ≤ RN k (nat⌈l powr (3/4)⌉)
    using assms l-le-k by (auto simp: ineq-Red-5-6-def Big-Red-5-6-def c-def)
  finally show k^6 ∗ RN k (m-of l) ≤ RN k (nat⌈l powr (3/4)⌉)
    using of-nat-le-iff by blast
qed
```

4.2 Lemma 5.4

```
definition Big-Red-5-4 ≡ λl. Big-Red-5-6 l ∧ (∀k≥l. real k + 2 ∗ real k^6 ≤ real
k^7)
```

establishing the size requirements for 5.4

```
lemma Big-Red-5-4: ∀∞l. Big-Red-5-4 l
  unfolding Big-Red-5-4-def eventually-conj-iff all-imp-conj-distrib
  apply (simp add: Big-Red-5-6)
  apply (intro conjI eventually-all-ge-at-top; real-asymp)
  done
```

```
context Book
begin

lemma Red-5-4:
  assumes i: i ∈ Step-class {red-step,dboost-step}
    and big: Big-Red-5-4 l
  defines X ≡ Xseq i and Y ≡ Yseq i
  shows weight X Y (cvx i) ≥ − card X / (real k)^5
proof −
  have l≠1
    using big by (auto simp: Big-Red-5-4-def)
  with ln0 l-le-k have l>1 k>1 by linarith+
  let ?R = RN k (m-of l)
  have finite X finite Y
    by (auto simp: X-def Y-def finite-Xseq finite-Yseq)
  have not-many-bluish: ¬ many-bluish X
    using i not-many-bluish unfolding X-def by blast
  have nonterm: ¬ termination-condition X Y
    using X-def Y-def i step-non-terminating-iff by (force simp: Step-class-def)
  moreover have l powr (2/3) ≤ l powr (3/4)
    using ‹l>1› by (simp add: powr-mono)
  ultimately have RNX: ?R < card X
    unfolding termination-condition-def m-of-def
    by (meson RN-mono order.trans ceiling-mono le-refl nat-mono not-le)
  have 0 ≤ (∑ x ∈ X. ∑ x' ∈ X. Weight X Y x x')
   by (simp add: X-def Y-def sum-Weight-ge0 Xseq-subset-V Yseq-subset-V Xseq-Yseq-disjnt)
  also have ... = (∑ y ∈ X. weight X Y y + Weight X Y y y)
    unfolding weight-def X-def
    by (smt (verit) sum.cong sum.infinite sum.remove)
  finally have ge0: 0 ≤ (∑ y∈X. weight X Y y + Weight X Y y y) .
  have w-maximal: weight X Y (cvx i) ≥ weight X Y x
    if central-vertex X x for x
    using X-def Y-def ‹finite X› central-vx-is-best cvx-works i that by presburger

  have |real (card (S ∩ Y)) ∗ (real (card X) ∗ real (card Y)) −
          real (edge-card Red X Y) ∗ real (card (T ∩ Y))|
       ≤ real (card X) ∗ real (card Y) ∗ real (card Y) for S T
    using card-mono [OF - Int-lower2] ‹finite X› ‹finite Y›
     by (smt (verit, best) of-nat-mult edge-card-le mult.commute mult-right-mono
of-nat-0-le-iff of-nat-mono)
  then have W1abs: |Weight X Y x y| ≤ 1 for x y
    using RNX edge-card-le [of X Y Red] ‹finite X› ‹finite Y›
    apply (simp add: mult-ac Weight-def divide-simps gen-density-def)
    by (metis Int-lower2 card-mono mult-of-nat-commute)
  then have W1: Weight X Y x y ≤ 1 for x y
    by (smt (verit))
  have WW-le-cardX: weight X Y y + Weight X Y y y ≤ card X if y ∈ X for y
  proof −
    have weight X Y y + Weight X Y y y = sum (Weight X Y y) X
```

by (*simp add*: ‹*finite X*› *sum-diff1 that weight-def*)
also have ... ≤ *card X*
using *W1* **by** (*smt* (*verit*) *real-of-card sum-mono*)
finally show *?thesis* **.**
qed
have *weight X Y x* ≤ *real* (*card*(*X* − {*x*})) ∗ *1* **for** *x*
unfolding *weight-def* **by** (*meson DiffE abs-le-D1 sum-bounded-above W1*)
then have *wgt-le-X1*: *weight X Y x* ≤ *card X* − *1* **if** *x* ∈ *X* **for** *x*
using *that card-Diff-singleton One-nat-def* **by** (*smt* (*verit*, *best*))
define *XB* **where** *XB* ≡ {*x*∈*X*. *bluish X x*}
have *card-XB*: *card XB* < *?R*
using *not-many-bluish* **by** (*auto simp*: *m-of-def many-bluish-def XB-def*)
have *XB* ⊆ *X finite XB*
using ‹*finite X*› **by** (*auto simp*: *XB-def*)
then have *cv-non-XB*: ⋀*y*. *y* ∈ *X* − *XB* ⟹ *central-vertex X y*
by (*auto simp*: *central-vertex-def XB-def bluish-def*)
have *0* ≤ (∑*y*∈*X*. *weight X Y y* + *Weight X Y y y*)
by (*fact ge0*)
also have ... = (∑*y*∈*XB*. *weight X Y y* + *Weight X Y y y*) + (∑*y*∈*X*−*XB*. *weight X Y y* + *Weight X Y y y*)
using *sum.subset-diff* [*OF* ‹*XB*⊆*X*›] **by** (*smt* (*verit*) *X-def Xseq-subset-V finV finite-subset*)
also have ... ≤ (∑*y*∈*XB*. *weight X Y y* + *Weight X Y y y*) + (∑*y*∈*X*−*XB*. *weight X Y* (*cvx i*) + *1*)
by (*intro add-mono sum-mono w-maximal W1 order-refl cv-non-XB*)
also have ... = (∑*y*∈*XB*. *weight X Y y* + *Weight X Y y y*) + (*card X* − *card XB*) ∗ (*weight X Y* (*cvx i*) + *1*)
using ‹*XB*⊆*X*› ‹*finite XB*› **by** (*simp add*: *card-Diff-subset*)
also have ... ≤ *card XB* ∗ *card X* + (*card X* − *card XB*) ∗ (*weight X Y* (*cvx i*) + *1*)
using *sum-bounded-above WW-le-cardX*
by (*smt* (*verit*, *ccfv-threshold*) *XB-def mem-Collect-eq of-nat-mult*)
also have ... = *real* (*?R* ∗ *card X*) + (*real* (*card XB*) − *?R*) ∗ *card X* + (*card X* − *card XB*) ∗ (*weight X Y* (*cvx i*) + *1*)
using *card-XB* **by** (*simp add*: *algebra-simps flip*: *of-nat-mult of-nat-diff*)
also have ... ≤ *real* (*?R* ∗ *card X*) + (*card X* − *?R*) ∗ (*weight X Y* (*cvx i*) + *1*)
proof −
have (*real* (*card X*) − *card XB*) ∗ (*weight X Y* (*cvx i*) + *1*)
≤ (*real* (*card X*) − *?R*) ∗ (*weight X Y* (*cvx i*) + *1*) + (*real* (*?R*) − *card XB*) ∗ (*weight X Y* (*cvx i*) + *1*)
by (*simp add*: *algebra-simps*)
also have ... ≤ (*real* (*card X*) − *?R*) ∗ (*weight X Y* (*cvx i*) + *1*) + (*real* (*?R*) − *card XB*) ∗ *card X*
using *RNX X-def i card-XB cvx-in-Xseq wgt-le-X1* **by** *fastforce*
finally show *?thesis*
by (*smt* (*verit*, *del-insts*) *RNX* ‹*XB* ⊆ *X*› ‹*finite X*› *card-mono nat-less-le of-nat-diff distrib-right*)
qed

```
  finally have weight-ge-0: 0 ≤ ?R ∗ card X + (card X − ?R) ∗ (weight X Y
(cvx i) + 1) .
  have rk61: real k^6 > 1
    using ‹k>1› by simp
  have k267: real k + 2 ∗ real k^6 ≤ (real k^7)
    using ‹l ≤ k› big by (auto simp: Big-Red-5-4-def)
  have k-le: real k^6 + (?R ∗ real k + ?R ∗ (real k^6)) ≤ 1 + ?R ∗ (real k^7)
    using mult-left-mono [OF k267, of ?R] assms
   by (smt (verit, ccfv-SIG) distrib-left card-XB mult-le-cancel-right1 nat-less-real-le
of-nat-0-le-iff zero-le-power)
  have [simp]: real k^m = real k^n ⟷ m=n real k^m < real k^n ⟷ m<n for
m n
    using ‹1 < k› by auto
  have RN k (nat⌈l powr (3/4)⌉) ≥ k^6 ∗ ?R
    using ‹l ≤ k› big Red-5-6 by (auto simp: Big-Red-5-4-def)
  then have cardX-ge: card X ≥ k^6 ∗ ?R
    by (meson le-trans nat-le-linear nonterm termination-condition-def)
  have −1 / (real k)^5 ≤ − 1 / (real k^6 − 1) + −1 / (real k^6 ∗ ?R)
      using rk61 card-XB mult-left-mono [OF k-le, of real k^5]
      by (simp add: field-split-simps eval-nat-numeral)
  also have ... ≤ − ?R / (real k^6 ∗ ?R − ?R) + −1 / (real k^6 ∗ ?R)
    using card-XB rk61  by (simp add: field-split-simps)
  finally have −1 / (real k)^5 ≤ − ?R / (real k^6 ∗ ?R − ?R) + −1 / (real k^6
∗ ?R) .
  also have ... ≤ − ?R / (real (card X) − ?R) + −1 / card X
  proof (intro add-mono divide-left-mono-neg)
    show real k^6 ∗ real ?R − real ?R ≤ real (card X) − real ?R
      using cardX-ge of-nat-mono by fastforce
    show real k^6 ∗ real ?R ≤ real (card X)
      using cardX-ge of-nat-mono by fastforce
  qed (use RNX rk61 kn0 card-XB in auto)
  also have ... ≤ weight X Y (cvx i) / card X
   using RNX mult-left-mono [OF weight-ge-0, of card X] by (simp add: field-split-simps)
  finally show ?thesis
    using RNX by (simp add: X-def Y-def divide-simps)
qed

lemma Red-5-7a: ε / k ≤ alpha (hgt p)
  by (simp add: alpha-ge hgt-gt0)

lemma Red-5-7b:
  assumes p ≥ qfun 0 shows alpha (hgt p) ≤ ε ∗ (p − qfun 0 + 1/k)
proof −
  have qh-le-p: qfun (hgt p − Suc 0) ≤ p
   by (smt (verit) assms diff-Suc-less hgt-gt0 hgt-less-imp-qfun-less zero-less-iff-neq-zero)
  have alpha (hgt p) = ε ∗ (1 + ε)^(hgt p − 1) / k
    using alpha-eq alpha-hgt-eq by blast
  also have ... = ε ∗ (qfun (hgt p − 1) − qfun 0 + 1/k)
    by (simp add: diff-divide-distrib qfun-eq)
```

also have . . . ≤ *ε* ∗ (*p* − *qfun 0* + *1/k*)
by (*simp add*: *eps-ge0 mult-left-mono qh-le-p*)
finally show *?thesis* **.**
qed

lemma *Red-5-7c*:
assumes *p* ≤ *qfun 1* **shows** *alpha* (*hgt p*) = *ε* / *k*
using *alpha-hgt-eq Book-axioms assms hgt-Least* **by** *fastforce*

lemma *Red-5-8*:
assumes *i*: *i* ∈ *Step-class* {*dreg-step*} **and** *x*: *x* ∈ *Xseq* (*Suc i*)
shows *card* (*Neighbours Red x* ∩ *Yseq* (*Suc i*))
≥ (*1* − *ε powr* (*1/2*)) ∗ *pseq i* ∗ (*card* (*Yseq* (*Suc i*)))
proof −
obtain *X Y A B*
where *step*: *stepper i* = (*X*,*Y*,*A*,*B*)
and *nonterm*: ¬ *termination-condition X Y*
and *even i*
and *Suc-i*: *stepper* (*Suc i*) = *degree-reg* (*X*,*Y*,*A*,*B*)
and *XY*: *X* = *Xseq i Y* = *Yseq i*
using *i* **by** (*auto simp*: *step-kind-defs split*: *if-split-asm prod.split-asm*)
have *Xseq* (*Suc i*) = ((λ(*X*, *Y*, *A*, *B*). *X*) ∘ *stepper*) (*Suc i*)
by (*simp add*: *Xseq-def*)
also have . . . = *X-degree-reg X Y*
using ‹*even i*› *step nonterm* **by** (*auto simp*: *degree-reg-def*)
finally have *XSuc*: *Xseq* (*Suc i*) = *X-degree-reg X Y* **.**
have *YSuc*: *Yseq* (*Suc i*) = *Yseq i*
using *Suc-i step* **by** (*auto simp*: *degree-reg-def stepper-XYseq*)
have *p-gt-invk*: (*pseq i*) > *1/k*
using *XY nonterm pseq-def termination-condition-def* **by** *auto*
have *RedN*: (*pseq i* − *ε powr* −(*1/2*) ∗ *alpha* (*hgt* (*pseq i*))) ∗ *card Y* ≤ *card* (*Neighbours Red x* ∩ *Y*)
using *x XY* **by** (*simp add*: *XSuc YSuc X-degree-reg-def pseq-def red-dense-def*)
show *?thesis*
proof (*cases pseq i* ≥ *qfun 0*)
case *True*
have *i* ∉ *Step-class* {*halted*}
using *i* **by** (*simp add*: *Step-class-def*)
then have *p0*: *1/k* < *p0*
by (*metis Step-class-not-halted gr0I nat-less-le not-halted-pee-gt pee-eq-p0*)
have *0*: *ε powr* −(*1/2*) ≥ *0*
by *simp*
have *ε powr* −(*1/2*) ∗ *alpha* (*hgt* (*pseq i*)) ≤ *ε powr* (*1/2*) ∗ ((*pseq i*) − *qfun 0* + *1/k*)
using *mult-left-mono* [*OF Red-5-7b* [*OF True*] *0*]
by (*simp add*: *eps-def powr-mult-base flip*: *mult-ac*)
also have . . . ≤ *ε powr* (*1/2*) ∗ (*pseq i*)
using *p0* **by** (*intro mult-left-mono*) (*auto simp flip*: *pee-eq-p0*)
finally have *ε powr* −(*1/2*) ∗ *alpha* (*hgt* (*pseq i*)) ≤ *ε powr* (*1/2*) ∗ (*pseq i*) **.**

then have (*1* − ε *powr* (*1*/*2*)) ∗ (*pseq i*) ∗ (*card Y*) ≤ ((*pseq i*) − ε *powr* −(*1*/*2*) ∗ *alpha* (*hgt* (*pseq i*))) ∗ *card Y*
  by (*intro mult-right-mono*) (*auto simp*: *algebra-simps*)
  with *XY RedN YSuc* **show** *?thesis* **by** *fastforce*
next
  case *False*
  then have *pseq i* ≤ *qfun 1*
    by (*smt* (*verit*) *One-nat-def alpha-Suc-eq alpha-ge0 q-Suc-diff*)
  then have ε *powr* −(*1*/*2*) ∗ *alpha* (*hgt* (*pseq i*)) = ε *powr* (*1*/*2*) / *k*
    using *powr-mult-base* [*of* ε] *eps-gt0* **by** (*force simp*: *Red-5-7c mult.commute*)
  also have ... ≤ ε *powr* (*1*/*2*) ∗ (*pseq i*)
    using *p-gt-invk*
    by (*smt* (*verit*) *divide-inverse inverse-eq-divide mult-left-mono powr-ge-zero*)
  finally have ε *powr* −(*1*/*2*) ∗ *alpha* (*hgt* (*pseq i*)) ≤ ε *powr* (*1*/*2*) ∗ (*pseq i*) **.**
  then have (*1* − ε *powr* (*1*/*2*)) ∗ *pseq i* ∗ *card Y* ≤ (*pseq i* − ε *powr* −(*1*/*2*) ∗ *alpha* (*hgt* (*pseq i*))) ∗ *card Y*
    by (*intro mult-right-mono*) (*auto simp*: *algebra-simps*)
  with *XY RedN YSuc* **show** *?thesis* **by** *fastforce*
qed
qed

corollary *Y-Neighbours-nonempty-Suc*:
  assumes *i*: *i* ∈ *Step-class* {*dreg-step*} **and** *x*: *x* ∈ *Xseq* (*Suc i*) **and** *k*≥*2*
  shows *Neighbours Red x* ∩ *Yseq* (*Suc i*) ≠ {}
proof
  assume *con*: *Neighbours Red x* ∩ *Yseq* (*Suc i*) = {}
  have *not-halted*: *i* ∉ *Step-class* {*halted*}
    using *i* **by** (*auto simp*: *Step-class-def*)
  then have *0*: *pseq i* > *0*
    using *not-halted-pee-gt0* **by** *blast*
  have *Y'*: *card* (*Yseq* (*Suc i*)) > *0*
    using *i Yseq-gt0* [*OF not-halted*] *stepper-XYseq*
    by (*auto simp*: *step-kind-defs degree-reg-def split*: *if-split-asm prod.split-asm*)
  have (*1* − ε *powr* (*1*/*2*)) ∗ *pseq i* ∗ *card* (*Yseq* (*Suc i*)) ≤ *0*
    using *Red-5-8* [*OF i x*] *con* **by** *simp*
  with *0 Y'* **have** (*1* − ε *powr* (*1*/*2*)) ≤ *0*
    by (*simp add*: *mult-le-0-iff zero-le-mult-iff*)
  then show *False*
    using ‹*k*≥*2*› *powr-le-cancel-iff* [*of k 1*/*8 0*]
    by (*simp add*: *eps-def powr-minus-divide powr-divide powr-powr*)
qed

corollary *Y-Neighbours-nonempty*:
  assumes *i*: *i* ∈ *Step-class* {*red-step*,*dboost-step*} **and** *x*: *x* ∈ *Xseq i* **and** *k*≥*2*
  shows *card* (*Neighbours Red x* ∩ *Yseq i*) > *0*
proof (*cases i*)
  case *0*
  with *assms* **show** *?thesis*
    by (*auto simp*: *Step-class-def stepper-kind-def split*: *if-split-asm*)

next
  case (*Suc i'*)
  then have $i' \in$ *Step-class* {*dreg-step*}
    by (*metis dreg-before-step dreg-before-step i Step-class-insert Un-iff*)
  then have *Neighbours Red x* ∩ *Yseq* (*Suc i'*) ≠ {}
    using *Suc Y-Neighbours-nonempty-Suc assms* **by** *blast*
  then show *?thesis*
    by (*simp add*: *Suc card-gt-0-iff finite-Neighbours*)
qed

end

4.3 Lemma 5.1

definition *Big-Red-5-1* ≡ λμ *l*. (*1*−μ) ∗ *real l* > *1* ∧ *l powr* (*5/2*) ≥ *3* / (*1*−μ)
∧ *l powr* (*1/4*) ≥ *4*
    ∧ *Big-Red-5-4 l* ∧ *Big-Red-5-6 l*

establishing the size requirements for 5.1

lemma *Big-Red-5-1*:
  assumes μ*1*<*1*
  shows ∀∞*l*. ∀μ. μ ∈ {μ*0*..μ*1*} ⟶ *Big-Red-5-1* μ *l*
proof −
  have (∀∞*l*. ∀μ. μ*0* ≤ μ ∧ μ ≤ μ*1* ⟶ *1* < (*1*−μ) ∗ *real l*)
  proof (*intro eventually-all-geI1*)
    show ⋀*l* μ. ⟦*1* < (*1*−μ*1*) ∗ *real l*; μ ≤ μ*1*⟧ ⟹ *1* < (*1*−μ) ∗ *l*
      by (*smt* (*verit*, *best*) *mult-right-mono of-nat-0-le-iff*)
  qed (*use assms* **in** *real-asymp*)
  moreover have (∀∞*l*. ∀μ. μ*0* ≤ μ ∧ μ ≤ μ*1* ⟶ *3* / (*1*−μ) ≤ *real l powr*
(*5/2*))
  proof (*intro eventually-all-geI1*)
    show ⋀*l* μ. ⟦*3* / (*1*−μ*1*) ≤ *real l powr* (*5/2*); μ ≤ μ*1*⟧
        ⟹ *3* / (*1*−μ) ≤ *real l powr* (*5/2*)
      by (*smt* (*verit*, *ccfv-SIG*) *assms frac-le*)
  qed (*use assms* **in** *real-asymp*)
  moreover have ∀∞*l*. *4* ≤ *real l powr* (*1* / *4*)
    by *real-asymp*
  ultimately show *?thesis*
  using *assms Big-Red-5-6 Big-Red-5-4* **by** (*auto simp*: *Big-Red-5-1-def all-imp-conj-distrib*
eventually-conj-iff)
qed

context *Book*
begin

lemma *card-cvx-Neighbours*:
  assumes *i*: *i* ∈ *Step-class* {*red-step*,*dboost-step*}
  defines *x* ≡ *cvx i*
  defines *X* ≡ *Xseq i*
  defines *NBX* ≡ *Neighbours Blue x* ∩ *X*

defines *NRX* ≡ *Neighbours Red x* ∩ *X*
shows *card NBX* ≤ *μ* ∗ *card X card NRX* ≥ (*1*−*μ*) ∗ *card X* − *1*
proof −
obtain *x*∈*X X*⊆*V*
by (*metis Xseq-subset-V cvx-in-Xseq X-def i x-def*)
then have *card-NRBX*: *card NRX* + *card NBX* = *card X* − *1*
using *Neighbours-RB* [*of x X*] *disjnt-Red-Blue-Neighbours*
by (*simp add*: *NRX-def NBX-def finite-Neighbours subsetD flip*: *card-Un-disjnt*)
moreover have *card-NBX-le*: *card NBX* ≤ *μ* ∗ *card X*
by (*metis cvx-works NBX-def X-def central-vertex-def i x-def*)
ultimately show *card NBX* ≤ *μ* ∗ *card X card NRX* ≥ (*1*−*μ*) ∗ *card X* − *1*
by (*auto simp*: *algebra-simps*)
qed

lemma *Red-5-1*:
assumes *i*: *i* ∈ *Step-class* {*red-step*,*dboost-step*}
and *Big*: *Big-Red-5-1 μ l*
defines *p* ≡ *pseq i*
defines *x* ≡ *cvx i*
defines *X* ≡ *Xseq i* **and** *Y* ≡ *Yseq i*
defines *NBX* ≡ *Neighbours Blue x* ∩ *X*
defines *NRX* ≡ *Neighbours Red x* ∩ *X*
defines *NRY* ≡ *Neighbours Red x* ∩ *Y*
defines *β* ≡ *card NBX* / *card X*
shows *red-density NRX NRY* ≥ *p* − *alpha* (*hgt p*)
∨ *red-density NBX NRY* ≥ *p* + (*1* − *ε*) ∗ ((*1*−*β*) / *β*) ∗ *alpha* (*hgt p*) ∧ *β* > *0*
proof −
have *Red-5-4*: *weight X Y x* ≥ − *real* (*card X*) / (*real k*)^*5*
using *Big i Red-5-4* **by** (*auto simp*: *Big-Red-5-1-def x-def X-def Y-def*)
have *lA*: (*1*−*μ*) ∗ *l* > *1* **and** *l*≤*k* **and** *l144*: *l powr* (*1*/*4*) ≥ *4*
using *Big* **by** (*auto simp*: *Big-Red-5-1-def l-le-k*)
then have *k-powr-14*: *k powr* (*1*/*4*) ≥ *4*
by (*smt* (*verit*) *divide-nonneg-nonneg of-nat-0-le-iff of-nat-mono powr-mono2*)
have *k* ≥ *256*
using *powr-mono2* [*of 4*, *OF* - - *k-powr-14*] **by** (*simp add*: *powr-powr flip*: *powr-numeral*)
then have *k*>*0* **by** *linarith*
have *k52*: *3* / (*1*−*μ*) ≤ *k powr* (*5*/*2*)
using *Big* ‹*l*≤*k*› **unfolding** *Big-Red-5-1-def*
by (*smt* (*verit*) *of-nat-0-le-iff of-nat-mono powr-mono2 zero-le-divide-iff*)
have *RN-le-RN*: *k*^*6* ∗ *RN k* (*m-of l*) ≤ *RN k* (*nat* ⌈*l powr* (*3*/*4*)⌉)
using *Big* ‹*l* ≤ *k*› *Red-5-6* **by** (*auto simp*: *Big-Red-5-1-def*)
have *l34-ge3*: *l powr* (*3*/*4*) ≥ *3*
by (*smt* (*verit*, *ccfv-SIG*) *l144 divide-nonneg-nonneg frac-le of-nat-0-le-iff powr-le1 powr-less-cancel*)
note *XY* = *X-def Y-def*
obtain *A B*
where *step*: *stepper i* = (*X*,*Y*,*A*,*B*)

and *nonterm*: ¬ *termination-condition X Y*
and *odd i*
and *non-mb*: ¬ *many-bluish X* **and** *card X* > *0*
and *not-halted*: *i* ∉ *Step-class* {*halted*}
using *i* **by** (*auto simp*: *XY step-kind-defs termination-condition-def split*: *if-split-asm prod.split-asm*)
with *Yseq-gt0 XY* **have** *card Y* ≠ *0*
by *blast*
have *cX-RN*: *card X* > *RN k* (*nat* ⌈*l powr* (*3/4*)⌉)
by (*meson linorder-not-le nonterm termination-condition-def*)
then have *X-gt-k*: *card X* > *k*
by (*metis l34-ge3 RN-3plus′ of-nat-numeral order.trans le-natceiling-iff not-less*)
have *0* < *RN k* (*m-of l*)
using *RN-eq-0-iff m-of-def many-bluish-def non-mb* **by** *presburger*
then have *k^4* ≤ *k^6* ∗ *RN k* (*m-of l*)
by (*simp add*: *eval-nat-numeral*)
also have . . . < *card X*
using *cX-RN RN-le-RN* **by** *linarith*
finally have *card X* > *k^4* **.**
have *x* ∈ *X*
using *cvx-in-Xseq i XY x-def* **by** *blast*
have *X* ⊆ *V*
by (*simp add*: *Xseq-subset-V XY*)
have *finite NRX finite NBX finite NRY*
by (*auto simp*: *NRX-def NBX-def NRY-def finite-Neighbours*)
have *disjnt X Y*
using *Xseq-Yseq-disjnt step stepper-XYseq* **by** *blast*
then have *disjnt NRX NRY disjnt NBX NRY*
by (*auto simp*: *NRX-def NBX-def NRY-def disjnt-iff*)
have *card-NRBX*: *card NRX* + *card NBX* = *card X* − *1*
using *Neighbours-RB* [*of x X*] ‹*finite NRX*› ‹*x*∈*X*› ‹*X*⊆*V*› *disjnt-Red-Blue-Neighbours*
by (*simp add*: *NRX-def NBX-def finite-Neighbours subsetD flip*: *card-Un-disjnt*)
obtain *card-NBX-le*: *card NBX* ≤ *μ* ∗ *card X* **and** *card NRX* ≥ (*1*−*μ*) ∗ *card X* − *1*
unfolding *NBX-def NRX-def X-def x-def* **using** *card-cvx-Neighbours i* **by** *metis*
with *lA* ‹*l*≤*k*› *X-gt-k* **have** *card NRX* > *0*
by (*smt* (*verit*, *best*) *of-nat-0 μ01 gr0I mult-less-cancel-left-pos nat-less-real-le of-nat-mono*)
have *card NRY* > *0*
using *Y-Neighbours-nonempty* [*OF i*] ‹*k*≥*256*› *NRY-def* ‹*finite NRY*› ‹*x* ∈ *X*› *card-0-eq XY* **by** *force*
show *?thesis*
proof (*cases* (∑*y*∈*NRX*. *Weight X Y x y*) ≥ −*alpha* (*hgt p*) ∗ *card NRX* ∗ *card NRY* / *card Y*)
case *True*
then have (*p* − *alpha* (*hgt p*)) ∗ (*card NRX* ∗ *card NRY*) ≤ (∑ *y* ∈ *NRX*. *p* ∗ *card NRY* + *Weight X Y x y* ∗ *card Y*)
using ‹*card Y* ≠ *0*› **by** (*simp add*: *field-simps sum-distrib-left sum.distrib*)
also have . . . = (∑ *y* ∈ *NRX*. *card* (*Neighbours Red x* ∩ *Neighbours Red y* ∩

Y))
using ‹*card Y* ≠ *0*› **by** (*simp add*: *Weight-def pseq-def XY NRY-def field-simps p-def*)
also have ... = *edge-card Red NRY NRX*
using ‹*disjnt NRX NRY*› ‹*finite NRX*›
by (*simp add*: *disjnt-sym edge-card-eq-sum-Neighbours Red-E psubset-imp-subset NRY-def Int-ac*)
also have ... = *edge-card Red NRX NRY*
by (*simp add*: *edge-card-commute*)
finally have (*p* − *alpha* (*hgt p*)) ∗ *real* (*card NRX* ∗ *card NRY*) ≤ *real* (*edge-card Red NRX NRY*) **.**
then show *?thesis*
using ‹*card NRX* > *0*› ‹*card NRY* > *0*›
by (*simp add*: *NRX-def NRY-def gen-density-def field-split-simps XY*)
next
case *False*
have *x* ∈ *X*
unfolding *x-def* **using** *cvx-in-Xseq i XY* **by** *blast*
with *Neighbours-RB*[*of x X*] **have** *Xx*: *X* − {*x*} = *NBX* ∪ *NRX*
using *Xseq-subset-V NRX-def NBX-def XY* **by** *blast*
have *disjnt*: *NBX* ∩ *NRX* = {}
by (*auto simp*: *Blue-eq NRX-def NBX-def disjoint-iff in-Neighbours-iff*)
then have *weight X Y x* = (∑*y* ∈ *NRX*. *Weight X Y x y*) + (∑*y* ∈ *NBX*. *Weight X Y x y*)
by (*simp add*: *weight-def Xx sum.union-disjoint finite-Neighbours NRX-def NBX-def*)
with *False*
have *15*: (∑*y* ∈ *NBX*. *Weight X Y x y*)
≥ *weight X Y x* + *alpha* (*hgt p*) ∗ *card NRX* ∗ *card NRY* / *card Y*
by *linarith*
have *pm1*: *pseq* (*i*−*1*) > *1*/*k*
by (*meson Step-class-not-halted diff-le-self not-halted not-halted-pee-gt*)
have *β-eq*: *β* = *card NBX* / *card X*
using *NBX-def β-def XY* **by** *blast*
have *β*≤*μ*
by (*simp add*: *β-eq* ‹*0* < *card X*› *card-NBX-le pos-divide-le-eq*)
have *im1*: *i*−*1* ∈ *Step-class* {*dreg-step*}
using *i* ‹*odd i*› *dreg-before-step*
by (*metis Step-class-insert Un-iff One-nat-def odd-Suc-minus-one*)
have *ε* ≤ *1*/*4*
using ‹*k*>*0*› *k-powr-14* **by** (*simp add*: *eps-def powr-minus-divide*)
then have *ε powr* (*1*/*2*) ≤ (*1*/*4*) *powr* (*1*/*2*)
by (*simp add*: *eps-def powr-mono2*)
then have *A*: *1*/*2* ≤ *1* − *ε powr* (*1*/*2*)
by (*simp add*: *powr-divide*)
have *le*: *1* / (*2* ∗ *real k*) ≤ (*1* − *ε powr* (*1*/*2*)) ∗ *pseq* (*i*−*1*)
using *pm1* ‹*k*>*0*› *mult-mono* [*OF A less-imp-le* [*OF pm1*]] *A* **by** *simp*
have *card Y* / (*2* ∗ *real k*) ≤ (*1* − *ε powr* (*1*/*2*)) ∗ *pseq* (*i*−*1*) ∗ *card Y*
using *mult-left-mono* [*OF le*] **by** (*metis mult.commute divide-inverse inverse-eq-divide*

of-nat-0-le-iff)
  also have ... ≤ *card NRY*
    using *pm1 Red-5-8 im1* **by** (*metis NRY-def One-nat-def* ‹*odd i*› ‹$x \in X$› *XY odd-Suc-minus-one*)
  finally have *Y-NRY*: *card Y* / (*2* ∗ *real k*) ≤ *card NRY* **.**
  have *NBX* ≠ {}
  proof
    assume *empty*: *NBX* = {}
    then have *cNRX*: *card NRX* = *card X* − *1*
      using *card-NRBX* **by** *auto*
    have *card X* > *3*
      using ‹*k*≥*256*› *X-gt-k* **by** *linarith*
    then have *2* ∗ *card X* / *real* (*card X* − *1*) < *3*
      by (*simp add*: *divide-simps*)
    also have ... ≤ *k*^*2*
      using *mult-mono* [*OF* ‹*k*≥*256*› ‹*k*≥*256*›] **by** (*simp add*: *power2-eq-square flip*: *of-nat-mult*)
    also have ... ≤ ε ∗ *k*^*3*
      using ‹*k*≥*256*› **by** (*simp add*: *eps-def flip*: *powr-numeral powr-add*)
    finally have (*real* (*2* ∗ *card X*) / *real* (*card X* − *1*)) ∗ *k*^*2* < ε ∗ *real* (*k*^*3*) ∗ *k*^*2*
      using ‹*k*>*0*› **by** (*intro mult-strict-right-mono*) *auto*
    then have *real* (*2* ∗ *card X*) / *real* (*card X* − *1*) ∗ *k*^*2* < ε ∗ *real* (*k*^*5*)
      by (*simp add*: *mult.assoc flip*: *of-nat-mult*)
    then have *0* < − *real* (*card X*) / (*real k*)^*5* + (ε / *k*) ∗ *real* (*card X* − *1*) ∗ (*1* / (*2* ∗ *real k*))
      using ‹*k*>*0*› *X-gt-k* **by** (*simp add*: *field-simps power2-eq-square*)
    also have − *real* (*card X*) / (*real k*)^*5* + (ε / *k*) ∗ *real* (*card X* − *1*) ∗ (*1* / (*2* ∗ *real k*))
        ≤ − *real* (*card X*) / (*real k*)^*5* + (ε / *k*) ∗ *real* (*card NRX*) ∗ (*card NRY* / *card Y*)
      using *Y-NRY* ‹*k*>*0*› ‹*card Y* ≠ *0*›
      by (*intro add-mono mult-mono*) (*auto simp*: *cNRX eps-def divide-simps*)
    also have ... = − *real* (*card X*) / (*real k*)^*5* + (ε / *k*) ∗ *real* (*card NRX*) ∗ *card NRY* / *card Y*
      by *simp*
    also have ... ≤ − *real* (*card X*) / (*real k*)^*5* + *alpha* (*hgt p*) ∗ *real* (*card NRX*) ∗ *card NRY* / *card Y*
      using *alpha-ge* [*OF hgt-gt0*]
      by (*intro add-mono mult-right-mono divide-right-mono*) *auto*
    also have ... ≤ *0*
      using *empty 15 Red-5-4* **by** *auto*
    finally show *False*
      by *simp*
  qed
  have *card NBX* > *0*
    by (*simp add*: ‹*NBX* ≠ {}› ‹*finite NBX*› *card-gt-0-iff*)
  then have *0* < β
    by (*simp add*: β*-eq* ‹*0* < *card X*›)

have $\beta \leq \mu$
  using *X-gt-k card-NBX-le* **by** (*simp add*: *β-eq NBX-def divide-simps*)
have *cNRX*: *card NRX* = (*1*−β) ∗ *card X* − *1*
  using *X-gt-k card-NRBX* **by** (*simp add*: *β-eq divide-simps*)
have *cNBX*: *card NBX* = β ∗ *card X*
  using ‹*0* < *card X*› **by** (*simp add*: *β-eq*)
let *?E16* = *p* + ((*1*−β)/β) ∗ *alpha* (*hgt p*) − *alpha* (*hgt p*) / (β ∗ *card X*) + *weight X Y x* ∗ *card Y* / (β ∗ *card X* ∗ *card NRY*)
have *p* ∗ *card NBX* ∗ *card NRY* + *alpha* (*hgt p*) ∗ *card NRX* ∗ *card NRY* + *weight X Y x* ∗ *card Y*
    ≤ (∑*y* ∈ *NBX*. *p* ∗ *card NRY* + *Weight X Y x y* ∗ *card Y*)
  using *15* ‹*card Y* ≠ *0*› **apply** (*simp add*: *sum-distrib-left sum.distrib*)
  by (*simp only*: *sum-distrib-right divide-simps split*: *if-split-asm*)
also have ... ≤ (∑*y* ∈ *NBX*. *card* (*Neighbours Red x* ∩ *Neighbours Red y* ∩ *Y*))
  using ‹*card Y* ≠ *0*› **by** (*simp add*: *Weight-def pseq-def XY NRY-def field-simps p-def*)
also have ... = *edge-card Red NRY NBX*
  using ‹*disjnt NBX NRY*› ‹*finite NBX*›
  by (*simp add*: *disjnt-sym edge-card-eq-sum-Neighbours Red-E psubset-imp-subset NRY-def Int-ac*)
also have ... = *edge-card Red NBX NRY*
  by (*simp add*: *edge-card-commute*)
finally have *Red-bound*:
  p ∗ *card NBX* ∗ *card NRY* + *alpha* (*hgt p*) ∗ *card NRX* ∗ *card NRY* + *weight X Y x* ∗ *card Y* ≤ *edge-card Red NBX NRY* **.**
then have (*p* ∗ *card NBX* ∗ *card NRY* + *alpha* (*hgt p*) ∗ *card NRX* ∗ *card NRY* + *weight X Y x* ∗ *card Y*)
    / (*card NBX* ∗ *card NRY*) ≤ *red-density NBX NRY*
  by (*metis divide-le-cancel gen-density-def of-nat-less-0-iff*)
then have *p* + *alpha* (*hgt p*) ∗ *card NRX* / *card NBX* + *weight X Y x* ∗ *card Y* / (*card NBX* ∗ *card NRY*) ≤ *red-density NBX NRY*
  using ‹*card NBX* > *0*› ‹*card NRY* > *0*› **by** (*simp add*: *add-divide-distrib*)
then have *16*: *?E16* ≤ *red-density NBX NRY*
  using ‹β>*0*› ‹*card X* > *0*›
  by (*simp add*: *cNRX cNBX algebra-simps add-divide-distrib diff-divide-distrib*)
consider *qfun 0* ≤ *p* | *p* ≤ *qfun 1*
  by (*smt* (*verit*) *alpha-Suc-eq alpha-ge0 One-nat-def q-Suc-diff*)
then have *alpha-le-1*: *alpha* (*hgt p*) ≤ *1*
proof *cases*
  case *1*
  have *p* ∗ ε + ε / *real k* ≤ *1* + ε ∗ *p0*
  proof (*intro add-mono*)
    show *p* ∗ ε ≤ *1*
      by (*smt* (*verit*) *eps-le1* ‹*0* < *k*› *mult-left-le p-def pee-ge0 pee-le1*)
    have *p0* > *1*/*k*
      by (*metis Step-class-not-halted diff-le-self not-halted not-halted-pee-gt diff-is-0-eq′ pee-eq-p0*)
    then show ε / *real k* ≤ ε ∗ *p0*

by (*metis divide-inverse eps-ge0 mult-left-mono less-eq-real-def mult-cancel-right1*)
qed
then show *?thesis*
using *Red-5-7b* [*OF 1*] **by** (*simp add*: *algebra-simps*)
next
case *2*
show *?thesis*
using *Red-5-7c* [*OF 2*] ‹*k≥256*› *eps-less1* **by** *simp*
qed
have *B*: − (*3* / (*real k^4*)) ≤ (−*2* / *real k^4*) − *alpha* (*hgt p*) / *card X*
using ‹*card X* > *k^4*› ‹*card Y* ≠ *0*› ‹*0* < *k*› *alpha-le-1* **by** (*simp add*: *algebra-simps frac-le*)
have − (*3* / (*β* ∗ *real k^4*)) ≤ (−*2* / *real k^4*) / *β* − *alpha* (*hgt p*) / (*β* ∗ *card X*)
using ‹*β*>*0*› *divide-right-mono* [*OF B*, *of β*] ‹*k*>*0*› **by** (*simp add*: *field-simps*)
also have ... = (− *real* (*card X*) / *real k^5*) ∗ *card Y* / (*β* ∗ *real* (*card X*) ∗ (*card Y* / (*2* ∗ *real k*))) − *alpha* (*hgt p*) / (*β* ∗ *card X*)
using ‹*card Y* ≠ *0*› ‹*0* < *card X*›
by (*simp add*: *field-split-simps eval-nat-numeral*)
also have ... ≤ (− *real* (*card X*) / *real k^5*) ∗ *card Y* / (*β* ∗ *real* (*card X*) ∗ *card NRY*) − *alpha* (*hgt p*) / (*β* ∗ *card X*)
using *Y-NRY* ‹*k*>*0*› ‹*card NRY* > *0*› ‹*card X* > *0*› ‹*card Y* ≠ *0*› ‹*β*>*0*›
by (*intro diff-mono divide-right-mono mult-left-mono divide-left-mono-neg*) *auto*
also have ... ≤ *weight X Y x* ∗ *card Y* / (*β* ∗ *real* (*card X*) ∗ *card NRY*) − *alpha* (*hgt p*) / (*β* ∗ *card X*)
using *Red-5-4* ‹*k*>*0*› ‹*0* < *β*›
by (*intro diff-mono divide-right-mono mult-right-mono*) *auto*
finally have − (*3* / (*β* ∗ *real k^4*)) ≤ *weight X Y x* ∗ *card Y* / (*β* ∗ *real* (*card X*) ∗ *card NRY*) − *alpha* (*hgt p*) / (*β* ∗ *card X*) **.**
then have *17*: *p* + ((*1*−*β*)/*β*) ∗ *alpha* (*hgt p*) − *3* / (*β* ∗ *real k^4*) ≤ *?E16*
by *simp*
have *3* / *real k^4* ≤ (*1*−*μ*) ∗ *ε^2* / *k*
using ‹*k*>*0*› *μ01 mult-left-mono* [*OF k52*, *of k*]
by (*simp add*: *field-simps eps-def powr-powr powr-mult-base flip*: *powr-numeral powr-add*)
also have ... ≤ (*1*−*β*) ∗ *ε^2* / *k*
using ‹*β*≤*μ*›
by (*intro divide-right-mono mult-right-mono*) *auto*
also have ... ≤ (*1*−*β*) ∗ *ε* ∗ *alpha* (*hgt p*)
using *Red-5-7a* [*of p*] *eps-ge0* ‹*β*≤*μ*› *μ01*
unfolding *power2-eq-square divide-inverse mult.assoc*
by (*intro mult-mono*) *auto*
finally have †: *3* / *real k^4* ≤ (*1*−*β*) ∗ *ε* ∗ *alpha* (*hgt p*) **.**
have *p* + (*1* − *ε*) ∗ ((*1*−*β*) / *β*) ∗ *alpha* (*hgt p*) + *3* / (*β* ∗ *real k^4*) ≤ *p* + ((*1*−*β*)/*β*) ∗ *alpha* (*hgt p*)
using ‹*0*<*β*› ‹*k*>*0*› *mult-left-mono* [*OF* †, *of β*] **by** (*simp add*: *field-simps*)
with *16 17* **have** *p* + (*1* − *ε*) ∗ ((*1* − *β*) / *β*) ∗ *alpha* (*hgt p*) ≤ *red-density NBX NRY*

```
    by linarith
  then show ?thesis
    using ‹0 < β› NBX-def NRY-def XY by fastforce
 qed
qed
```

This and the previous result are proved under the assumption of a sufficiently large l

```
corollary Red-5-2:
 assumes i: i ∈ Step-class {dboost-step}
  and Big: Big-Red-5-1 μ l
 shows pseq (Suc i) − pseq i ≥ (1 − ε) ∗ ((1 − beta i) / beta i) ∗ alpha (hgt
(pseq i)) ∧
     beta i > 0
proof −
 let ?x = cvx i
 obtain X Y A B
  where step: stepper i = (X,Y,A,B)
   and nonterm: ¬ termination-condition X Y
   and odd i
   and non-mb: ¬ many-bluish X
   and nonredd: ¬ reddish k X Y (red-density X Y) (choose-central-vx (X,Y,A,B))
   and Xeq: X = Xseq i and Yeq: Y = Yseq i
  using i
  by (auto simp: step-kind-defs split: if-split-asm prod.split-asm)
 then have ?x ∈ Xseq i
  by (simp add: choose-central-vx-X cvx-def finite-Xseq)
 then have central-vertex (Xseq i) (cvx i)
  by (metis Xeq choose-central-vx-works cvx-def finite-Xseq step non-mb nonterm)
 with Xeq have card (Neighbours Blue (cvx i) ∩ Xseq i) ≤ μ ∗ card (Xseq i)
  by (simp add: central-vertex-def)
 then have βeq: card (Neighbours Blue (cvx i) ∩ Xseq i) = beta i ∗ card (Xseq
i)
  using Xeq step by (auto simp: beta-def)
 have SUC: stepper (Suc i) = (Neighbours Blue ?x ∩ X, Neighbours Red ?x ∩
Y, A, insert ?x B)
  using step nonterm ‹odd i› non-mb nonredd
  by (simp add: stepper-def next-state-def Let-def cvx-def)
 have pseq: pseq i = red-density X Y
  by (simp add: pseq-def Xeq Yeq)
 have choose-central-vx (X,Y,A,B) = cvx i
  by (simp add: cvx-def step)
 with nonredd have red-density (Neighbours Red (cvx i) ∩ X) (Neighbours Red
(cvx i) ∩ Y)
           < pseq i − alpha (hgt (red-density X Y))
  using nonredd by (simp add: reddish-def pseq)
 then have pseq i + (1 − ε) ∗ ((1 − beta i) / beta i) ∗ alpha (hgt (pseq i))
     ≤ red-density (Neighbours Blue (cvx i) ∩ Xseq i)
       (Neighbours Red (cvx i) ∩ Yseq i) ∧ beta i > 0
```

```
    using Red-5-1 Un-iff Xeq Yeq assms gen-density-ge0 pseq Step-class-insert
    by (smt (verit, ccfv-threshold) βeq divide-eq-eq)
  moreover have red-density (Neighbours Blue (cvx i) ∩ Xseq i)
                        (Neighbours Red (cvx i) ∩ Yseq i) ≤ pseq (Suc i)
    using SUC Xeq Yeq stepper-XYseq by (simp add: pseq-def)
  ultimately show ?thesis
    by linarith
qed

end
```

4.4 Lemma 5.3

This is a weaker consequence of the previous results

```
definition
  Big-Red-5-3 ≡
    λμ l. Big-Red-5-1 μ l
       ∧ (∀k≥l. k>1 ∧ 1 / (real k)^2 ≤ μ ∧ 1 / (real k)^2 ≤ 1 / (k / eps k / (1 −
eps k) + 1))
```

establishing the size requirements for 5.3. The one involving μ, namely $1\ /\ (real\ k)^2 \le \mu$, will be useful later with "big beta".

```
lemma Big-Red-5-3:
  assumes 0<μ0 μ1<1
  shows ∀∞l. ∀μ. μ ∈ {μ0..μ1} ⟶ Big-Red-5-3 μ l
  using assms Big-Red-5-1
  apply (simp add: Big-Red-5-3-def eps-def eventually-conj-iff all-imp-conj-distrib)

  apply (intro conjI strip eventually-all-geI0 eventually-all-ge-at-top)
  apply (real-asymp|force)+
  done

context Book
begin

corollary Red-5-3:
  assumes i: i ∈ Step-class {dboost-step}
    and big: Big-Red-5-3 μ l
  shows pseq (Suc i) ≥ pseq i ∧ beta i ≥ 1 / (real k)^2
proof
  have k>1 and big51: Big-Red-5-1 μ l
    using l-le-k big by (auto simp: Big-Red-5-3-def)
  let ?h = hgt (pseq i)
  have ?h > 0
    by (simp add: hgt-gt0 kn0 pee-le1)
  then obtain α: alpha ?h ≥ 0 and ∗: alpha ?h ≥ ε / k
    using alpha-ge0 ‹k>1› alpha-ge by auto
  moreover have −5/4 = −1/4 − (1::real)
    by simp
```

```
  ultimately have α54: alpha ?h ≥ k powr (−5/4)
    unfolding eps-def by (metis powr-diff of-nat-0-le-iff powr-one)
  have β: beta i ≤ μ
    by (metis Step-class-insert Un-iff beta-le i)
  have (1 − ε) ∗ ((1 − beta i) / beta i) ∗ alpha ?h ≥ 0
    using beta-ge0[of i] eps-le1 α β μ01 ‹k>1›
    by (simp add: zero-le-mult-iff zero-le-divide-iff)
  then show pseq (Suc i) ≥ pseq i
    using Red-5-2 [OF i big51] by linarith
  have pseq (Suc i) − pseq i ≤ 1
    by (smt (verit) pee-ge0 pee-le1)
  with Red-5-2 [OF i big51]
  have (1 − ε) ∗ ((1 − beta i) / beta i) ∗ alpha ?h ≤ 1 and beta-gt0: beta i > 0
    by linarith+
  with ∗ have (1 − ε) ∗ ((1 − beta i) / beta i) ∗ ε / k ≤ 1
     by (smt (verit, best) mult.commute eps-ge0 mult-mono mult-nonneg-nonpos
of-nat-0-le-iff times-divide-eq-right zero-le-divide-iff)
  then have (1 − ε) ∗ ((1 − beta i) / beta i) ≤ k / ε
    using beta-ge0 [of i] eps-gt0 kn0
   by (auto simp: divide-simps mult-less-0-iff mult-of-nat-commute split: if-split-asm)
  then have (1 − beta i) / beta i ≤ k / ε / (1 − ε)
    by (smt (verit) eps-less1 mult.commute pos-le-divide-eq ‹1 < k›)
  then have 1 / beta i ≤ k / ε / (1 − ε) + 1
    using beta-gt0 by (simp add: diff-divide-distrib)
  then have 1 / (k / ε / (1 − ε) + 1) ≤ beta i
    using beta-gt0 eps-gt0 eps-less1 [OF ‹k>1›] kn0
    apply (simp add: divide-simps split: if-split-asm)
    by (smt (verit, ccfv-SIG) mult.commute mult-less-0-iff)
  moreover have 1 / k^2 ≤ 1 / (k / ε / (1 − ε) + 1)
   using Big-Red-5-3-def l-le-k big eps-def by (metis (no-types, lifting) of-nat-power)
  ultimately show beta i ≥ 1 / (real k)²
    by auto
qed

corollary beta-gt0:
  assumes i ∈ Step-class {dboost-step}
    and Big-Red-5-3 μ l
  shows beta i > 0
  by (meson Big-Red-5-3-def Book.Red-5-2 Book-axioms assms)

end

end
```

5 Bounding the Size of Y

```
theory Bounding-Y imports Red-Steps

begin
```

yet another telescope variant, with weaker promises but a different conclusion; as written it holds even if $n = 0$

lemma *prod-lessThan-telescope-mult*:
  fixes *f*::*nat* ⇒ *'a*::*field*
  assumes ⋀*i*. *i*<*n* ⟹ *f i* ≠ *0*
  shows (∏*i*<*n*. *f* (*Suc i*) / *f i*) ∗ *f 0* = *f n*
  using *assms*
by (*induction n*) (*auto simp*: *divide-simps*)

5.1 The following results together are Lemma 6.4

Compared with the paper, all the indices are greater by one!!

context *Book*
begin

lemma *Y-6-4-Red*:
  assumes *i* ∈ *Step-class* {*red-step*}
  shows *pseq* (*Suc i*) ≥ *pseq i* − *alpha* (*hgt* (*pseq i*))
  using *assms*
  by (*auto simp*: *step-kind-defs next-state-def reddish-def pseq-def*
    split: *if-split-asm prod.split*)

lemma *Y-6-4-DegreeReg*:
  assumes *i* ∈ *Step-class* {*dreg-step*}
  shows *pseq* (*Suc i*) ≥ *pseq i*
  using *assms red-density-X-degree-reg-ge* [*OF Xseq-Yseq-disjnt*, *of i*]
  by (*auto simp*: *step-kind-defs degree-reg-def pseq-def split*: *if-split-asm prod.split-asm*)

lemma *Y-6-4-Bblue*:
  assumes *i*: *i* ∈ *Step-class* {*bblue-step*}
  shows *pseq* (*Suc i*) ≥ *pseq* (*i*−*1*) − (*ε powr* (−*1*/*2*)) ∗ *alpha* (*hgt* (*pseq* (*i*−*1*)))
proof −
  define *X* **where** *X* ≡ *Xseq i*
  define *Y* **where** *Y* ≡ *Yseq i*
  obtain *A B S T*
    where *step*: *stepper i* = (*X*,*Y*,*A*,*B*)
      and *nonterm*: ¬ *termination-condition X Y*
      and *odd i*
      and *mb*: *many-bluish X*
      and *bluebook*: (*S*,*T*) = *choose-blue-book* (*X*,*Y*,*A*,*B*)
    using *i*
      by (*simp add*: *X-def Y-def step-kind-defs split*: *if-split-asm prod.split-asm*)
(*metis mk-edge.cases*)
  then have *X1-eq*: *Xseq* (*Suc i*) = *T*
    by (*force simp*: *Xseq-def next-state-def split*: *prod.split*)
  have *Y1-eq*: *Yseq* (*Suc i*) = *Y*
    using *i* **by** (*simp add*: *Y-def step-kind-defs next-state-def split*: *if-split-asm*
prod.split-asm prod.split)

```
  have disjnt X Y
    using Xseq-Yseq-disjnt X-def Y-def by blast
  obtain fin: finite X finite Y
    by (metis V-state-stepper finX finY step)
  have X ≠ {} Y ≠ {}
    using gen-density-def nonterm termination-condition-def by fastforce+
  define i' where i' = i−1
  then have Suci': Suc i' = i
    by (simp add: ‹odd i›)
  have i': i' ∈ Step-class {dreg-step}
    by (metis dreg-before-step Step-class-insert Suci' UnCI i)
  then have Xseq (Suc i') = X-degree-reg (Xseq i') (Yseq i')
            Yseq (Suc i') = Yseq i'
      and nonterm': ¬ termination-condition (Xseq i') (Yseq i')
    by (auto simp: degree-reg-def X-degree-reg-def step-kind-defs split: if-split-asm
prod.split-asm)
  then have Xeq: X = X-degree-reg (Xseq i') (Yseq i')
       and  Yeq: Y = Yseq i'
    using Suci' by (auto simp: X-def Y-def)
  define pm where pm ≡ (pseq i' − ε powr (−1/2) ∗ alpha (hgt (pseq i')))
  have T ⊆ X
    using bluebook by (simp add: choose-blue-book-subset fin)
  then have T-reds: ⋀x. x ∈ T ⟹ pm ∗ card Y ≤ card (Neighbours Red x ∩
Y)
    by (auto simp: Xeq Yeq pm-def X-degree-reg-def pseq-def red-dense-def)
  have good-blue-book X (S,T)
    by (meson bluebook choose-blue-book-works fin)
  then have Tne: False if card T = 0
    using μ01 ‹X ≠ {}› fin by (simp add: good-blue-book-def pos-prod-le that)
  have pm ∗ card T ∗ card Y = (∑x∈T. pm ∗ card Y)
    by simp
  also have ... ≤ (∑x∈T. card (Neighbours Red x ∩ Y))
    using T-reds by (simp add: sum-bounded-below)
  also have ... = edge-card Red T Y
    using ‹disjnt X Y› ‹finite X› ‹T⊆X› Red-E
   by (metis disjnt-subset1 disjnt-sym edge-card-commute edge-card-eq-sum-Neighbours
finite-subset)
  also have ... = red-density T Y ∗ card T ∗ card Y
    using fin ‹T⊆X› by (simp add: finite-subset gen-density-def)
  finally have pm ≤ red-density T Y
   using fin ‹Y≠{}› Yeq Yseq-gt0 Tne nonterm' step-terminating-iff by fastforce
  then show ?thesis
    by (simp add: X1-eq Y1-eq i'-def pseq-def pm-def)
qed
```

The basic form is actually *Red-5-3*. This variant covers a gap of two, thanks to degree regularisation

```
corollary Y-6-4-dbooSt:
  assumes i: i ∈ Step-class {dboost-step} and big: Big-Red-5-3 μ l
```

```
  shows pseq (Suc i) ≥ pseq (i−1)
proof −
  have odd ii−1 ∈ Step-class {dreg-step}
   using step-odd i by (auto simp: Step-class-insert-NO-MATCH dreg-before-step)
  then show ?thesis
    using Red-5-3 Y-6-4-DegreeReg assms ‹odd i› by fastforce
qed
```

5.2 Towards Lemmas 6.3

```
definition Z-class ≡ {i ∈ Step-class {red-step,bblue-step,dboost-step}.
                        pseq (Suc i) < pseq (i−1) ∧ pseq (i−1) ≤ p0}

lemma finite-Z-class: finite (Z-class)
 using finite-components by (auto simp: Z-class-def Step-class-insert-NO-MATCH)


lemma Y-6-3:
  assumes big53: Big-Red-5-3 μ l and big41: Big-Blue-4-1 μ l
  shows (∑i ∈ Z-class. pseq (i−1) − pseq (Suc i)) ≤ 2 ∗ ε
proof −
  define S where S ≡ Step-class {dboost-step}
  define R where R ≡ Step-class {red-step}
  define B where B ≡ Step-class {bblue-step}
  { fix i
    assume i: i ∈ S
    moreover have odd i
      using step-odd [of i] i by (force simp: S-def Step-class-insert-NO-MATCH)
    ultimately have i−1 ∈ Step-class {dreg-step}
      by (simp add: S-def dreg-before-step Step-class-insert-NO-MATCH)
    then have pseq (i−1) ≤ pseq i ∧ pseq i ≤ pseq (Suc i)
      using big53 S-def
     by (metis Red-5-3 One-nat-def Y-6-4-DegreeReg ‹odd i› i odd-Suc-minus-one)
  }
  then have dboost: S ∩ Z-class = {}
    by (fastforce simp: Z-class-def)
  { fix i
    assume i: i ∈ B ∩ Z-class
    then have i−1 ∈ Step-class {dreg-step}
     using dreg-before-step step-odd i by (force simp: B-def Step-class-insert-NO-MATCH)
    have pseq: pseq (Suc i) < pseq (i−1) pseq (i−1) ≤ p0 and iB: i ∈ B
      using i by (auto simp: Z-class-def)
    have hgt (pseq (i−1)) = 1
    proof −
      have hgt (pseq (i−1)) ≤ 1
       by (smt (verit, del-insts) hgt-Least less-one pseq(2) qfun0 qfun-strict-mono)
      then show ?thesis
        by (metis One-nat-def Suc-pred' diff-is-0-eq hgt-gt0)
    qed
    then have pseq (i−1) − pseq (Suc i) ≤ ε powr (−1/2) ∗ alpha 1
```

```
      using pseq iB Y-6-4-Bblue μ01 by (fastforce simp: ℬ-def)
    also have ... ≤ 1/k
    proof −
      have k powr (−1/8) ≤ 1
        using kn0 by (simp add: ge-one-powr-ge-zero powr-minus-divide)
      then show ?thesis
        by (simp add: alpha-eq eps-def powr-powr divide-le-cancel flip: powr-add)
    qed
    finally have pseq (i−1) − pseq (Suc i) ≤ 1/k .
  }
  then have (∑ i ∈ ℬ ∩ Z-class. pseq (i−1) − pseq (Suc i))
            ≤ card (ℬ ∩ Z-class) ∗ (1/k)
    using sum-bounded-above by (metis (mono-tags, lifting))
  also have ... ≤ card (ℬ) ∗ (1/k)
    using bblue-step-finite
    by (simp add: ℬ-def divide-le-cancel card-mono)
  also have ... ≤ l powr (3/4) / k
    using big41 by (simp add: ℬ-def kn0 frac-le bblue-step-limit)
  also have ... ≤ ε
  proof −
    have ∗: l powr (3/4) ≤ k powr (3/4)
      by (simp add: l-le-k powr-mono2)
    have 3/4 − (1::real) = − 1/4
      by simp
    then show ?thesis
      using divide-right-mono [OF ∗, of k]
      by (metis eps-def of-nat-0-le-iff powr-diff powr-one)
  qed
  finally have bblue: (∑ i∈ℬ ∩ Z-class. pseq(i−1) − pseq (Suc i)) ≤ ε .
  { fix i
    assume i: i ∈ ℛ ∩ Z-class
    then have pee-alpha: pseq (i−1) − pseq (Suc i)
                    ≤ pseq (i−1) − pseq i + alpha (hgt (pseq i))
      using Y-6-4-Red by (force simp: ℛ-def)
    have pee-le: pseq (i−1) ≤ pseq i
      using dreg-before-step Y-6-4-DegreeReg[of i−1] i step-odd
      by (simp add: ℛ-def Step-class-insert-NO-MATCH)
    consider (1) hgt (pseq i) = 1 | (2) hgt (pseq i) > 1
      by (metis hgt-gt0 less-one nat-neq-iff)
    then have pseq (i−1) − pseq i + alpha (hgt (pseq i)) ≤ ε / k
    proof cases
      case 1
      then show ?thesis
        by (smt (verit) Red-5-7c kn0 pee-le hgt-works)
    next
      case 2
      then have p-gt-q: pseq i > qfun 1
        by (meson hgt-Least not-le zero-less-one)
      have pee-le-q0: pseq (i−1) ≤ qfun 0
```

```
        using 2 Z-class-def i by auto
      also have pee2: ... ≤ pseq i
        using alpha-eq p-gt-q by (smt (verit, best) kn0 qfun-mono zero-le-one)
      finally have pseq (i−1) ≤ pseq i .
      then have pseq (i−1) − pseq i + alpha (hgt (pseq i))
              ≤ qfun 0 − pseq i + ε ∗ (pseq i − qfun 0 + 1/k)
        using Red-5-7b pee-le-q0 pee2 by fastforce
      also have ... ≤ ε / k
       using kn0 pee2 by (simp add: algebra-simps) (smt (verit) affine-ineq eps-le1)
      finally show ?thesis .
    qed
    with pee-alpha have pseq (i−1) − pseq (Suc i) ≤ ε / k
      by linarith
  }
  then have (∑ i ∈ R ∩ Z-class. pseq (i−1) − pseq (Suc i))
          ≤ card (R ∩ Z-class) ∗ (ε / k)
    using sum-bounded-above by (metis (mono-tags, lifting))
  also have ... ≤ card (R) ∗ (ε / k)
    using eps-ge0 assms red-step-finite
    by (simp add: R-def divide-le-cancel mult-le-cancel-right card-mono)
  also have ... ≤ k ∗ (ε / k)
    using red-step-limit R-def μ01
     by (smt (verit, best) divide-nonneg-nonneg eps-ge0 mult-mono nat-less-real-le
of-nat-0-le-iff)
  also have ... ≤ ε
    using eps-ge0 by force
  finally have red: (∑ i∈R ∩ Z-class. pseq (i−1) − pseq (Suc i)) ≤ ε .
  have ∗: finite (B) finite (R) ⋀x. x ∈ B ⟹ x ∉ R
    using finite-components  by (auto simp: B-def R-def Step-class-def)
  have eq: Z-class = S ∩ Z-class  ∪ B ∩ Z-class ∪ R ∩ Z-class
    by (auto simp: Z-class-def B-def R-def S-def Step-class-insert-NO-MATCH)
  show ?thesis
    using bblue red
    by (subst eq) (simp add: sum.union-disjoint dboost disjoint-iff ∗)
qed
```

5.3 Lemma 6.5

```
lemma Y-6-5-Red:
  assumes i: i ∈ Step-class {red-step} and k≥16
  defines h ≡ λi. hgt (pseq i)
  shows h (Suc i) ≥ h i − 2
proof (cases h i ≤ 3)
  case True
  have h (Suc i) ≥ 1
    by (simp add: h-def Suc-leI hgt-gt0)
  with True show ?thesis
    by linarith
next
```

```
  case False
  have k>0 using assms by auto
  have ε ≤ 1/2
    using ‹k≥16› by (simp add: eps-eq-sqrt divide-simps real-le-rsqrt)
  moreover have 0 ≤ x ∧ x ≤ 1/2 ⟹ x * (1 + x)^2 + 1 ≤ (1 + x)^2 for x::real
    by sos
  ultimately have §: ε * (1 + ε)^2 + 1 ≤ (1 + ε)^2
    using eps-ge0 by presburger
  have le1: ε + 1 / (1 + ε)^2 ≤ 1
    using mult-left-mono [OF §, of inverse ((1 + ε)^2)]
    by (simp add: ring-distribs inverse-eq-divide) (smt (verit))
  have 0: 0 ≤ (1 + ε) ^ (h i − Suc 0)
    using eps-ge0 by auto
  have lesspi: qfun (h i − 1) < pseq i
    using False hgt-Least [of h i − 1 pseq i] unfolding h-def by linarith
  have A: (1 + ε) ^ h i = (1 + ε) * (1 + ε) ^ (h i − Suc 0)
    using False power.simps by (metis h-def Suc-pred hgt-gt0)
  have B: (1 + ε) ^ (h i − 3) = 1 / (1 + ε)^2 * (1 + ε) ^ (h i − Suc 0)
    using eps-gt0 False
    by (simp add: divide-simps Suc-diff-Suc numeral-3-eq-3 flip: power-add)
  have qfun (h i − 3) ≤ qfun (h i − 1) − (qfun (h i) − qfun (h i − 1))
    using kn0 mult-left-mono [OF le1 0]
   by (simp add: qfun-eq A B algebra-simps divide-right-mono flip: add-divide-distrib
diff-divide-distrib)
  also have ... < pseq i − alpha (h i)
    using lesspi by (simp add: alpha-def)
  also have ... ≤ pseq (Suc i)
    using Y-6-4-Red i by (force simp: h-def)
  finally have qfun (h i − 3) < pseq (Suc i) .
  with hgt-greater show ?thesis
    unfolding h-def by force
qed

lemma Y-6-5-DegreeReg:
  assumes i ∈ Step-class {dreg-step}
  shows hgt (pseq (Suc i)) ≥ hgt (pseq i)
  using hgt-mono Y-6-4-DegreeReg assms by presburger

corollary Y-6-5-dbooSt:
  assumes i ∈ Step-class {dboost-step} and Big-Red-5-3 μ l
  shows hgt (pseq (Suc i)) ≥ hgt (pseq i)
  using kn0 Red-5-3 assms hgt-mono by blast
```

this remark near the top of page 19 only holds in the limit

```
lemma ∀∞k. (1 + eps k) powr (− real (nat ⌊2 * eps k powr (−1/2)⌋)) ≤ 1 −
eps k powr (1/2)
  unfolding eps-def by real-asymp

end
```

definition *Big-Y-6-5-Bblue* ≡
$\lambda l.\ \forall k\geq l.\ (1 + eps\ k)\ powr\ (-\ real\ (nat\ \lfloor 2*(eps\ k\ powr\ (-1/2))\rfloor)) \leq 1 -$
$eps\ k\ powr\ (1/2)$

establishing the size requirements for Y 6.5

lemma *Big-Y-6-5-Bblue*:
shows $\forall^\infty l$. *Big-Y-6-5-Bblue l*
unfolding *Big-Y-6-5-Bblue-def eps-def* **by** (*intro eventually-all-ge-at-top*; *real-asymp*)

lemma (**in** *Book*) *Y-6-5-Bblue*:
fixes κ::*real*
defines $\kappa \equiv \varepsilon\ powr\ (-1/2)$
assumes *i*: $i \in$ *Step-class* {*bblue-step*} **and** *big*: *Big-Y-6-5-Bblue l*
defines $h \equiv hgt\ (pseq\ (i-1))$
shows $hgt\ (pseq\ (Suc\ i)) \geq h - 2*\kappa$
proof (*cases* $h > 2*\kappa + 1$)
case *True*
then have $0 < h - 1$
by (*smt* (*verit*, *best*) *κ-def one-less-of-natD powr-non-neg zero-less-diff*)
with *True* **have** $pseq\ (i-1) > qfun\ (h-1)$
by (*simp add*: *h-def hgt-less-imp-qfun-less*)
then have $qfun\ (h-1) - \varepsilon\ powr\ (1/2) * (1 + \varepsilon)\ \hat{}\ (h-1) / k < pseq\ (i-1) -$
$\kappa * alpha\ h$
using ‹$0 < h-1$› *Y-6-4-Bblue* [*OF i*] *eps-ge0*
apply (*simp add*: *alpha-eq κ-def*)
by (*smt* (*verit*, *best*) *field-sum-of-halves mult.assoc mult.commute powr-mult-base*)
also have ... $\leq pseq\ (Suc\ i)$
using *Y-6-4-Bblue i h-def κ-def* **by** *blast*
finally have *A*: $qfun\ (h-1) - \varepsilon\ powr\ (1/2) * (1 + \varepsilon)\ \hat{}\ (h-1) / k < pseq\ (Suc$
$i)$ **.**
have *ek0*: $0 < 1 + \varepsilon$
by (*smt* (*verit*, *best*) *eps-ge0*)
have *less-h*: $nat\ \lfloor 2*\kappa\rfloor < h$
using *True* ‹$0 < h - 1$› **by** *linarith*
have $qfun\ (h - nat\ \lfloor 2*\kappa\rfloor - 1) = p0 + ((1 + \varepsilon)\ \hat{}\ (h - nat\ \lfloor 2*\kappa\rfloor - 1) - 1)$
$/ k$
by (*simp add*: *qfun-eq*)
also have ... $\leq p0 + ((1 - \varepsilon\ powr\ (1/2)) * (1 + \varepsilon)\ \hat{}\ (h-1) - 1) / k$
proof −
have *ge0*: $(1 + \varepsilon)\ \hat{}\ (h-1) \geq 0$
using *eps-ge0* **by** *auto*
have $(1 + \varepsilon)\ \hat{}\ (h - nat\ \lfloor 2*\kappa\rfloor - 1) = (1 + \varepsilon)\ \hat{}\ (h-1) * (1 + \varepsilon)\ powr\ -$
$real(nat\ \lfloor 2*\kappa\rfloor)$
using *less-h ek0* **by** (*simp add*: *algebra-simps flip*: *powr-realpow powr-add*)
also have ... $\leq (1 - \varepsilon\ powr\ (1/2)) * (1 + \varepsilon)\ \hat{}\ (h-1)$
using *big l-le-k* **unfolding** *κ-def Big-Y-6-5-Bblue-def*
by (*metis mult.commute ge0 mult-left-mono*)
finally have $(1 + \varepsilon)\ \hat{}\ (h - nat\ \lfloor 2*\kappa\rfloor - 1)$

$\leq$ (*1* − *ε powr* (*1/2*)) ∗ (*1* + *ε*) ^ (*h*−*1*) **.**
then show *?thesis*
by (*intro add-left-mono divide-right-mono diff-right-mono*) *auto*
qed
also have ... $\leq$ *qfun* (*h*−*1*) − *ε powr* (*1/2*) ∗ (*1* + *ε*) ^ (*h*−*1*) / *real k*
using *kn0 eps-ge0* **by** (*simp add*: *qfun-eq powr-half-sqrt field-simps*)
also have ... < *pseq* (*Suc i*)
using *A* **by** *blast*
finally have *qfun* (*h* − *nat* ⌊*2*∗*κ*⌋ − *1*) < *pseq* (*Suc i*) **.**
then have *h* − *nat* ⌊*2*∗*κ*⌋ $\leq$ *hgt* (*pseq* (*Suc i*))
using *hgt-greater* **by** *force*
with *less-h* **show** *?thesis*
unfolding *κ-def*
by (*smt* (*verit*) *less-imp-le-nat of-nat-diff of-nat-floor of-nat-mono powr-ge-zero*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*, *del-insts*) *of-nat-0 hgt-gt0 nat-less-real-le*)
qed

5.4 Lemma 6.2

definition *Big-Y-6-2* ≡ λ*μ l*. *Big-Y-6-5-Bblue l* ∧ *Big-Red-5-3 μ l* ∧ *Big-Blue-4-1 μ l*
∧ (∀*k*≥*l*. ((*1* + *eps k*)^*2*) ∗ *eps k powr* (*1/2*) $\leq$ *1*
∧ (*1* + *eps k*) *powr* (*2* ∗ *eps k powr* (−*1/2*)) $\leq$ *2* ∧ *k* $\geq$ *16*)

establishing the size requirements for 6.2

lemma *Big-Y-6-2*:
assumes *0*<*μ0 μ1*<*1*
shows ∀$^\infty$*l*. ∀*μ*. *μ* ∈ {*μ0*..*μ1*} ⟶ *Big-Y-6-2 μ l*
using *assms Big-Y-6-5-Bblue Big-Red-5-3 Big-Blue-4-1*
unfolding *Big-Y-6-2-def eps-def*
apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)
apply (*intro conjI strip eventually-all-geI1 eventually-all-ge-at-top*; *real-asymp*)
done

context *Book*
begin

Following Bhavik in excluding the even steps (degree regularisation). Assuming it hasn't halted, the conclusion also holds for the even cases anyway.

proposition *Y-6-2*:
defines *RBS* ≡ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
assumes *j*: *j* ∈ *RBS* **and** *big*: *Big-Y-6-2 μ l*
shows *pseq* (*Suc j*) $\geq$ *p0* − *3* ∗ *ε*
proof (*cases pseq* (*Suc j*) $\geq$ *p0*)
case *True*
then show *?thesis*

by (*smt* (*verit*) *eps-ge0*)
next
case *False*
then have *pj-less*: *pseq*(*Suc j*) < *p0* **by** *linarith*
have *big53*: *Big-Red-5-3* *μ* *l*
and *Y63*: (∑*i* ∈ *Z-class*. *pseq* (*i*−*1*) − *pseq* (*Suc i*)) ≤ *2* ∗ *ε*
and *Y65B*: ⋀*i*. *i* ∈ *Step-class* {*bblue-step*} ⟹ *hgt* (*pseq* (*Suc i*)) ≥ *hgt* (*pseq* (*i*−*1*)) − *2*∗(*ε* *powr* (−*1*/*2*))
and *big1*: ((*1* + *ε*)ˆ*2*) ∗ *ε* *powr* (*1*/*2*) ≤ *1* **and** *big2*: (*1* + *ε*) *powr* (*2* ∗ *ε* *powr* (−*1*/*2*)) ≤ *2*
and *k*≥*16*
using *big Y-6-5-Bblue Y-6-3 kn0 l-le-k* **by** (*auto simp*: *Big-Y-6-2-def*)
have *Y64-S*: ⋀*i*. *i* ∈ *Step-class* {*dboost-step*} ⟹ *pseq* *i* ≤ *pseq* (*Suc i*)
using *big53 Red-5-3* **by** *simp*
define *J* **where** *J* ≡ {*j*′. *j*′<*j* ∧ *pseq* *j*′ ≥ *p0* ∧ *even j*′}
have *finite J*
by (*auto simp*: *J-def*)
have *pseq 0* = *p0*
by (*simp add*: *pee-eq-p0*)
have *odd-RBS*: *odd i* **if** *i* ∈ *RBS* **for** *i*
using *step-odd that* **unfolding** *RBS-def* **by** *blast*
with *odd-pos j* **have** *j*>*0* **by** *auto*
have *non-halted*: *j* ∉ *Step-class* {*halted*}
using *j* **by** (*auto simp*: *Step-class-def RBS-def*)
have *exists*: *J* ≠ {}
using ‹*0* < *j*› ‹*pseq 0* = *p0*› **by** (*force simp*: *J-def less-eq-real-def*)
define *j*′ **where** *j*′ ≡ *Max J*
have *j*′ ∈ *J*
using ‹*finite J*› *exists* **by** (*force simp*: *j*′*-def*)
then have *j*′ < *j even j*′ **and** *pSj*′: *pseq j*′ ≥ *p0*
by (*auto simp*: *J-def odd-RBS*)
have *maximal*: *j*′′ ≤ *j*′ **if** *j*′′ ∈ *J* **for** *j*′′
using ‹*finite J*› *exists* **by** (*simp add*: *j*′*-def that*)
have *pseq* (*j*′+*2*) − *2* ∗ *ε* ≤ *pseq* (*j*′+*2*) − (∑*i* ∈ *Z-class*. *pseq* (*i*−*1*) − *pseq* (*Suc i*))
using *Y63* **by** *simp*
also have . . . ≤ *pseq* (*Suc j*)
proof −
define *Z* **where** *Z* ≡ *λj*. {*i*. *pseq* (*Suc i*) < *pseq* (*i*−*1*) ∧ *j*′+*2* < *i* ∧ *i*≤*j* ∧ *i* ∈ *RBS*}
have *Zsub*: *Z i* ⊆ {*Suc j*′<..*i*} **for** *i*
by (*auto simp*: *Z-def*)
then have *finZ*: *finite* (*Z i*) **for** *i*
by (*meson finite-greaterThanAtMost finite-subset*)
have ∗: (∑*i* ∈ *Z j*. *pseq* (*i*−*1*) − *pseq* (*Suc i*)) ≤ (∑*i* ∈ *Z-class*. *pseq* (*i*−*1*) − *pseq* (*Suc i*))
proof (*intro sum-mono2* [*OF finite-Z-class*])
show *Z j* ⊆ *Z-class*
proof

```
      fix i
      assume i: i ∈ Z j
      then have dreg: i−1 ∈ Step-class {dreg-step} and i≠0 j' < i
        by (auto simp: Z-def RBS-def dreg-before-step)
      with i dreg maximal have pseq (i−1) < p0
        unfolding Z-def J-def
        using Suc-less-eq2 less-eq-Suc-le odd-RBS by fastforce
      then show i ∈ Z-class
        using i by (simp add: Z-def RBS-def Z-class-def)
    qed
    show 0 ≤ pseq (i−1) − pseq (Suc i) if i ∈ Z-class − Z j for i
      using that by (auto simp: Z-def Z-class-def)
  qed
  then have pseq (j'+2) − (∑i∈Z-class. pseq (i−1) − pseq (Suc i))
         ≤ pseq (j'+2) − (∑i ∈ Z j. pseq (i−1) − pseq (Suc i))
    by auto
  also have ... ≤ pseq (Suc j)
  proof −
    have pseq (j'+2) − pseq (Suc m) ≤ (∑i ∈ Z m. pseq (i−1) − pseq (Suc i))
      if m ∈ RBS j' < m m≤j for m
      using that
    proof (induction m rule: less-induct)
      case (less m)
      then have odd m
        using odd-RBS by blast
      show ?case
      proof (cases j'+2 < m)
        case True
        with less.prems
          have Z-if: Z m = (if pseq (Suc m) < pseq (m−1) then insert m (Z
(m−2)) else Z (m−2))
          by (auto simp: Z-def)
            (metis le-diff-conv2 Suc-leI add-2-eq-Suc' add-leE even-Suc nat-less-le
odd-RBS)+
        have m−2 ∈ RBS
          using True ‹m ∈ RBS› step-odd-minus2 by (auto simp: RBS-def)
        then have *: pseq (j'+2) − pseq (m − Suc 0) ≤ (∑i∈Z (m − 2). pseq
(i−1) − pseq (Suc i))
          using less.IH True less ‹j' ∈ J› by (force simp: J-def Suc-less-eq2)
        moreover have m ∉ Z (m − 2)
          by (auto simp: Z-def)
        ultimately show ?thesis
          by (simp add: Z-if finZ)
      next
        case False
        then have [simp]: m = Suc j'
          using ‹odd m› ‹j' < m› ‹even j'› by presburger
        have Z m = {}
          by (auto simp: Z-def)
```

```
        then show ?thesis
          by simp
      qed
    qed
    then show ?thesis
      using j J-def ‹j' ∈ J› ‹j' < j› by force
  qed
  finally show ?thesis .
qed
finally have p2-le-pSuc: pseq (j'+2) − 2 ∗ ε ≤ pseq (Suc j) .
have Suc j' ∈ RBS
  unfolding RBS-def
proof (intro not-halted-odd-RBS)
  show Suc j' ∉ Step-class {halted}
    using Step-class-halted-forever Suc-leI ‹j' < j› non-halted by blast
qed (use ‹even j'› in auto)
then have pseq (j'+2) < p0
  using maximal[of j'+2] False ‹j' < j› j odd-RBS
  by (simp add: J-def) (smt (verit, best) Suc-lessI even-Suc)
then have le1: hgt (pseq (j'+2)) ≤ 1
  by (smt (verit) kn0 hgt-Least qfun0 qfun-strict-mono zero-less-one)
moreover
have j'-dreg: j' ∈ Step-class {dreg-step}
  using RBS-def ‹Suc j' ∈ RBS› dreg-before-step by blast
have 1: ε powr −(1/2) ≥ 1
  using kn0 by (simp add: eps-def powr-powr ge-one-powr-ge-zero)
consider (R) Suc j' ∈ Step-class {red-step}
       | (B) Suc j' ∈ Step-class {bblue-step}
       | (S) Suc j' ∈ Step-class {dboost-step}
  by (metis Step-class-insert UnE ‹Suc j' ∈ RBS› RBS-def)
note j'-cases = this
then have hgt-le-hgt: hgt (pseq j') ≤ hgt (pseq (j'+2)) + 2 ∗ ε powr (−1/2)
proof cases
  case R
  have real (hgt (pseq j')) ≤ hgt (pseq (Suc j'))
    using Y-6-5-DegreeReg[OF j'-dreg] kn0 by (simp add: eval-nat-numeral)
  also have ... ≤ hgt (pseq (j'+2)) + 2 ∗ ε powr (−1/2)
    using Y-6-5-Red[OF R ‹k≥16›] 1 by (simp add: eval-nat-numeral)
  finally show ?thesis .
next
  case B
  show ?thesis
    using Y65B [OF B] by simp
next
  case S
  then show ?thesis
    using Y-6-4-DegreeReg ‹pseq (j'+2) < p0› Y64-S j'-dreg pSj' by force
qed
ultimately have B: hgt (pseq j') ≤ 1 + 2 ∗ ε powr (−1/2)
```

by *linarith*
have $2 \le$ *real k powr* $(1/2)$
using ‹$k \ge 16$› **by** (*simp add*: *powr-half-sqrt real-le-rsqrt*)
then have *8*: $2 \le$ *real k powr 1* $*$ *real k powr* $-(1/8)$
unfolding *powr-add* [*symmetric*] **using** ‹$k \ge 16$› *order.trans nle-le* **by** *fastforce*
have $p0 - \varepsilon \le$ *qfun 0* $- 2 * \varepsilon$ *powr* $(1/2) / k$
using *mult-left-mono* [*OF 8*, *of k powr* $(-1/8)$] *kn0*
by (*simp add*: *qfun-eq eps-def powr-powr field-simps flip*: *powr-add*)
also have ... $\le$ *pseq j'* $- \varepsilon$ *powr* $(-1/2) *$ *alpha* (*hgt* (*pseq j'*))
proof −
have *2*: $(1 + \varepsilon)$ ^ (*hgt* (*pseq j'*) − *Suc 0*) ≤ 2
using *B big2 kn0 eps-ge0*
by (*smt* (*verit*) *diff-Suc-less hgt-gt0 nat-less-real-le powr-mono powr-realpow*)
have $*$: $x \ge 0 \Longrightarrow$ *inverse* (*x powr* $(1/2)$) $* x = x$ *powr* $(1/2)$ **for** *x*::*real*
by (*simp add*: *inverse-eq-divide powr-half-sqrt real-div-sqrt*)
have *p0* − *pseq j'* ≤ 0
by (*simp add*: *pSj'*)
also have ... $\le 2 * \varepsilon$ *powr* $(1/2) / k - (\varepsilon$ *powr* $(1/2)) * (1 + \varepsilon)$ ^ (*hgt* (*pseq j'*) − *1*) / *k*
using *mult-left-mono* [*OF 2*, *of* ε *powr* $(1/2) / k$]
by (*simp add*: *field-simps diff-divide-distrib*)
finally have $p0 - 2 * \varepsilon$ *powr* $(1/2) / k$
$\le$ *pseq j'* $- (\varepsilon$ *powr* $(1/2)) * (1 + \varepsilon)$ ^ (*hgt* (*pseq j'*) − *1*) / *k*
by *simp*
with $*$ [*OF eps-ge0*] **show** *?thesis*
by (*simp add*: *alpha-hgt-eq powr-minus*) (*metis mult.assoc*)
qed
also have ... $\le$ *pseq* (*j'*+*2*)
using *j'-cases*
proof *cases*
case *R*
have *hs-le3*: *hgt* (*pseq* (*Suc j'*)) ≤ 3
using *le1 Y-6-5-Red*[*OF R* ‹$k \ge 16$›] **by** *simp*
then have *h-le3*: *hgt* (*pseq j'*) ≤ 3
using *Y-6-5-DegreeReg* [*OF j'-dreg*] **by** *simp*
have *alpha1*: *alpha* (*hgt* (*pseq* (*Suc j'*))) $\le \varepsilon * (1 + \varepsilon)$ ^ $2 / k$
by (*metis alpha-Suc-eq alpha-mono hgt-gt0 hs-le3 numeral-nat*(*3*))
have *alpha2*: *alpha* (*hgt* (*pseq j'*)) $\ge \varepsilon / k$
by (*simp add*: *Red-5-7a*)
have *pseq j'* $- \varepsilon$ *powr* $(-1/2) *$ *alpha* (*hgt* (*pseq j'*))
$\le$ *pseq* (*Suc j'*) − *alpha* (*hgt* (*pseq* (*Suc j'*)))
proof −
have *alpha* (*hgt* (*pseq* (*Suc j'*))) $\le (1 + \varepsilon)^2 *$ *alpha* (*hgt* (*pseq j'*))
using *alpha1 mult-left-mono* [*OF alpha2*, *of* $(1 + \varepsilon)^2$]
by (*simp add*: *mult.commute*)
also have ... $\le$ *inverse* (ε *powr* $(1/2)$) $*$ *alpha* (*hgt* (*pseq j'*))
using *mult-left-mono* [*OF big1*, *of alpha* (*hgt* (*pseq j'*))] *eps-gt0 alpha-ge0*
by (*simp add*: *divide-simps mult-ac*)
finally have *alpha* (*hgt* (*pseq* (*Suc j'*)))

```
          ≤ inverse (ε powr (1/2)) * alpha (hgt (pseq j')) .
      then show ?thesis
        using Y-6-4-DegreeReg[OF j'-dreg] by (simp add: powr-minus)
    qed
    also have ... ≤ pseq (j'+2)
      by (simp add: R Y-6-4-Red)
    finally show ?thesis .
  next
    case B
    then show ?thesis
      using Y-6-4-Bblue by force
  next
    case S
    show ?thesis
      using Y-6-4-DegreeReg S ‹pseq (j'+2) < p0› Y64-S j'-dreg pSj' by fastforce
  qed
  finally have p0 − ε ≤ pseq (j'+2) .
  then have p0 − 3 * ε ≤ pseq (j'+2) − 2 * ε
    by simp
  with p2-le-pSuc show ?thesis
    by linarith
qed

corollary Y-6-2-halted:
  assumes big: Big-Y-6-2 μ l
  shows pseq halted-point ≥ p0 − 3 * ε
proof (cases halted-point=0)
  case True
  then show ?thesis
    by (simp add: eps-ge0 pee-eq-p0)
next
  case False
  then have halted-point−1 ∉ Step-class {halted}
    by (simp add: halted-point-minimal)
  then consider halted-point−1 ∈ Step-class {red-step,bblue-step,dboost-step}
          | halted-point−1 ∈ Step-class {dreg-step}
    using not-halted-even-dreg not-halted-odd-RBS by blast
  then show ?thesis
  proof cases
    case 1
    with False Y-6-2[of halted-point−1] big show ?thesis by simp
  next
    case m1-dreg: 2
    then have *: pseq halted-point ≥ pseq (halted-point−1)
      using False Y-6-4-DegreeReg[of halted-point−1] by simp
    have odd halted-point
      using m1-dreg False step-even[of halted-point−1] by simp
    then consider halted-point=1 | halted-point≥2
      by (metis False less-2-cases One-nat-def not-le)
```

then show *?thesis*
proof *cases*
case *1*
with ∗ *eps-gt0 kn0* **show** *?thesis*
by (*simp add*: *pee-eq-p0*)
next
case *2*
then have *m2*: *halted-point*−*2* ∈ *Step-class* {*red-step*,*bblue-step*,*dboost-step*}
using *step-before-dreg*[*of halted-point*−*2*] *m1-dreg*
by (*simp flip*: *Suc-diff-le*)
then obtain *j* **where** *j*: *halted-point*−*1* = *Suc j*
using *2 not0-implies-Suc* **by** *fastforce*
then have *pseq* (*Suc j*) ≥ *p0* − *3* ∗ ε
by (*metis m2 Suc-1 Y-6-2 big diff-Suc-1 diff-Suc-eq-diff-pred*)
with ∗ *j* **show** *?thesis* **by** *simp*
qed
qed
qed

end

5.5 Lemma 6.1

context *P0-min*
begin

definition *ok-fun-61* ≡ λ*k*. (*2* ∗ *real k*) ∗ *log 2* (*1* − *2* ∗ *eps k powr* (*1*/*2*) / *p0-min*)

lemma *ok-fun-61-works*:
assumes *p0-min* > *2* ∗ *eps k powr* (*1*/*2*)
shows *2 powr* (*ok-fun-61 k*) = (*1* − *2* ∗ (*eps k*) *powr* (*1*/*2*) / *p0-min*) ^ (*2*∗*k*)
using *p0-min assms*
by (*simp add*: *powr-def ok-fun-61-def log-def flip*: *powr-realpow*)

lemma *ok-fun-61*: *ok-fun-61* ∈ *o*(*real*)
unfolding *eps-def ok-fun-61-def*
using *p0-min* **by** *real-asymp*

definition
Big-Y-6-1 ≡
λμ *l*. *Big-Y-6-2* μ *l* ∧ (∀ *k*≥*l*. *eps k powr* (*1*/*2*) ≤ *1*/*3* ∧ *p0-min* > *2* ∗ *eps k powr* (*1*/*2*))

establishing the size requirements for 6.1

lemma *Big-Y-6-1*:
assumes *0*<μ*0* μ*1*<*1*
shows ∀∞*l*. ∀μ. μ ∈ {μ*0*..μ*1*} ⟶ *Big-Y-6-1* μ *l*
using *p0-min assms Big-Y-6-2*
unfolding *Big-Y-6-1-def eps-def*

```
  apply (simp add: eventually-conj-iff all-imp-conj-distrib)
  apply (intro conjI strip eventually-all-ge-at-top eventually-all-geI0; real-asymp)
  done

end

lemma (in Book) Y-6-1:
  assumes big: Big-Y-6-1 μ l
  defines st ≡ Step-class {red-step,dboost-step}
  shows card (Yseq halted-point) / card Y0 ≥ 2 powr (ok-fun-61 k) * p0 ^ card st
proof -
  have big13: ε powr (1/2) ≤ 1/3
    and big-p0: p0-min > 2 * ε powr (1/2)
    and big62: Big-Y-6-2 μ l
    and big41: Big-Blue-4-1 μ l
    using big l-le-k by (auto simp: Big-Y-6-1-def Big-Y-6-2-def)
  with l-le-k have dboost-step-limit: card (Step-class {dboost-step}) < k
    using bblue-dboost-step-limit by fastforce
  define p0m where p0m ≡ p0 - 2 * ε powr (1/2)
  have p0m > 0
    using big-p0 p0-ge by (simp add: p0m-def)
  let ?RS = Step-class {red-step,dboost-step}
  let ?BD =  Step-class {bblue-step,dreg-step}
  have not-halted-below-m: i ∉ Step-class {halted} if i < halted-point for i
    using that by (simp add:  halted-point-minimal)
  have BD-card: card (Yseq i) = card (Yseq (Suc i))
    if i ∈ ?BD for i
  proof -
    have Yseq (Suc i) = Yseq i
      using that
        by (auto simp: step-kind-defs next-state-def degree-reg-def split: prod.split
if-split-asm)
    with p0-01 kn0 show ?thesis
      by auto
  qed
  have RS-card: p0m * card (Yseq i) ≤ card (Yseq (Suc i))
    if i ∈ ?RS for i
  proof -
    have Yeq: Yseq (Suc i) = Neighbours Red (cvx i) ∩ Yseq i
      using that
      by (auto simp: step-kind-defs next-state-def split: prod.split if-split-asm)
    have odd i
      using that step-odd by (auto simp: Step-class-def)
    moreover have i-not-halted: i ∉ Step-class {halted}
      using that by (auto simp: Step-class-def)
    ultimately have iminus1-dreg: i - 1 ∈ Step-class {dreg-step}
      by (simp add: dreg-before-step not-halted-odd-RBS)
    have p0m * card (Yseq i) ≤ (1 - ε powr (1/2)) * pseq (i-1) * card (Yseq i)
    proof (cases i=1)
```

```
      case True
      with p0-01 show ?thesis
        by (simp add: p0m-def pee-eq-p0 algebra-simps mult-right-mono)
    next
      case False
      with ‹odd i› have i>2
        by (metis Suc-lessI dvd-refl One-nat-def odd-pos one-add-one plus-1-eq-Suc)
      have i−2 ∈ Step-class {red-step,bblue-step,dboost-step}
      proof (intro not-halted-odd-RBS)
        show i − 2 ∉ Step-class {halted}
          using i-not-halted Step-class-not-halted diff-le-self by blast
        show odd (i−2)
          using ‹2 < i› ‹odd i› by auto
      qed
      then have Y62: pseq (i−1) ≥ p0 − 3 ∗ ε
        using Y-6-2 [OF - big62] ‹2 < i› by (metis Suc-1 Suc-diff-Suc Suc-lessD)
      show ?thesis
      proof (intro mult-right-mono)
        have ε powr (1/2) ∗ pseq (i−1) ≤ ε powr (1/2) ∗ 1
          by (metis mult.commute mult-right-mono powr-ge-zero pee-le1)
        moreover have 3 ∗ ε ≤ ε powr (1/2)
        proof −
          have 3 ∗ ε = 3 ∗ (ε powr (1/2))^2
            using eps-ge0 powr-half-sqrt real-sqrt-pow2 by presburger
          also have . . . ≤ 3 ∗ ((1/3) ∗ ε powr (1/2))
            by (smt (verit) big13 mult-right-mono power2-eq-square powr-ge-zero)
          also have . . . ≤ ε powr (1/2)
            by simp
          finally show ?thesis .
        qed
        ultimately show p0m ≤ (1 − ε powr (1/2)) ∗ pseq (i − 1)
          using Y62 by (simp add: p0m-def algebra-simps)
      qed auto
    qed
    also have . . . ≤ card (Neighbours Red (cvx i) ∩ Yseq i)
      using Red-5-8 [OF iminus1-dreg] cvx-in-Xseq that ‹odd i›
        by fastforce
    finally show ?thesis
      by (simp add: Yeq)
  qed
  define ST where ST ≡ λi. ?RS ∩ {..<i}
  have ST (Suc i) = (if i ∈ ?RS then insert i (ST i) else ST i) for i
    by (auto simp: ST-def less-Suc-eq)
  then have [simp]: card (ST (Suc i)) = (if i ∈ ?RS then Suc (card (ST i)) else
card (ST i)) for i
    by (simp add: ST-def)
  have STm: ST halted-point = st
    by (auto simp: ST-def st-def Step-class-def simp flip: halted-point-minimal)
  have p0m ^ card (ST i) ≤ (∏j<i. card (Yseq(Suc j)) / card (Yseq j)) if
```

```
i≤halted-point for i
    using that
  proof (induction i)
    case 0
    then show ?case
      by (auto simp: ST-def)
  next
    case (Suc i)
    then have i: i ∉ Step-class {halted}
      by (simp add: not-halted-below-m)
    consider (RS) i ∈ ?RS
          | (BD) i ∈ ?BD ∧ i ∉ ?RS
      using i stepkind.exhaust by (auto simp: Step-class-def)
    then show ?case
    proof cases
      case RS
      then have p0m ^ card (ST (Suc i)) = p0m * p0m ^ card (ST i)
        by simp
      also have ... ≤ p0m * (∏j<i. card (Yseq(Suc j)) / card (Yseq j))
        using Suc Suc-leD ‹0 < p0m› mult-left-mono by auto
      also have ... ≤ (card (Yseq (Suc i)) / card (Yseq i)) * (∏j<i. card (Yseq
(Suc j)) / card (Yseq j))
      proof (intro mult-right-mono)
        show p0m ≤ card (Yseq (Suc i)) / card (Yseq i)
          by (simp add: RS RS-card Yseq-gt0 i pos-le-divide-eq)
      qed (simp add: prod-nonneg)
      also have ... = (∏j<Suc i.  card (Yseq (Suc j)) / card (Yseq j))
        by simp
      finally show ?thesis .
    next
      case BD
      with Yseq-gt0 [OF i] show ?thesis
        by (simp add: Suc Suc-leD BD-card)
    qed
  qed
  then have p0m ^ card (ST halted-point) ≤ (∏j < halted-point. card (Yseq(Suc
j)) / card (Yseq j))
    by blast
  also have ... = card (Yseq halted-point) / card (Yseq 0)
  proof −
    have ⋀i. i < halted-point ⟹ card (Yseq i) ≠ 0
      by (metis Yseq-gt0 less-irrefl not-halted-below-m)
    then show ?thesis
      using card-XY0 prod-lessThan-telescope-mult [of halted-point λi. real (card
(Yseq i))]
      by (simp add: nonzero-eq-divide-eq)
  qed
  finally have *: (p0 − 2 * ε powr (1/2)) ^ card st ≤ card (Yseq halted-point)
/ card (Y0)
```

```
    by (simp add: STm p0m-def)
  — Asymptotic part of the argument
  have st-le-2k: card st ≤ 2 * k
  proof -
    have st ⊆ Step-class {red-step,dboost-step}
      by (auto simp: st-def Step-class-insert-NO-MATCH)
    moreover have finite (Step-class {red-step,dboost-step})
      using finite-components by (auto simp: Step-class-insert-NO-MATCH)
    ultimately have card st ≤ card (Step-class {red-step,dboost-step})
      using card-mono by blast
    also have ... = card (Step-class {red-step} ∪ Step-class {dboost-step})
      by (auto simp: Step-class-insert-NO-MATCH)
    also have ... ≤ k+k
       by (meson add-le-mono card-Un-le dboost-step-limit le-trans less-imp-le-nat
red-step-limit)
    finally show ?thesis
      by auto
  qed
  have 2 powr (ok-fun-61 k) * p0 ^ card st ≤ (p0 − 2 * ε powr (1/2)) ^ card st
  proof -
    have 2 powr (ok-fun-61 k) = (1 − 2 * ε powr(1/2) / p0-min) ^ (2*k)
      using big-p0 ok-fun-61-works by blast
    also have ... ≤ (1 − 2 * ε powr(1/2) / p0) ^ (2*k)
      using p0-ge p0-min big-p0 by (intro power-mono) (auto simp: frac-le)
    also have ... ≤ (1 − 2 * ε powr(1/2) / p0) ^ card st
      using big-p0 p0-01 ‹0 < p0m›
      by (intro power-decreasing st-le-2k) (auto simp: p0m-def)
    finally have §: 2 powr ok-fun-61 k ≤ (1 − 2 * ε powr (1/2) / p0) ^ card st .
    have (1 − 2 * ε powr (1/2) / p0) ^ card st * p0 ^ card st
      = ((1 − 2 * ε powr (1/2) / p0) * p0) ^ card st
      by (simp add: power-mult-distrib)
    also have ... = (p0 − 2 * ε powr (1/2)) ^ card st
      using p0-01 by (simp add: algebra-simps)
    finally show ?thesis
      using mult-right-mono [OF §, of p0 ^ card st] p0-01 by auto
  qed
  with * show ?thesis
    by linarith
qed

end
```

6 Bounding the Size of X

```
theory Bounding-X imports Bounding-Y

begin
```

6.1 Preliminaries

lemma *sum-odds-even*:
  fixes f :: *nat* ⇒ *'a* :: *ab-group-add*
  assumes *even m*
  shows $(\sum i \in \{i.\ i<m \wedge odd\ i\}.\ f\ (Suc\ i) - f\ (i - Suc\ 0)) = f\ m - f\ 0$
  using *assms*
proof (*induction m rule*: *less-induct*)
  case (*less m*)
  show *?case*
  proof (*cases m<2*)
    case *True*
    with ‹*even m*› **show** *?thesis*
      by *fastforce*
  next
    case *False*
    have *eq*: $\{i.\ i<m \wedge odd\ i\} = insert\ (m-1)\ \{i.\ i<m-2 \wedge odd\ i\}$
    proof
      show $\{i.\ i < m \wedge odd\ i\} \subseteq insert\ (m - 1)\ \{i.\ i < m - 2 \wedge odd\ i\}$
        using ‹*even m*› **by** *clarify presburger*
    qed (*use False less* **in** *auto*)
    have [*simp*]: $\neg\ (m - Suc\ 0 < m - 2)$
      by *linarith*
    show *?thesis*
      using *False* **by** (*simp add*: *eq less flip*: *numeral-2-eq-2*)
  qed
qed

lemma *sum-odds-odd*:
  fixes f :: *nat* ⇒ *'a* :: *ab-group-add*
  assumes *odd m*
  shows $(\sum i \in \{i.\ i<m \wedge odd\ i\}.\ f\ (Suc\ i) - f\ (i - Suc\ 0)) = f\ (m-1) - f\ 0$
proof −
  have *eq*: $\{i.\ i<m \wedge odd\ i\} = \{i.\ i<m-1 \wedge odd\ i\}$
    using *assms not-less-iff-gr-or-eq* **by** *fastforce*
  show *?thesis*
    by (*simp add*: *sum-odds-even eq assms*)
qed

context *Book*
begin

the set of moderate density-boost steps (page 20)

definition *dboost-star* **where**
  dboost-star ≡ $\{i \in Step\text{-}class\ \{dboost\text{-}step\}.\ real\ (hgt\ (pseq\ (Suc\ i))) - hgt\ (pseq\ i) \le \varepsilon\ powr\ (-1/4)\}$

definition *bigbeta* **where**
  bigbeta ≡ *let* S = *dboost-star in if* $S = \{\}$ *then* μ *else* (*card* S) ∗ *inverse* ($\sum i{\in}S.$

inverse (*beta i*))

lemma *dboost-star-subset*: *dboost-star* ⊆ *Step-class* {*dboost-step*}
  by (*auto simp*: *dboost-star-def*)

lemma *finite-dboost-star*: *finite* (*dboost-star*)
    by (*meson dboost-step-finite dboost-star-subset finite-subset*)

lemma *bigbeta-ge0*: *bigbeta* ≥ *0*
  using *μ01* **by** (*simp add*: *bigbeta-def Let-def beta-ge0 sum-nonneg*)

lemma *bigbeta-ge-square*:
  assumes *big*: *Big-Red-5-3 μ l*
  shows *bigbeta* ≥ *1* / (*real k*)^*2*
proof −
  have *k*: *1* / (*real k*)^2 ≤ *μ*
    using *big kn0 l-le-k* **by** (*auto simp*: *Big-Red-5-3-def*)
  have *fin*: *finite* (*dboost-star*)
    using *assms finite-dboost-star* **by** *blast*
  have *R53*: ∀ *i* ∈ *Step-class* {*dboost-step*}. *1* / (*real k*)^*2* ≤ *beta i*
    using *Red-5-3 assms* **by** *blast*
  show *1* / (*real k*)^*2* ≤ *bigbeta*
  proof (*cases dboost-star* = {})
    case *True*
    then show *?thesis*
      using *assms k* **by** (*simp add*: *bigbeta-def*)
  next
    case *False*
    then have *card-gt0*: *card* (*dboost-star*) > *0*
      by (*meson card-gt-0-iff dboost-star-subset fin finite-subset*)
    moreover have ∗: ∀ *i* ∈ *dboost-star*. *beta i* > *0* ∧ (*real k*)^*2* ≥ *inverse* (*beta*
i)
      using *R53 kn0 assms* **by** (*simp add*: *beta-gt0 field-simps dboost-star-def*)
    ultimately have (∑*i*∈*dboost-star*. *inverse* (*beta i*)) ≤ *card* (*dboost-star*) ∗
(*real k*)^*2*
      by (*simp add*: *sum-bounded-above*)
    moreover have (∑*i*∈*dboost-star*. *inverse* (*beta i*)) ≠ *0*
      by (*metis* ∗ *False fin inverse-positive-iff-positive less-irrefl sum-pos*)
    ultimately show *?thesis*
      using *False card-gt0 k bigbeta-ge0*
      by (*simp add*: *bigbeta-def Let-def divide-simps split*: *if-split-asm*)
  qed
qed

lemma *bigbeta-gt0*:
  assumes *big*: *Big-Red-5-3 μ l*
  shows *bigbeta* > *0*
  by (*smt* (*verit*) *kn0 assms bigbeta-ge-square of-nat-zero-less-power-iff zero-less-divide-iff*)

```
lemma bigbeta-less1:
  assumes big: Big-Red-5-3 μ l
  shows bigbeta < 1
proof –
  have *: ∀ i∈Step-class {dboost-step}. 0 < beta i
    using assms beta-gt0 big by blast
  have fin: finite (Step-class {dboost-step})
    using dboost-step-finite  assms by blast
  show bigbeta < 1
  proof (cases dboost-star = {})
    case True
    then show ?thesis
      using assms μ01 by (simp add: bigbeta-def)
  next
    case False
    then have gt0: card (dboost-star) > 0
      by (meson card-gt-0-iff dboost-star-subset fin finite-subset)
    have real (card (dboost-star)) = (∑ i∈dboost-star. 1)
      by simp
    also have ...  < (∑ i∈dboost-star. 1 / beta i)
    proof (intro sum-strict-mono)
      show finite (dboost-star)
        using card-gt-0-iff gt0 by blast
      fix i
      assume i ∈ dboost-star
      with assms μ01 * dboost-star-subset beta-le
      show 1 < 1 / beta i
        by (force simp: Step-class-insert-NO-MATCH)
    qed (use False in auto)
    finally show ?thesis
      using False by (simp add: bigbeta-def Let-def divide-simps)
  qed
qed

lemma bigbeta-le:
  assumes big: Big-Red-5-3 μ l
  shows bigbeta ≤ μ
proof –
  have real (card (dboost-star)) = (∑ i∈dboost-star. 1)
    by simp
  also have ... ≤ (∑ i∈dboost-star. μ / beta i)
  proof (intro sum-mono)
    fix i
    assume i: i ∈ dboost-star
    with beta-le dboost-star-subset have beta i ≤ μ
      by (auto simp: Step-class-insert-NO-MATCH)
    with beta-gt0 assms show 1 ≤ μ / beta i
      by (smt (verit) dboost-star-subset divide-less-eq-1-pos i subset-iff)
```

qed
also have ... = $\mu * (\sum i{\in}dboost\text{-}star.\ 1\ /\ beta\ i)$
by (*simp add*: *sum-distrib-left*)
finally have *real* (*card* (*dboost-star*)) $\le \mu * (\sum i{\in}dboost\text{-}star.\ 1\ /\ beta\ i)$ **.**
moreover have $(\sum i{\in}dboost\text{-}star.\ 1\ /\ beta\ i) \ge 0$
by (*simp add*: *beta-ge0 sum-nonneg*)
ultimately show *?thesis*
using *μ01* **by** (*simp add*: *bigbeta-def Let-def divide-simps*)
qed

end

6.2 Lemma 7.2

definition *Big-X-7-2* ≡ λμ *l*. *nat* ⌈*real l powr* (*3/4*)⌉ ≥ *3* ∧ *l* > *1* / (*1*−μ)

establishing the size requirements for 7.11

lemma *Big-X-7-2*:
assumes *0*<μ*0* μ*1*<*1*
shows ∀$^{\infty}$*l*. ∀μ. μ ∈ {μ*0*..μ*1*} ⟶ *Big-X-7-2* μ *l*
unfolding *Big-X-7-2-def eventually-conj-iff all-imp-conj-distrib eps-def*
apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)
apply (*intro conjI strip eventually-all-geI1* [**where** *L*=*1*] *eventually-all-ge-at-top*)
apply *real-asymp*+
by (*smt* (*verit*, *best*) ‹μ*1*<*1*› *frac-le*)

definition *ok-fun-72* ≡ λμ *k*. (*real k* / *ln 2*) ∗ *ln* (*1* − *1* / (*k* ∗ (*1*−μ)))

lemma *ok-fun-72*:
assumes μ<*1*
shows *ok-fun-72* μ ∈ *o*(*real*)
using *assms* **unfolding** *ok-fun-72-def* **by** *real-asymp*

lemma *ok-fun-72-uniform*:
assumes *0*<μ*0* μ*1*<*1*
assumes *e*>*0*
shows ∀$^{\infty}$*k*. ∀μ. μ*0* ≤ μ ∧ μ ≤ μ*1* ⟶ |*ok-fun-72* μ *k*| / *k* ≤ *e*
proof (*intro eventually-all-geI1* [**where** *L* = *Suc*(*nat*⌈*1*/(*1*−μ*1*)⌉)])
show ∀$^{\infty}$*k*. |*ok-fun-72* μ*1* *k*| / *real k* ≤ *e*
using *assms* **unfolding** *ok-fun-72-def* **by** *real-asymp*
next
fix *k* μ
assume *le-e*: |*ok-fun-72* μ*1* *k*| / *real k* ≤ *e*
and μ: μ*0* ≤ μ μ ≤ μ*1*
and *k*: *Suc*(*nat*⌈*1*/(*1*−μ*1*)⌉) ≤ *k*
with *assms* **have** *1* > *1* / (*real k* ∗ (*1* − μ*1*))
by (*smt* (*verit*, *best*) *divide-less-eq divide-less-eq-1 less-eq-Suc-le natceiling-lessD*)
then have ∗: *1* > *1* / (*real k* ∗ (*1* − *r*)) **if** *r*≤μ*1* **for** *r*
using *that assms k less-le-trans* **by** *fastforce*
have †: *1* / (*k* ∗ (*1* − μ)) ≤ *1* / (*k* ∗ (*1* − μ*1*))

using *μ assms* **by** (*simp add*: *divide-simps mult-less-0-iff*)
obtain *μ<1 k>0* **using** *μ k assms* **by** *force*
then have |*ok-fun-72 μ k*| ≤ |*ok-fun-72 μ1 k*|
using *μ* ∗ *assms* †
by (*simp add*: *ok-fun-72-def abs-mult zero-less-mult-iff abs-of-neg divide-le-cancel*)
then show |*ok-fun-72 μ k*| / *real k* ≤ *e*
by (*smt* (*verit*, *best*) *le-e divide-right-mono of-nat-0-le-iff*)
qed

lemma (**in** *Book*) *X-7-2*:
defines *𝓡* ≡ *Step-class* {*red-step*}
assumes *big*: *Big-X-7-2 μ l*
shows (∏*i*∈*𝓡*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr* (*ok-fun-72 μ k*) ∗ (*1*−*μ*) ^ *card 𝓡*
proof −
define *R* **where** *R* ≡ *RN k* (*nat* ⌈*real l powr* (*3*/*4*)⌉)
have *l34-ge3*: *nat* ⌈*real l powr* (*3*/*4*)⌉ ≥ *3* **and** *k-gt*: *k* > *1* / (*1*−*μ*)
using *big l-le-k* **by** (*auto simp*: *Big-X-7-2-def*)
then obtain *R* > *k k* ≥ *2*
using *μ01 RN-gt1 R-def l-le-k*
by (*smt* (*verit*, *best*) *divide-le-eq-1-pos fact-2 nat-le-real-less of-nat-fact*)
with *k-gt μ01* **have** *bigR*: *1*−*μ* > *1*/*R*
by (*smt* (*verit*, *best*) *less-imp-of-nat-less ln-div ln-le-cancel-iff zero-less-divide-iff*)
have ∗: *1*−*μ* − *1*/*R* ≤ *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)
if *i* ∈ *𝓡* **for** *i*
proof −
let *?NRX* = *λi*. *Neighbours Red* (*cvx i*) ∩ *Xseq i*
have *nextX*: *Xseq* (*Suc i*) = *?NRX i* **and** *nont*: ¬ *termination-condition* (*Xseq i*) (*Yseq i*)
using *that* **by** (*auto simp*: *𝓡-def step-kind-defs next-state-def split*: *prod.split*)
then have *cardX*: *card* (*Xseq i*) > *R*
unfolding *R-def* **by** (*meson not-less termination-condition-def*)
have *1*: *card* (*?NRX i*) ≥ (*1*−*μ*) ∗ *card* (*Xseq i*) − *1*
using *that card-cvx-Neighbours μ01* **by** (*simp add*: *𝓡-def Step-class-def*)
have *R* ≠ *0*
using ‹*k* < *R*› **by** *linarith*
with *cardX* **have** (*1*−*μ*) − *1* / *R* ≤ (*1*−*μ*) − *1* / *card* (*Xseq i*)
by (*simp add*: *inverse-of-nat-le*)
also have ... ≤ *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)
using *cardX nextX 1* **by** (*simp add*: *divide-simps*)
finally show *?thesis* **.**
qed
have *fin-red*: *finite 𝓡*
using *red-step-finite* **by** (*auto simp*: *𝓡-def*)
define *t* **where** *t* ≡ *card 𝓡*
have *t*≥*0*
by (*auto simp*: *t-def*)
have (*1*−*μ* − *1*/*R*) ^ *card Red-steps* ≤ (∏ *i* ∈ *Red-steps*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))

if *Red-steps* ⊆ $\mathcal{R}$ **for** *Red-steps*
using *finite-subset* [*OF that fin-red*] *that*
proof *induction*
case *empty*
then show *?case*
by *auto*
next
case (*insert i Red-steps*)
then have *i*: $i \in \mathcal{R}$
by *auto*
have $((1-\mu) - 1/R)$ ^ *card* (*insert i Red-steps*) = $((1-\mu) - 1/R) * ((1-\mu) - 1/R)$ ^ *card* (*Red-steps*)
by (*simp add*: *insert*)
also have ... ≤ (*card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)) ∗ $((1-\mu) - 1/R)$ ^ *card* (*Red-steps*)
using *bigR* **by** (*intro mult-right-mono* ∗ *i*) *auto*
also have ... ≤ (*card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)) ∗ (∏*i* ∈ *Red-steps*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
using *insert* **by** (*intro mult-left-mono*) *auto*
also have ... = (∏*i*∈*insert i Red-steps*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
using *insert* **by** *simp*
finally show *?case* **.**
qed
then have ∗: $(1-\mu - 1/R)$ ^ *t* ≤ (∏*i* ∈ $\mathcal{R}$. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
using *t-def* **by** *blast*
— Asymptotic part of the argument
have $1-\mu - 1/k \le 1-\mu - 1/R$
using *kn0* ‹$k < R$› **by** (*simp add*: *inverse-of-nat-le*)
then have *ln-le*: *ln* $(1-\mu - 1/k)$ ≤ *ln* $(1-\mu - 1/R)$
using *μ01 k-gt* ‹$R>k$› **by** (*simp add*: *bigR divide-simps mult.commute less-le-trans*)
have *ok-fun-72 μ k* ∗ *ln 2* = *k* ∗ *ln* $(1 - 1 / (k * (1-\mu)))$
by (*simp add*: *ok-fun-72-def*)
also have ... ≤ *t* ∗ *ln* $(1 - 1 / (k * (1-\mu)))$
proof (*intro mult-right-mono-neg*)
have *red-steps*: *card* $\mathcal{R} < k$
using *red-step-limit* ‹$0<\mu$› **by** (*auto simp*: $\mathcal{R}$*-def*)
show *real t* ≤ *real k*
using *nat-less-le red-steps* **by** (*simp add*: *t-def*)
show *ln* $(1 - 1 / (k * (1-\mu))) \le 0$
using *μ01 divide-less-eq k-gt ln-one-minus-pos-upper-bound* **by** *fastforce*
qed
also have ... = *t* ∗ *ln* $((1-\mu - 1/k) / (1-\mu))$
using ‹$t \ge 0$› *μ01* **by** (*simp add*: *diff-divide-distrib*)
also have ... = *t* ∗ (*ln* $(1-\mu - 1/k)$ − *ln* $(1-\mu)$)
using ‹$t \ge 0$› *μ01 k-gt kn0 ln-div* **by** *force*
also have ... ≤ *t* ∗ (*ln* $(1-\mu - 1/R)$ − *ln* $(1-\mu)$)
by (*simp add*: *ln-le mult-left-mono*)
finally have *ok-fun-72 μ k* ∗ *ln 2* + *t* ∗ *ln* $(1-\mu)$ ≤ *t* ∗ *ln* $(1-\mu - 1/R)$

by (*simp add*: *ring-distribs*)
then have *2 powr ok-fun-72 μ k* ∗ (*1*−*μ*) ^ *t* ≤ (*1*−*μ* − *1*/*R*) ^ *t*
using *μ01* **by** (*simp add*: *bigR ln-mult ln-powr ln-realpow flip*: *ln-le-cancel-iff*)
with ∗ **show** *?thesis*
by (*simp add*: *t-def*)
qed

6.3 Lemma 7.3

context *Book*
begin

definition *Bdelta* ≡ λ *μ i*. *Bseq* (*Suc i*) \ *Bseq i*

lemma *card-Bdelta*: *card* (*Bdelta μ i*) = *card* (*Bseq* (*Suc i*)) − *card* (*Bseq i*)
by (*simp add*: *Bseq-mono Bdelta-def card-Diff-subset finite-Bseq*)

lemma *card-Bseq-mono*: *card* (*Bseq* (*Suc i*)) ≥ *card* (*Bseq i*)
by (*simp add*: *Bseq-Suc-subset card-mono finite-Bseq*)

lemma *card-Bseq-sum*: *card* (*Bseq i*) = (∑*j*<*i*. *card* (*Bdelta μ j*))
proof (*induction i*)
case *0*
then show *?case*
by *auto*
next
case (*Suc i*)
with *card-Bseq-mono* **show** *?case*
unfolding *card-Bdelta sum.lessThan-Suc*
by (*smt* (*verit*, *del-insts*) *Nat.add-diff-assoc diff-add-inverse*)
qed

definition *get-blue-book* ≡ λ*i*. *let* (*X*,*Y*,*A*,*B*) = *stepper i in choose-blue-book* (*X*,*Y*,*A*,*B*)

Tracking changes to X and B. The sets are necessarily finite

lemma *Bdelta-bblue-step*:
assumes *i* ∈ *Step-class* {*bblue-step*}
shows ∃ *S* ⊆ *Xseq i*. *Bdelta μ i* = *S*
∧ *card* (*Xseq* (*Suc i*)) ≥ (*μ* ^ *card S*) ∗ *card* (*Xseq i*) / *2*
proof −
obtain *X Y A B S T* **where** *step*: *stepper i* = (*X*,*Y*,*A*,*B*) **and** *bb*: *get-blue-book i* = (*S*,*T*)
and *valid*: *valid-state*(*X*,*Y*,*A*,*B*)
by (*metis surj-pair valid-state-stepper*)
moreover have *finite X*
by (*metis V-state-stepper finX step*)
ultimately have ∗: *stepper* (*Suc i*) = (*T*, *Y*, *A*, *B*∪*S*) ∧ *good-blue-book X* (*S*,*T*)

and *Xeq*: *X* = *Xseq i*

using *assms choose-blue-book-works* [*of X S T Y A B*]
by (*simp-all add*: *step-kind-defs next-state-def valid-state-def get-blue-book-def choose-blue-book-works split*: *if-split-asm*)
show *?thesis*
proof (*intro exI conjI*)
have $S \subseteq X$
proof (*intro choose-blue-book-subset* [*THEN conjunct1*] ‹*finite X*›)
show (*S*, *T*) = *choose-blue-book* (*X*, *Y*, *A*, *B*)
using *bb step* **by** (*simp add*: *get-blue-book-def Xseq-def*)
qed
then show $S \subseteq$ *Xseq i*
using *Xeq* **by** *force*
have *disjnt X B*
using *valid* **by** (*auto simp*: *valid-state-def disjoint-state-def*)
then show *Bdelta* μ *i* = *S*
using ∗ *step* ‹$S \subseteq X$› **by** (*auto simp*: *Bdelta-def Bseq-def disjnt-iff*)
show μ ^ *card S* ∗ *real* (*card* (*Xseq i*)) / *2* ≤ *real* (*card* (*Xseq* (*Suc i*)))
using ∗ **by** (*auto simp*: *Xseq-def good-blue-book-def step*)
qed
qed

lemma *Bdelta-dboost-step*:
assumes $i \in$ *Step-class* {*dboost-step*}
shows $\exists x \in$ *Xseq i*. *Bdelta* μ *i* = {*x*}
proof −
obtain *X Y A B* **where** *step*: *stepper i* = (*X*,*Y*,*A*,*B*) **and** *valid*: *valid-state*(*X*,*Y*,*A*,*B*)
by (*metis surj-pair valid-state-stepper*)
have *cvx*: *choose-central-vx* (*X*,*Y*,*A*,*B*) ∈ *X*
by (*metis Step-class-insert Un-iff cvx-def cvx-in-Xseq assms step stepper-XYseq*)
then have ∃ *X′ Y′*. *stepper* (*Suc i*) = (*X′*, *Y′*, *A*, *insert* (*choose-central-vx* (*X*,*Y*,*A*,*B*)) *B*)
using *assms step*
by (*auto simp*: *step-kind-defs next-state-def split*: *if-split-asm*)
moreover have *choose-central-vx* (*X*,*Y*,*A*,*B*) ∉ *B*
using *valid cvx* **by** (*force simp*: *valid-state-def disjoint-state-def disjnt-iff*)
ultimately show *?thesis*
using *step cvx* **by** (*auto simp*: *Bdelta-def Bseq-def disjnt-iff Xseq-def*)
qed

lemma *card-Bdelta-dboost-step*:
assumes $i \in$ *Step-class* {*dboost-step*}
shows *card* (*Bdelta* μ *i*) = *1*
using *Bdelta-dboost-step* [*OF assms*] **by** *force*

lemma *Bdelta-trivial-step*:
assumes *i*: $i \in$ *Step-class* {*red-step*,*dreg-step*,*halted*}
shows *Bdelta* μ *i* = {}
using *assms*
by (*auto simp*: *step-kind-defs next-state-def Bdelta-def degree-reg-def split*: *if-split-asm*

```
prod.split)

end

definition ok-fun-73 ≡ λk. − (real k powr (3/4))

lemma ok-fun-73: ok-fun-73 ∈ o(real)
  unfolding ok-fun-73-def by real-asymp

lemma (in Book) X-7-3:
  assumes big: Big-Blue-4-1 μ l
  defines ℬ ≡ Step-class {bblue-step}
  defines 𝒮 ≡ Step-class {dboost-step}
  shows (∏ i ∈ ℬ. card (Xseq(Suc i)) / card (Xseq i)) ≥ 2 powr (ok-fun-73 k) *
μ ^ (l − card 𝒮)
proof −
  have [simp]: finite ℬ finite 𝒮 and cardℬ: card ℬ ≤ l powr (3/4)
    using assms bblue-step-limit big by (auto simp: ℬ-def 𝒮-def)
  define b where b ≡ λi. card (Bdelta μ i)
  obtain i where card (Bseq i) = sum b ℬ + card 𝒮
  proof −
    define i where i = Suc (Max (ℬ ∪ 𝒮))
    define TRIV where TRIV ≡ Step-class {red-step,dreg-step,halted} ∩ {..<i}
    have [simp]: finite TRIV
      by (auto simp: TRIV-def)
    have eq: ℬ ∪ 𝒮 ∪ TRIV = {..<i}
    proof
      show ℬ ∪ 𝒮 ∪ TRIV ⊆ {..<i}
        by (auto simp: i-def TRIV-def less-Suc-eq-le)
      show {..<i} ⊆ ℬ ∪ 𝒮 ∪ TRIV
       using stepkind.exhaust by (auto simp: ℬ-def 𝒮-def TRIV-def Step-class-def)
    qed
    have dis: ℬ ∩ 𝒮 = {} (ℬ ∪ 𝒮) ∩ TRIV = {}
      by (auto simp: ℬ-def 𝒮-def TRIV-def Step-class-def)
    show thesis
    proof
      have card (Bseq i) = (∑ j ∈ ℬ ∪ 𝒮 ∪ TRIV. b j)
        using card-Bseq-sum eq unfolding b-def by metis
      also have ... = (∑ j∈ℬ. b j) + (∑ j∈𝒮. b j) + (∑ j∈TRIV. b j)
        by (simp add: sum-Un-nat dis)
      also have ... = sum b ℬ + card 𝒮
      by (simp add: b-def 𝒮-def card-Bdelta-dboost-step TRIV-def Bdelta-trivial-step)
      finally show card (Bseq i) = sum b ℬ + card 𝒮 .
    qed
  qed
  then have sum-b-ℬ: sum b ℬ ≤ l − card 𝒮
    by (metis Bseq-less-l less-diff-conv nat-less-le)
  have real (card ℬ) ≤ real k powr (3/4)
    using cardℬ l-le-k
```

by (*smt* (*verit*, *best*) *divide-nonneg-pos of-nat-0-le-iff of-nat-mono powr-mono2*)
then have *2 powr* (*ok-fun-73 k*) ≤ (*1/2*) ^ *card* $\mathcal{B}$
by (*simp add*: *ok-fun-73-def powr-minus divide-simps flip*: *powr-realpow*)
then have *2 powr* (*ok-fun-73 k*) ∗ μ ^ (*l* − *card* $\mathcal{S}$) ≤ (*1/2*) ^ *card* $\mathcal{B}$ ∗ μ ^ (*l* − *card* $\mathcal{S}$)
by (*simp add*: μ*01*)
also have (*1/2*) ^ *card* $\mathcal{B}$ ∗ μ ^ (*l* − *card* $\mathcal{S}$) ≤ (*1/2*) ^ *card* $\mathcal{B}$ ∗ μ ^ (*sum b* $\mathcal{B}$)
using μ*01 sum-b-*$\mathcal{B}$ **by** *simp*
also have ... = (∏ *i*∈$\mathcal{B}$. μ ^ *b i* / *2*)
by (*simp add*: *power-sum prod-dividef divide-simps*)
also have ... ≤ (∏ *i*∈$\mathcal{B}$. *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*))
proof (*rule prod-mono*)
fix *i* :: *nat*
assume *i* ∈ $\mathcal{B}$
then have ¬ *termination-condition* (*Xseq i*) (*Yseq i*)
by (*simp add*: $\mathcal{B}$*-def Step-class-def flip*: *step-non-terminating-iff*)
then have *card* (*Xseq i*) ≠ *0*
using *termination-condition-def* **by** *force*
with ‹*i*∈$\mathcal{B}$› μ*01* **show** *0* ≤ μ ^ *b i* / *2* ∧ μ ^ *b i* / *2* ≤ *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)
by (*force simp*: *b-def* $\mathcal{B}$*-def divide-simps dest*!: *Bdelta-bblue-step*)
qed
finally show *?thesis* **.**
qed

6.4 Lemma 7.5

Small $o(k)$ bounds on summations for this section

This is the explicit upper bound for heights given just below (5) on page 9

definition *ok-fun-26* ≡ λ*k*. *2* ∗ *ln k* / *eps k*

definition *ok-fun-28* ≡ λ*k*. −*2* ∗ *real k powr* (*7/8*)

lemma *ok-fun-26*: *ok-fun-26* ∈ *o*(*real*) **and** *ok-fun-28*: *ok-fun-28* ∈ *o*(*real*)
unfolding *ok-fun-26-def ok-fun-28-def eps-def* **by** *real-asymp*+

definition
Big-X-7-5 ≡
λμ *l*. *Big-Blue-4-1* μ *l* ∧ *Big-Red-5-3* μ *l* ∧ *Big-Y-6-5-Bblue l*
∧ (∀ *k*≥*l*. *Big-height-upper-bound k* ∧ *k*≥*16* ∧ (*ok-fun-26 k* − *ok-fun-28 k* ≤ *k*))

establishing the size requirements for 7.5

lemma *Big-X-7-5*:
assumes *0*<μ*0* μ*1*<*1*
shows ∀$^\infty$*l*. ∀μ. μ ∈ {μ*0*..μ*1*} ⟶ *Big-X-7-5* μ *l*

```
proof −
  have ok: ∀∞l. ok-fun-26 l − ok-fun-28 l ≤ l
    unfolding eps-def ok-fun-26-def ok-fun-28-def by real-asymp
  show ?thesis
    using assms Big-Y-6-5-Bblue Big-Red-5-3 Big-Blue-4-1
    unfolding Big-X-7-5-def
    apply (simp add: eventually-conj-iff all-imp-conj-distrib)
      apply (intro conjI strip eventually-all-ge-at-top ok Big-height-upper-bound;
real-asymp)
    done
qed

context Book
begin

lemma X-26-and-28:
  assumes big: Big-X-7-5 μ l
  defines 𝒟 ≡ Step-class {dreg-step}
  defines ℬ ≡ Step-class {bblue-step}
  defines ℋ ≡ Step-class {halted}
  defines h ≡ λi. real (hgt (pseq i))
  obtains (∑ i∈{..<halted-point} \ 𝒟. h (Suc i) − h (i−1)) ≤ ok-fun-26 k
          ok-fun-28 k ≤ (∑ i ∈ ℬ. h(Suc i) − h(i−1))
proof −
  define 𝒮 where 𝒮 ≡ Step-class {dboost-step}
  have B-limit: Big-Blue-4-1 μ l and bigY65B: Big-Y-6-5-Bblue l
    and hub: Big-height-upper-bound k
    using big l-le-k by (auto simp: Big-X-7-5-def)
  have m-minimal: i ∉ ℋ ⟷ i < halted-point for i
    unfolding ℋ-def using halted-point-minimal assms by blast
  have oddset: {..<halted-point} \ 𝒟 = {i ∈ {..<halted-point}. odd i}
    using m-minimal step-odd step-even not-halted-even-dreg
    by (auto simp: 𝒟-def ℋ-def Step-class-insert-NO-MATCH)
      — working on 28
  have ok-fun-28 k ≤ −2 ∗ ε powr (−1/2) ∗ card ℬ
  proof −
    have k powr (1/8) ∗ card ℬ ≤ k powr (1/8) ∗ l powr (3/4)
      using B-limit bblue-step-limit by (simp add: ℬ-def mult-left-mono)
    also have ... ≤ k powr (1/8) ∗ k powr (3/4)
      by (simp add: l-le-k mult-mono powr-mono2)
    also have ... = k powr (7/8)
      by (simp flip: powr-add)
    finally show ?thesis
      by (simp add: eps-def powr-powr ok-fun-28-def)
  qed
  also have ... ≤ (∑ i ∈ ℬ. h(Suc i) − h(i−1))
  proof −
    have (∑ i ∈ ℬ. −2 ∗ ε powr (−1/2)) ≤ (∑ i ∈ ℬ. h(Suc i) − h(i−1))
    proof (rule sum-mono)
```

fix *i* :: *nat*
assume *i*: $i \in \mathcal{B}$
show $-2 * \varepsilon$ *powr* $(-1/2) \leq h(Suc\ i) - h(i-1)$
using *bigY65B kn0 i Y-6-5-Bblue* **by** (*fastforce simp*: $\mathcal{B}$*-def h-def*)
qed
then show *?thesis*
by (*simp add*: *mult.commute*)
qed
finally have *28*: *ok-fun-28 k* $\leq (\sum i \in \mathcal{B}.\ h(Suc\ i) - h(i-1))$ **.**
have $(\sum i \in \{..<halted\text{-}point\} \setminus \mathcal{D}.\ h(Suc\ i) - h(i-1)) \leq h\ halted\text{-}point - h\ 0$
proof (*cases even halted-point*)
case *False*
have *hgt* (*pseq* (*halted-point* − *Suc 0*)) ≤ *hgt* (*pseq halted-point*)
using *Y-6-5-DegreeReg* [*of halted-point*−*1*] *False m-minimal not-halted-even-dreg odd-pos*
by (*fastforce simp*: $\mathcal{H}$*-def*)
then have *h*(*halted-point* − *Suc 0*) ≤ *h halted-point*
using *h-def of-nat-mono* **by** *blast*
with *False* **show** *?thesis*
by (*simp add*: *oddset sum-odds-odd*)
qed (*simp add*: *oddset sum-odds-even*)
also have ... ≤ *ok-fun-26 k*
proof −
have *hgt* (*pseq i*) ≥ *1* **for** *i*
by (*simp add*: *Suc-leI hgt-gt0*)
moreover have *hgt* (*pseq halted-point*) ≤ *ok-fun-26 k*
using *hub pee-le1 height-upper-bound* **unfolding** *ok-fun-26-def* **by** *blast*
ultimately show *?thesis*
by (*simp add*: *h-def*)
qed
finally have *26*: $(\sum i \in \{..<halted\text{-}point\} \setminus \mathcal{D}.\ h\ (Suc\ i) - h\ (i-1)) \leq ok\text{-}fun\text{-}26\ k$ **.**
with *28* **show** *?thesis*
using *that* **by** *blast*
qed

proposition *X-7-5*:
assumes μ: $0<\mu\ \mu<1$
defines $\mathcal{S} \equiv$ *Step-class* {*dboost-step*} **and** $\mathcal{SS} \equiv$ *dboost-star*
assumes *big*: *Big-X-7-5* μ *l*
shows *card* $(\mathcal{S} \setminus \mathcal{SS}) \leq 3 * \varepsilon$ *powr* $(1/4) * k$
proof −
define $\mathcal{D}$ **where** $\mathcal{D} \equiv$ *Step-class* {*dreg-step*}
define $\mathcal{R}$ **where** $\mathcal{R} \equiv$ *Step-class* {*red-step*}
define $\mathcal{B}$ **where** $\mathcal{B} \equiv$ *Step-class* {*bblue-step*}
define *h* **where** $h \equiv \lambda i.$ *real* (*hgt* (*pseq i*))
obtain *26*: $(\sum i \in \{..<halted\text{-}point\} \setminus \mathcal{D}.\ h\ (Suc\ i) - h\ (i-1)) \leq ok\text{-}fun\text{-}26\ k$
and *28*: *ok-fun-28 k* $\leq (\sum i \in \mathcal{B}.\ h(Suc\ i) - h(i-1))$
using *X-26-and-28 assms*(*1*−*3*) *big*

unfolding *B-def D-def h-def Big-X-7-5-def* **by** *blast*
have *SS*: $\mathcal{SS} = \{i \in \mathcal{S}.\ h(Suc\ i) - h\ i \le \varepsilon\ powr\ (-1/4)\}$ **and** $\mathcal{SS} \subseteq \mathcal{S}$
by (*auto simp*: *SS-def S-def dboost-star-def h-def*)
have *in-S*: $h(Suc\ i) - h\ i > \varepsilon\ powr\ (-1/4)$ **if** $i \in \mathcal{S} \backslash \mathcal{SS}$ **for** *i*
using *that* **by** (*fastforce simp*: *SS*)
have *B-limit*: *Big-Blue-4-1* μ *l*
and *bigR53*: *Big-Red-5-3* μ *l*
and *16*: $k \ge 16$
and *ok-fun*: $ok\text{-}fun\text{-}26\ k - ok\text{-}fun\text{-}28\ k \le k$
using *big l-le-k* **by** (*auto simp*: *Big-X-7-5-def*)
have [*simp*]: *finite* $\mathcal{R}$ *finite* $\mathcal{B}$ *finite* $\mathcal{S}$
using *finite-components* **by** (*auto simp*: *R-def B-def S-def*)
have [*simp*]: $\mathcal{R} \cap \mathcal{S} = \{\}$ $\mathcal{B} \cap (\mathcal{R} \cup \mathcal{S}) = \{\}$
by (*auto simp*: *R-def S-def B-def Step-class-def*)

obtain *cardss*: $card\ \mathcal{SS} \le card\ \mathcal{S}$ $card\ (\mathcal{S} \backslash \mathcal{SS}) = card\ \mathcal{S} - card\ \mathcal{SS}$
by (*meson* ‹$\mathcal{SS} \subseteq \mathcal{S}$› ‹*finite* $\mathcal{S}$› *card-Diff-subset card-mono infinite-super*)
have $(\sum i \in \mathcal{S}.\ h(Suc\ i) - h(i-1)) \ge \varepsilon\ powr\ (-1/4) * card\ (\mathcal{S} \backslash \mathcal{SS})$
proof −
have $(\sum i \in \mathcal{S} \backslash \mathcal{SS}.\ h(Suc\ i) - h(i-1)) \ge (\sum i \in \mathcal{S} \backslash \mathcal{SS}.\ \varepsilon\ powr\ (-1/4))$
proof (*rule sum-mono*)
fix *i* :: *nat*
assume *i*: $i \in \mathcal{S} \backslash \mathcal{SS}$
with *i* **obtain** $i-1 \in \mathcal{D}$ $i>0$
using *dreg-before-step1 dreg-before-gt0* **by** (*fastforce simp*: *S-def D-def Step-class-insert-NO-MATCH*)
with *i* **show** $\varepsilon\ powr\ (-1/4) \le h(Suc\ i) - h(i-1)$
using *in-S*[*of i*] *Y-6-5-DegreeReg*[*of i−1*] **by** (*simp add*: *D-def h-def*)
qed
moreover
have $(\sum i \in \mathcal{SS}.\ h(Suc\ i) - h(i-1)) \ge 0$
proof (*intro sum-nonneg*)
show $\bigwedge i.\ i \in \mathcal{SS} \Longrightarrow 0 \le h\ (Suc\ i) - h\ (i - 1)$
using *Y-6-4-dbooSt* μ *bigR53* **by**(*auto simp*: *h-def SS S-def hgt-mono*)
qed
ultimately show *?thesis*
by (*simp add*: *mult.commute sum.subset-diff* [*OF* ‹$\mathcal{SS} \subseteq \mathcal{S}$› ‹*finite* $\mathcal{S}$›])
qed
moreover
have $(\sum i \in \mathcal{R}.\ h(Suc\ i) - h(i-1)) \ge (\sum i \in \mathcal{R}.\ -2)$
proof (*rule sum-mono*)
fix *i* :: *nat*
assume *i*: $i \in \mathcal{R}$
with *i* **obtain** $i-1 \in \mathcal{D}$ $i>0$
using *dreg-before-step1 dreg-before-gt0*
by (*fastforce simp*: *R-def D-def Step-class-insert-NO-MATCH*)
with *i* **have** $hgt\ (pseq\ (i-1)) - 2 \le hgt\ (pseq\ (Suc\ i))$
using *Y-6-5-Red*[*of i*] *16 Y-6-5-DegreeReg*[*of i−1*]
by (*fastforce simp*: *algebra-simps R-def D-def*)

then show $- 2 \le h(Suc\ i) - h(i-1)$
unfolding *h-def* **by** *linarith*
qed
ultimately have *27*: $(\sum i \in \mathcal{R}\cup\mathcal{S}.\ h(Suc\ i) - h(i-1)) \ge \varepsilon\ powr\ (-1/4) * card\ (\mathcal{S}\backslash\mathcal{SS}) - 2 * card\ \mathcal{R}$
by (*simp add*: *sum.union-disjoint*)

have $ok\text{-}fun\text{-}28\ k + (\varepsilon\ powr\ (-1/4) * card\ (\mathcal{S}\backslash\mathcal{SS}) - 2 * card\ \mathcal{R}) \le (\sum i \in \mathcal{B}.\ h(Suc\ i) - h(i-1)) + (\sum i \in \mathcal{R}\cup\mathcal{S}.\ h(Suc\ i) - h(i-1))$
using *27 28* **by** *simp*
also have $\ldots = (\sum i \in \mathcal{B} \cup (\mathcal{R}\cup\mathcal{S}).\ h(Suc\ i) - h(i-1))$
by (*simp add*: *sum.union-disjoint*)
also have $\ldots = (\sum i \in \{..<halted\text{-}point\} \setminus \mathcal{D}.\ h(Suc\ i) - h(i-1))$
proof −
have $i \in \mathcal{B} \cup (\mathcal{R}\cup\mathcal{S})$ **if** $i < halted\text{-}point\ i \notin \mathcal{D}$ **for** *i*
using *that* **unfolding** *$\mathcal{D}$-def $\mathcal{B}$-def $\mathcal{R}$-def $\mathcal{S}$-def*
using *Step-class-cases halted-point-minimal* **by** *auto*
moreover
have $i \in \{..<halted\text{-}point\} \setminus \mathcal{D}$ **if** $i \in \mathcal{B} \cup (\mathcal{R}\cup\mathcal{S})$ **for** *i*
using *halted-point-minimal′ that* **by** (*force simp*: *$\mathcal{D}$-def $\mathcal{B}$-def $\mathcal{R}$-def $\mathcal{S}$-def Step-class-def*)
ultimately have $\mathcal{B} \cup (\mathcal{R}\cup\mathcal{S}) = \{..<halted\text{-}point\} \setminus \mathcal{D}$
by *auto*
then show *?thesis*
by *simp*
qed
finally have $ok\text{-}fun\text{-}28\ k + (\varepsilon\ powr\ (-1/4) * card\ (\mathcal{S}\backslash\mathcal{SS}) - real\ (2 * card\ \mathcal{R})) \le ok\text{-}fun\text{-}26\ k$
using *26* **by** *simp*
then have $real\ (card\ (\mathcal{S} \setminus \mathcal{SS})) \le (ok\text{-}fun\text{-}26\ k - ok\text{-}fun\text{-}28\ k + 2 * card\ \mathcal{R}) * \varepsilon\ powr\ (1/4)$
using *eps-gt0* **by** (*simp add*: *powr-minus field-simps del*: *div-add div-mult-self3*)
moreover have $card\ \mathcal{R} < k$
using *red-step-limit* μ **unfolding** *$\mathcal{R}$-def* **by** *blast*
ultimately have $card\ (\mathcal{S}\backslash\mathcal{SS}) \le (k + 2 * k) * \varepsilon\ powr\ (1/4)$
by (*smt* (*verit*, *best*) *of-nat-add mult-2 mult-right-mono nat-less-real-le ok-fun powr-ge-zero*)
then show *?thesis*
by (*simp add*: *algebra-simps*)
qed

end

6.5 Lemma 7.4

definition
$Big\text{-}X\text{-}7\text{-}4 \equiv \lambda\mu\ l.\ Big\text{-}X\text{-}7\text{-}5\ \mu\ l \wedge Big\text{-}Red\text{-}5\text{-}3\ \mu\ l$

establishing the size requirements for 7.4

lemma *Big-X-7-4*:

assumes $0<\mu 0$ $\mu 1<1$
shows $\forall^\infty l.\ \forall \mu.\ \mu \in \{\mu 0..\mu 1\} \longrightarrow$ *Big-X-7-4 μ l*
using *assms Big-X-7-5 Big-Red-5-3*
unfolding *Big-X-7-4-def*
by (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)

definition *ok-fun-74* ≡ $\lambda k.\ -6 * eps\ k\ powr\ (1/4) * k * ln\ k\ /\ ln\ 2$

lemma *ok-fun-74*: *ok-fun-74* ∈ *o*(*real*)
unfolding *ok-fun-74-def eps-def* **by** *real-asymp*

context *Book*
begin

lemma *X-7-4*:
assumes *big*: *Big-X-7-4 μ l*
defines $\mathcal{S} \equiv$ *Step-class* {*dboost-step*}
shows ($\prod i{\in}\mathcal{S}$. *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr ok-fun-74 k* ∗ *bigbeta* ^ *card* $\mathcal{S}$
proof −
define $\mathcal{SS}$ **where** $\mathcal{SS}$ ≡ *dboost-star*
then have *big53*: *Big-Red-5-3 μ l* **and** *X75*: *card* ($\mathcal{S}\backslash\mathcal{SS}$) ≤ *3* ∗ *ε powr* (*1/4*) ∗ *k*
using *μ01 big* **by** (*auto simp*: *Big-X-7-4-def X-7-5* $\mathcal{S}$*-def* $\mathcal{SS}$*-def*)
then have *R53*: *pseq* (*Suc i*) ≥ *pseq i* ∧ *beta i* ≥ *1* / (*real k*)^2 **and** *beta-gt0*: *0* < *beta i*
if $i \in \mathcal{S}$ **for** *i*
using *that Red-5-3 beta-gt0* **by** (*auto simp*: $\mathcal{S}$*-def*)
have *bigbeta01*: *bigbeta* ∈ {*0*<..<*1*}
using *big53 assms bigbeta-gt0 bigbeta-less1* **by** *force*
have $\mathcal{SS} \subseteq \mathcal{S}$
unfolding $\mathcal{SS}$*-def* $\mathcal{S}$*-def dboost-star-def* **by** *auto*
then obtain [*simp*]: *finite* $\mathcal{S}$ *finite* $\mathcal{SS}$
by (*simp add*: $\mathcal{SS}$*-def* $\mathcal{S}$*-def finite-dboost-star*)
have *card-SSS*: *card* $\mathcal{SS}$ ≤ *card* $\mathcal{S}$
by (*metis* $\mathcal{SS}$*-def* $\mathcal{S}$*-def* ‹*finite* $\mathcal{S}$› *card-mono dboost-star-subset*)
have *β*: *beta i* = *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*) **if** $i \in \mathcal{S}$ **for** *i*
proof −
have *Xseq* (*Suc i*) = *Neighbours Blue* (*cvx i*) ∩ *Xseq i*
using *that* **unfolding** $\mathcal{S}$*-def*
by (*auto simp*: *step-kind-defs next-state-def split*: *prod.split*)
then show *?thesis*
by (*force simp*: *beta-eq*)
qed
then have ∗: ($\prod i{\in}\mathcal{S}$. *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)) = ($\prod i{\in}\mathcal{S}$. *beta i*)
by *force*
have *prod-beta-gt0*: *prod* (*beta*) *S'* > *0* **if** $S' \subseteq \mathcal{S}$ **for** *S'*
using *beta-gt0 that*

```
    by (force simp: beta-ge0 intro: prod-pos)
      — bounding the immoderate steps
  have (∏ i∈S\SS. 1 / beta i) ≤ (∏ i∈S\SS. real k ^ 2)
  proof (rule prod-mono)
    fix i
    assume i: i ∈ S \ SS
    with R53 kn0 beta-ge0 [of i] show 0 ≤ 1 / beta i ∧ 1 / beta i ≤ (real k)^2
      by (force simp: R53 divide-simps mult.commute)
  qed
  then have (∏ i∈S\SS. 1 / beta i) ≤ real k ^ (2 ∗ card(S\SS))
    by (simp add: power-mult)
  also have ... = real k powr (2 ∗ card(S\SS))
    by (metis kn0 of-nat-0-less-iff powr-realpow)
  also have ... ≤ k powr (2 ∗ 3 ∗ ε powr (1/4) ∗ k)
    using X75 kn0 by (intro powr-mono; linarith)
  also have ... ≤ exp (6 ∗ ε powr (1/4) ∗ k ∗ ln k)
    by (simp add: powr-def)
  also have ... = 2 powr −ok-fun-74 k
    by (simp add: ok-fun-74-def powr-def)
  finally have (∏ i∈S\SS. 1 / beta i) ≤ 2 powr −ok-fun-74 k .
  then have A: (∏ i∈S\SS. beta i) ≥ 2 powr ok-fun-74 k
    using prod-beta-gt0[of S\SS]
    by (simp add: powr-minus prod-dividef mult.commute divide-simps)
— bounding the moderate steps
  have (∏ i∈SS. 1 / beta i) ≤ bigbeta powr (− (card SS))
  proof (cases SS = {})
    case True
    with bigbeta01 show ?thesis
      by fastforce
  next
    case False
    then have card SS > 0
      using ‹finite SS› card-0-eq by blast
    have (∏ i∈SS. 1 / beta i) powr (1 / card SS) ≤ (∑ i∈SS. 1 / beta i / card
SS)
    proof (rule arith-geom-mean [OF ‹finite SS› ‹SS ≠ {}›])
      show ⋀i. i ∈ SS ⟹ 0 ≤ 1 / beta i
        by (simp add: beta-ge0)
    qed
    then have ((∏ i∈SS. 1 / beta i) powr (1 / card SS)) powr (card SS)
        ≤ (∑ i∈SS. 1 / beta i / card SS) powr (card SS)
      using powr-mono2 by auto
    with ‹SS ≠ {}›
    have (∏ i∈SS. 1 / beta i) ≤ (∑ i∈SS. 1 / beta i / card SS) powr (card SS)
      by (simp add: powr-powr beta-ge0 prod-nonneg)
    also have ... ≤ (1 / (card SS) ∗ (∑ i∈SS. 1 / beta i)) powr (card SS)
      using ‹card SS > 0› by (simp add: field-simps sum-divide-distrib)
    also have ... ≤ bigbeta powr (− (card SS))
      using ‹SS ≠ {}› ‹card SS > 0›
```

by (*simp add*: *bigbeta-def field-simps powr-minus powr-divide beta-ge0 sum-nonneg flip*: *𝒮𝒮-def*)
finally show *?thesis* **.**
qed
then have *B*: $(\prod i{\in}\mathcal{SS}.\ beta\ i) \geq bigbeta\ powr\ (card\ \mathcal{SS})$
using ‹$\mathcal{SS} \subseteq \mathcal{S}$› *prod-beta-gt0*[*of* 𝒮𝒮] *bigbeta01*
by (*simp add*: *powr-minus prod-dividef mult.commute divide-simps*)
have $2\ powr\ ok\text{-}fun\text{-}74\ k * bigbeta\ powr\ card\ \mathcal{S} \leq 2\ powr\ ok\text{-}fun\text{-}74\ k * bigbeta\ powr\ card\ \mathcal{SS}$
using *bigbeta01 big53 card-SSS* **by** (*simp add*: *powr-mono'*)
also have $\ldots \leq (\prod i{\in}\mathcal{S}\backslash\mathcal{SS}.\ beta\ i) * (\prod i{\in}\mathcal{SS}.\ beta\ i)$
using *beta-ge0* **by** (*intro mult-mono A B*) (*auto simp*: *prod-nonneg*)
also have $\ldots = (\prod i{\in}\mathcal{S}.\ beta\ i)$
by (*metis* ‹$\mathcal{SS} \subseteq \mathcal{S}$› ‹*finite* 𝒮› *prod.subset-diff*)
finally have $2\ powr\ ok\text{-}fun\text{-}74\ k * bigbeta\ powr\ real\ (card\ \mathcal{S}) \leq prod\ (beta)\ \mathcal{S}$ **.**
with *bigbeta01* **show** *?thesis*
by (*simp add*: * *powr-realpow*)
qed

6.6 Observation 7.7

lemma *X-7-7*:
assumes *i*: $i \in Step\text{-}class\ \{dreg\text{-}step\}$
defines $q \equiv \varepsilon\ powr\ (-1/2) * alpha\ (hgt\ (pseq\ i))$
shows $pseq\ (Suc\ i) - pseq\ i \geq card\ (Xseq\ i \setminus Xseq\ (Suc\ i))\ /\ card\ (Xseq\ (Suc\ i)) * q \wedge card\ (Xseq\ (Suc\ i)) > 0$
proof −
have *finX*: *finite* (*Xseq i*) **for** *i*
using *finite-Xseq* **by** *blast*
define *Y* **where** $Y \equiv Yseq$
have $Xseq\ (Suc\ i) = \{x \in Xseq\ i.\ red\text{-}dense\ (Y\ i)\ (red\text{-}density\ (Xseq\ i)\ (Y\ i))\ x\}$
and *Y*: $Y\ (Suc\ i) = Y\ i$
using *i*
by (*simp-all add*: *step-kind-defs next-state-def X-degree-reg-def degree-reg-def Y-def split*: *if-split-asm prod.split-asm*)
then have *Xseq*: $Xseq\ (Suc\ i) = \{x \in Xseq\ i.\ card\ (Neighbours\ Red\ x \cap Y\ i) \geq (pseq\ i - q) * card\ (Y\ i)\}$
by (*simp add*: *red-dense-def q-def pseq-def Y-def*)
have *Xsub*[*simp*]: $Xseq\ (Suc\ i) \subseteq Xseq\ i$
using *Xseq-Suc-subset* **by** *blast*
then have *card-le*: $card\ (Xseq\ (Suc\ i)) \leq card\ (Xseq\ i)$
by (*simp add*: *card-mono finX*)
have [*simp*]: *disjnt* (*Xseq i*) (*Y i*)
using *Xseq-Yseq-disjnt Y-def* **by** *blast*
have *Xnon0*: $card\ (Xseq\ i) > 0$ **and** *Ynon0*: $card\ (Y\ i) > 0$
using *i* **by** (*simp-all add*: *Y-def Xseq-gt0 Yseq-gt0 Step-class-def*)
have $alpha\ (hgt\ (pseq\ i)) > 0$
by (*simp add*: *alpha-gt0 kn0 hgt-gt0*)

with *kn0* **have** *q > 0*
  by (*smt* (*verit*) *q-def eps-gt0 mult-pos-pos powr-gt-zero*)
have *Xdif*: *Xseq i* \ *Xseq* (*Suc i*) = {*x* ∈ *Xseq i*. *card* (*Neighbours Red x* ∩ *Y i*) < (*pseq i* − *q*) ∗ *card* (*Y i*)}
  using *Xseq* **by** *force*
have *disYX*: *disjnt* (*Y i*) (*Xseq i* \ *Xseq* (*Suc i*))
  by (*metis Diff-subset* ‹*disjnt* (*Xseq i*) (*Y i*)› *disjnt-subset2 disjnt-sym*)
have *edge-card Red* (*Y i*) (*Xseq i* \ *Xseq* (*Suc i*))
    = (∑*x* ∈ *Xseq i* \ *Xseq* (*Suc i*). *real* (*card* (*Neighbours Red x* ∩ *Y i*)))
  using *edge-card-eq-sum-Neighbours* [*OF - - disYX*] *finX Red-E* **by** *simp*
also have ... ≤ (∑*x* ∈ *Xseq i* \ *Xseq* (*Suc i*). (*pseq i* − *q*) ∗ *card* (*Y i*))
  by (*smt* (*verit*, *del-insts*) *Xdif mem-Collect-eq sum-mono*)
finally have *A*: *edge-card Red* (*Xseq i* \ *Xseq* (*Suc i*)) (*Y i*) ≤ *card* (*Xseq i* \ *Xseq* (*Suc i*)) ∗ (*pseq i* − *q*) ∗ *card* (*Y i*)
  by (*simp add*: *edge-card-commute*)
then have *False* **if** *Xseq* (*Suc i*) = {}
  using ‹*q*>*0*› *Xnon0 Ynon0 that* **by** (*simp add*: *edge-card-eq-pee Y-def mult-le-0-iff*)
then have *XSnon0*: *card* (*Xseq* (*Suc i*)) > *0*
  using *card-gt-0-iff finX* **by** *blast*
have *pseq i* ∗ *card* (*Xseq i*) ∗ *real* (*card* (*Y i*)) − *edge-card Red* (*Xseq* (*Suc i*)) (*Y i*)
    ≤ *card* (*Xseq i* \ *Xseq* (*Suc i*)) ∗ (*pseq i* − *q*) ∗ *card* (*Y i*)
  by (*metis A edge-card-eq-pee edge-card-mono Y-def Xsub* ‹*disjnt* (*Xseq i*) (*Y i*)› *edge-card-diff finX of-nat-diff*)
moreover have *real* (*card* (*Xseq* (*Suc i*))) ≤ *real* (*card* (*Xseq i*))
  using *Xsub* **by** (*simp add*: *card-le*)
ultimately have §: *edge-card Red* (*Xseq* (*Suc i*)) (*Y i*) ≥ *pseq i* ∗ *card* (*Xseq* (*Suc i*)) ∗ *card* (*Y i*) + *card* (*Xseq i* \ *Xseq* (*Suc i*)) ∗ *q* ∗ *card* (*Y i*)
  using *Xnon0*
  by (*smt* (*verit*, *del-insts*) *Xsub card-Diff-subset card-gt-0-iff card-le left-diff-distrib finite-subset mult-of-nat-commute of-nat-diff*)
have *edge-card Red* (*Xseq* (*Suc i*)) (*Y i*) / (*card* (*Xseq* (*Suc i*)) ∗ *card* (*Y i*)) ≥ *pseq i* + *card* (*Xseq i* \ *Xseq* (*Suc i*)) ∗ *q* / *card* (*Xseq* (*Suc i*))
  using *divide-right-mono* [*OF* §, *of card* (*Xseq* (*Suc i*)) ∗ *card* (*Y i*)] *XSnon0 Ynon0*
  by (*simp add*: *add-divide-distrib split*: *if-split-asm*)
moreover have *pseq* (*Suc i*) = *real* (*edge-card Red* (*Xseq* (*Suc i*)) (*Y i*)) / (*real* (*card* (*Y i*)) ∗ *real* (*card* (*Xseq* (*Suc i*))))
  using *Y* **by** (*simp add*: *pseq-def gen-density-def Y-def*)
ultimately show *?thesis*
  by (*simp add*: *algebra-simps XSnon0*)
qed

end

6.7 Lemma 7.8

definition *Big-X-7-8* ≡ λ*k*. *k*≥*2* ∧ *eps k powr* (*1/2*) / *k* ≥ *2* / *k*^*2*

```
lemma Big-X-7-8: ∀∞k. Big-X-7-8 k
  unfolding eps-def Big-X-7-8-def eventually-conj-iff eps-def
  by (intro conjI; real-asymp)

lemma (in Book) X-7-8:
  assumes big: Big-X-7-8 k
    and i: i ∈ Step-class {dreg-step}
  shows card (Xseq (Suc i)) ≥ card (Xseq i) / k^2
proof −
  define q where q ≡ ε powr (−1/2) ∗ alpha (hgt (pseq i))
  have k>0 ‹k≥2› using big by (auto simp: Big-X-7-8-def)
  have 2 / k^2 ≤ ε powr (1/2) / k
    using big by (auto simp: Big-X-7-8-def)
  also have . . . ≤ q
    using kn0 eps-gt0 Red-5-7a [of pseq i]
    by (simp add: q-def powr-minus divide-simps flip: powr-add)
  finally have q-ge: q ≥ 2 / k^2 .
  define Y where Y ≡ Yseq
  have Xseq (Suc i) = {x ∈ Xseq i. red-dense (Y i) (red-density (Xseq i) (Y i))
x}
   and Y: Y (Suc i) = Y i
    using i
    by (simp-all add: step-kind-defs next-state-def X-degree-reg-def degree-reg-def
        Y-def split: if-split-asm prod.split-asm)
  have XSnon0: card (Xseq (Suc i)) > 0
    using X-7-7 kn0 assms by simp
  have finX: finite (Xseq i) for i
    using finite-Xseq by blast
  have Xsub[simp]: Xseq (Suc i) ⊆ Xseq i
    using Xseq-Suc-subset by blast
  then have card-le: card (Xseq (Suc i)) ≤ card (Xseq i)
    by (simp add: card-mono finX)
  have 2 ≤ (real k)²
   by (metis of-nat-numeral ‹2 ≤ k› of-nat-power-le-of-nat-cancel-iff self-le-ge2-pow)
  then have 2: 2 / (real k ^ 2 + 2) ≥ 1 / k^2
    by (simp add: divide-simps)
  have q ∗ card (Xseq i \ Xseq (Suc i)) / card (Xseq (Suc i)) ≤ pseq (Suc i) −
pseq i
    using X-7-7 µ01 kn0 assms by (simp add: q-def mult-of-nat-commute)
  also have . . . ≤ 1
    by (smt (verit) pee-ge0 pee-le1)
  finally have q ∗ card (Xseq i \ Xseq (Suc i)) ≤ card (Xseq (Suc i))
    using XSnon0 by auto
  with q-ge have card (Xseq (Suc i)) ≥ (2 / k^2) ∗ card (Xseq i \ Xseq (Suc i))
    by (smt (verit, best) mult-right-mono of-nat-0-le-iff)
  then have card (Xseq (Suc i)) ∗ (1 + 2/k^2) ≥ (2/k^2) ∗ card (Xseq i)
    by (simp add: card-Diff-subset finX card-le diff-divide-distrib field-simps)
  then have card (Xseq (Suc i)) ≥ (2/(real k ^ 2 + 2)) ∗ card (Xseq i)
    using kn0 add-nonneg-nonneg[of real k^2 2]
```

by (*simp del*: *add-nonneg-nonneg add*: *divide-simps split*: *if-split-asm*)
then show *?thesis*
using *mult-right-mono* [*OF 2*, *of card* (*Xseq i*)] by *simp*
qed

6.8 Lemma 7.9

definition *Big-X-7-9* ≡ λk. ((*1* + *eps k*) *powr* (*eps k powr* (−*1*/*4*) + *1*) − *1*) / *eps k* ≤ *2* ∗ *eps k powr* (−*1*/*4*)
∧ *k*≥*2* ∧ *eps k powr* (*1*/*2*) / *k* ≥ *2* / *k*^*2*

lemma *Big-X-7-9*: ∀∞*k*. *Big-X-7-9 k*
unfolding *eps-def Big-X-7-9-def eventually-conj-iff eps-def*
by (*intro conjI*; *real-asymp*)

lemma *one-plus-powr-le*:
fixes *p*::*real*
assumes *0*≤*p p*≤*1 x*≥*0*
shows (*1*+*x*) *powr p* − *1* ≤ *x*∗*p*
proof −
define *f* **where** *f* ≡ λ*x*. *x*∗*p* − ((*1*+*x*) *powr p* − *1*)
have *0* ≤ *f 0*
by (*simp add*: *f-def*)
also have ... ≤ *f x*
proof (*intro DERIV-nonneg-imp-nondecreasing*[*of concl*: *f*] *exI conjI assms*)
fix *y*::*real*
assume *y*: *0* ≤ *y y* ≤ *x*
show (*f has-real-derivative p* − (*1*+*y*)*powr* (*p*−*1*) ∗ *p*) (*at y*)
unfolding *f-def* **using** *assms y* **by** (*intro derivative-eq-intros* | *simp*)+
show *p* − (*1*+*y*)*powr* (*p*−*1*) ∗ *p* ≥ *0*
using *y assms less-eq-real-def powr-less-one* **by** *fastforce*
qed
finally show *?thesis*
by (*simp add*: *f-def*)
qed

lemma (**in** *Book*) *X-7-9*:
assumes *i*: *i* ∈ *Step-class* {*dreg-step*} **and** *big*: *Big-X-7-9 k*
defines *hp* ≡ λ*i*. *hgt* (*pseq i*)
assumes *pseq i* ≥ *p0* **and** *hgt*: *hp* (*Suc i*) ≤ *hp i* + ε *powr* (−*1*/*4*)
shows *card* (*Xseq* (*Suc i*)) ≥ (*1* − *2* ∗ ε *powr* (*1*/*4*)) ∗ *card* (*Xseq i*)
proof −
have *k*: *k*≥*2* ε *powr* (*1*/*2*) / *k* ≥ *2* / *k*^*2*
using *big* **by** (*auto simp*: *Big-X-7-9-def*)
let *?q* = ε *powr* (−*1*/*2*) ∗ *alpha* (*hp i*)
have *k*>*0* **using** *k* **by** *auto*
have *Xsub*[*simp*]: *Xseq* (*Suc i*) ⊆ *Xseq i*
using *Xseq-Suc-subset* **by** *blast*
have *finX*: *finite* (*Xseq i*) **for** *i*

using *finite-Xseq* **by** *blast*
then have *card-le*: *card* (*Xseq* (*Suc* *i*)) ≤ *card* (*Xseq* *i*)
by (*simp add*: *card-mono finX*)
have *XSnon0*: *card* (*Xseq* (*Suc* *i*)) > *0*
using *X-7-7* ‹*0* < *k*› *i* **by** *blast*
have *card* (*Xseq* *i* \ *Xseq* (*Suc* *i*)) / *card* (*Xseq* (*Suc* *i*)) ∗ *?q* ≤ *pseq* (*Suc* *i*) − *pseq* *i*
using *X-7-7* *i* *k* *hp-def* **by** *auto*
also have . . . ≤ *2* ∗ ε *powr* (−*1*/*4*) ∗ *alpha* (*hp* *i*)
proof −
have *hgt-le*: *hp* *i* ≤ *hp* (*Suc* *i*)
using *Y-6-5-DegreeReg* ‹*0* < *k*› *i* *hp-def* **by** *blast*
have *A*: *pseq* (*Suc* *i*) ≤ *qfun* (*hp* (*Suc* *i*))
by (*simp add*: ‹*0* < *k*› *hp-def hgt-works*)
have *B*: *qfun* (*hp* *i* − *1*) ≤ *pseq* *i*
using *hgt-Least* [*of hp i* − *1 pseq i*] ‹*pseq* *i* ≥ *p0*› **by** (*force simp*: *hp-def*)
have *pseq* (*Suc* *i*) − *pseq* *i* ≤ *qfun* (*hp* (*Suc* *i*)) − *qfun* (*hp* *i* − *1*)
using *A B* **by** *auto*
also have . . . = ((*1* + ε) ^ (*Suc* (*hp* *i* − *1* + *hp* (*Suc* *i*)) − *hp* *i*) −
(*1* + ε) ^ (*hp* *i* − *1*)) / *k*
using *kn0 eps-gt0 hgt-le* ‹*pseq* *i* ≥ *p0*› *hgt-gt0* [*of k*]
by (*simp add*: *hp-def qfun-eq Suc-diff-eq-diff-pred hgt-gt0 diff-divide-distrib*)
also have . . . = *alpha* (*hp* *i*) / ε ∗ ((*1* + ε) ^ (*1* + *hp* (*Suc* *i*) − *hp* *i*) − *1*)
using *kn0 hgt-le hgt-gt0*
by (*simp add*: *hp-def alpha-eq right-diff-distrib flip*: *diff-divide-distrib power-add*)
also have . . . ≤ *2* ∗ ε *powr* (−*1*/*4*) ∗ *alpha* (*hp* *i*)
proof −
have ((*1* + ε) ^ (*1* + *hp* (*Suc* *i*) − *hp* *i*) − *1*) / ε ≤ ((*1* + ε) *powr* (ε *powr* (−*1*/*4*) + *1*) − *1*) / ε
using *hgt eps-ge0 hgt-le powr-mono-both* **by** (*force simp flip*: *powr-realpow intro*: *divide-right-mono*)
also have . . . ≤ *2* ∗ ε *powr* (−*1*/*4*)
using *big* **by** (*meson Big-X-7-9-def*)
finally have ∗: ((*1* + ε) ^ (*1* + *hp* (*Suc* *i*) − *hp* *i*) − *1*) / ε ≤ *2* ∗ ε *powr* (−*1*/*4*) **.**
show *?thesis*
using *mult-left-mono* [*OF* ∗, *of alpha* (*hp* *i*)]
by (*smt* (*verit*) *alpha-ge0 mult.commute times-divide-eq-right*)
qed
finally show *?thesis* **.**
qed
finally have *29*: *card* (*Xseq* *i* \ *Xseq* (*Suc* *i*)) / *card* (*Xseq* (*Suc* *i*)) ∗ *?q* ≤ *2* ∗ ε *powr* (−*1*/*4*) ∗ *alpha* (*hp* *i*) **.**
moreover have *alpha* (*hp* *i*) > *0*
unfolding *hp-def*
by (*smt* (*verit*, *ccfv-SIG*) *eps-gt0* ‹*0* < *k*› *alpha-ge divide-le-0-iff hgt-gt0 of-nat-0-less-iff*)
ultimately have *card* (*Xseq* *i* \ *Xseq* (*Suc* *i*)) / *card* (*Xseq* (*Suc* *i*)) ∗ ε *powr* (−*1*/*2*) ≤ *2* ∗ ε *powr* (−*1*/*4*)

using *mult-le-cancel-right* **by** *fastforce*
then have *card* (*Xseq i* \ *Xseq* (*Suc i*)) / *card* (*Xseq* (*Suc i*)) ≤ *2* ∗ ε *powr* (−*1*/*4*) ∗ ε *powr* (*1*/*2*)
using ‹*0* < *k*› *eps-gt0*
by (*force simp*: *powr-minus divide-simps mult.commute mult-less-0-iff*)
then have *card* (*Xseq i* \ *Xseq* (*Suc i*)) ≤ *2* ∗ ε *powr* (*1*/*4*) ∗ *card* (*Xseq* (*Suc i*))
using *XSnon0* **by** (*simp add*: *field-simps flip*: *powr-add*)
also have ... ≤ *2* ∗ ε *powr* (*1*/*4*) ∗ *card* (*Xseq i*)
by (*simp add*: *card-le mult-mono'*)
finally show *?thesis*
by (*simp add*: *card-Diff-subset finX card-le algebra-simps*)
qed

6.9 Lemma 7.10

definition *Big-X-7-10* ≡ λμ *l*. *Big-X-7-5* μ *l* ∧ *Big-Red-5-3* μ *l*

establishing the size requirements for 7.10

lemma *Big-X-7-10*:
assumes *0*<μ*0* μ*1*<*1*
shows ∀∞*l*. ∀μ. μ ∈ {μ*0*..μ*1*} ⟶ *Big-X-7-10* μ *l*
using *Big-X-7-10-def Big-X-7-4 Big-X-7-4-def assms* **by** *force*

lemma (**in** *Book*) *X-7-10*:
defines 𝓡 ≡ *Step-class* {*red-step*}
defines 𝓢 ≡ *Step-class* {*dboost-step*}
defines *h* ≡ λ*i*. *real* (*hgt* (*pseq i*))
defines *C* ≡ {*i*. *h i* ≥ *h* (*i*−*1*) + ε *powr* (−*1*/*4*)}
assumes *big*: *Big-X-7-10* μ *l*
shows *card* ((𝓡∪𝓢) ∩ *C*) ≤ *3* ∗ ε *powr* (*1*/*4*) ∗ *k*
proof −
define 𝓓 **where** 𝓓 ≡ *Step-class* {*dreg-step*}
define 𝓑 **where** 𝓑 ≡ *Step-class* {*bblue-step*}
have *hub*: *Big-height-upper-bound k*
and *16*: *k*≥*16*
and *ok-le-k*: *ok-fun-26 k* − *ok-fun-28 k* ≤ *k*
and *bigR53*: *Big-Red-5-3* μ *l*
using *big l-le-k* **by** (*auto simp*: *Big-X-7-5-def Big-X-7-10-def*)
have 𝓡∪𝓢 ⊆ {..<*halted-point*} \ 𝓓 \ 𝓑 **and** *BmD*: 𝓑 ⊆ {..<*halted-point*} \ 𝓓
using *halted-point-minimal'*
by (*fastforce simp*: *𝓡-def 𝓢-def 𝓓-def 𝓑-def Step-class-def*)+
then have *RS-eq*: 𝓡∪𝓢 = {..<*halted-point*} \ 𝓓 − 𝓑
using *halted-point-minimal Step-class-cases* **by** (*auto simp*: *𝓡-def 𝓢-def 𝓓-def 𝓑-def*)
obtain *26*: (∑*i*∈{..<*halted-point*} \ 𝓓. *h* (*Suc i*) − *h* (*i*−*1*)) ≤ *ok-fun-26 k*
and *28*: *ok-fun-28 k* ≤ (∑*i* ∈ 𝓑. *h*(*Suc i*) − *h*(*i*−*1*))
using *X-26-and-28 big* **unfolding** *𝓑-def 𝓓-def h-def Big-X-7-10-def* **by** *blast*

have (∑*i*∈𝓡∪𝓢. *h* (*Suc i*) − *h* (*i*−*1*)) = (∑*i*∈{..<*halted-point*} \ 𝓓. *h* (*Suc i*) − *h* (*i*−*1*)) − (∑*i* ∈ 𝓑. *h*(*Suc i*) − *h*(*i*−*1*))
  unfolding *RS-eq* **by** (*intro sum-diff BmD*) *auto*
also have ... ≤ *ok-fun-26 k* − *ok-fun-28 k*
  using *26 28* **by** *linarith*
finally have ∗: (∑*i*∈𝓡∪𝓢. *h* (*Suc i*) − *h* (*i*−*1*)) ≤ *ok-fun-26 k* − *ok-fun-28 k*
.

have [*simp*]: *finite* 𝓡 *finite* 𝓢
using *finite-components* **by** (*auto simp*: 𝓡*-def* 𝓢*-def*)
have *h-ge-0-if-S*: *h*(*Suc i*) − *h*(*i*−*1*) ≥ *0* **if** *i* ∈ 𝓢 **for** *i*
proof −
  have ∗: *hgt* (*pseq i*) ≤ *hgt* (*pseq* (*Suc i*))
    using *bigR53 Y-6-5-dbooSt that* **unfolding** 𝓢*-def* **by** *blast*
  obtain *i*−*1* ∈ 𝓓 *i*>*0*
    using *that* ‹*i*∈𝓢› *dreg-before-step1*[*of i*] *dreg-before-gt0*[*of i*]
    by (*force simp*: 𝓢*-def* 𝓓*-def Step-class-insert-NO-MATCH*)
  then have *hgt* (*pseq* (*i*−*1*)) ≤ *hgt* (*pseq i*)
    using *that kn0* **by** (*metis Suc-diff-1 Y-6-5-DegreeReg* 𝓓*-def*)
  with ∗ **show** *0* ≤ *h*(*Suc i*) − *h*(*i*−*1*)
    using *kn0* **unfolding** *h-def* **by** *linarith*
qed

have *card* ((𝓡∪𝓢) ∩ *C*) ∗ ε *powr* (−*1*/*4*) + *real* (*card* 𝓡) ∗ (−*2*)
    = (∑*i* ∈ 𝓡∪𝓢. *if i*∈*C then* ε *powr* (−*1*/*4*) *else 0*) + (∑*i* ∈ 𝓡∪𝓢. *if i*∈𝓡 *then* −*2 else 0*)
  by (*simp add*: *Int-commute Int-left-commute flip*: *sum.inter-restrict*)
also have ... = (∑*i* ∈ 𝓡∪𝓢. (*if i*∈*C then* ε *powr* (−*1*/*4*) *else 0*) + (*if i*∈𝓡 *then* −*2 else 0*))
  by (*simp add*: *sum.distrib*)
also have ... ≤ (∑*i* ∈ 𝓡∪𝓢. *h*(*Suc i*) − *h*(*i*−*1*))
proof (*rule sum-mono*)
  fix *i* :: *nat*
  assume *i*: *i* ∈ 𝓡∪𝓢
  with *i dreg-before-step1 dreg-before-gt0* **have** *D*: *i*−*1* ∈ 𝓓 *i*>*0*
    by (*force simp*: 𝓢*-def* 𝓡*-def* 𝓓*-def dreg-before-step Step-class-def*)+
  then have ∗: *hgt* (*pseq* (*i*−*1*)) ≤ *hgt* (*pseq i*)
    by (*metis Suc-diff-1 Y-6-5-DegreeReg* 𝓓*-def*)
  show (*if i*∈*C then* ε *powr* (−*1*/*4*) *else 0*) + (*if i*∈𝓡 *then* − *2 else 0*) ≤ *h* (*Suc i*) − *h* (*i*−*1*)
  proof (*cases i*∈𝓡)
    case *True*
    then have *h i* − *2* ≤ *h* (*Suc i*)
      using *Y-6-5-Red*[*of i*] *16* **by** (*force simp*: *algebra-simps* 𝓡*-def h-def*)
    with ∗ *True* **show** *?thesis*
      by (*simp add*: *h-def C-def*)
  next
    case *False*
    with *i* **have** *i*∈𝓢 **by** *blast*

show *?thesis*
proof (*cases* $i{\in}C$)
case *True*
then have $h\ (i - Suc\ 0) + \varepsilon\ powr\ (-1/4) \le h\ i$
by (*simp add*: *C-def*)
then show *?thesis*
using $*$ *i* $\langle i{\notin}\mathcal{R}\rangle$ *kn0 bigR53 Y-6-5-dbooSt* **by** (*force simp*: *h-def* $\mathcal{S}$*-def*)
qed (*use* $\langle i{\notin}\mathcal{R}\rangle$ $\langle i{\in}\mathcal{S}\rangle$ *h-ge-0-if-S* **in** *auto*)
qed
qed
also have $\ldots \le k$
using $*$ *ok-le-k*
by *linarith*
finally have $card\ ((\mathcal{R}{\cup}\mathcal{S}) \cap C) * \varepsilon\ powr\ (-1/4) - 2 * card\ \mathcal{R} \le k$
by *linarith*
moreover have $card\ \mathcal{R} \le k$
by (*metis* $\mathcal{R}$*-def nless-le red-step-limit*)
ultimately have $card\ ((\mathcal{R}{\cup}\mathcal{S}) \cap C) * \varepsilon\ powr\ (-1/4) \le 3 * k$
by *linarith*
with *eps-gt0* **show** *?thesis*
by (*simp add*: *powr-minus divide-simps mult.commute split*: *if-split-asm*)
qed

6.10 Lemma 7.11

definition *Big-X-7-11-inequalities* $\equiv \lambda k.$
$eps\ k * eps\ k\ powr\ (-1/4) \le (1 + eps\ k)\ \hat{}\ (2 * nat\ \lfloor eps\ k\ powr\ (-1/4)\rfloor) - 1$
$\wedge\ k \ge 2 * eps\ k\ powr\ (-1/2) * k\ powr\ (3/4)$
$\wedge\ ((1 + eps\ k) * (1 + eps\ k)\ powr\ (2 * eps\ k\ powr\ (-1/4))) \le 2$
$\wedge\ (1 + eps\ k)\ \hat{}\ (nat\ \lfloor 2 * eps\ k\ powr\ (-1/4)\rfloor + nat\ \lfloor 2 * eps\ k\ powr\ (-1/2)\rfloor - 1) \le 2$

definition *Big-X-7-11* $\equiv$
$\lambda\mu\ l.$ *Big-X-7-5* $\mu\ l \wedge$ *Big-Red-5-3* $\mu\ l \wedge$ *Big-Y-6-5-Bblue* l
$\wedge\ (\forall k.\ l{\le}k \longrightarrow$ *Big-X-7-11-inequalities* $k)$

establishing the size requirements for 7.11

lemma *Big-X-7-11*:
assumes $0{<}\mu 0$ $\mu 1{<}1$
shows $\forall^\infty l.\ \forall\mu.\ \mu \in \{\mu 0..\mu 1\} \longrightarrow$ *Big-X-7-11* $\mu\ l$
using *assms Big-Red-5-3 Big-X-7-5 Big-Y-6-5-Bblue*
unfolding *Big-X-7-11-def Big-X-7-11-inequalities-def eventually-conj-iff all-imp-conj-distrib eps-def*
apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)
apply (*intro conjI strip eventually-all-geI0 eventually-all-ge-at-top*; *real-asymp*)
done

lemma (**in** *Book*) *X-7-11*:
defines $\mathcal{R} \equiv$ *Step-class* {*red-step*}

```
  defines 𝒮 ≡ Step-class {dboost-step}
  defines C ≡ {i. pseq i ≥ pseq (i−1) + ε powr (−1/4) ∗ alpha 1 ∧ pseq (i−1)
≤ p0}
  assumes big: Big-X-7-11 μ l
  shows card ((ℛ∪𝒮) ∩ C) ≤ 4 ∗ ε powr (1/4) ∗ k
proof −
  define qstar where qstar ≡ p0 + ε powr (−1/4) ∗ alpha 1
  define pstar where pstar ≡ λi. min (pseq i) qstar
  define 𝒟 where 𝒟 ≡ Step-class {dreg-step}
  define ℬ where ℬ ≡ Step-class {bblue-step}
  have big-x75: Big-X-7-5 μ l
    and 711: ε ∗ ε powr (−1/4) ≤ (1 + ε) ^ (2 ∗ nat ⌊ε powr (−1/4)⌋) − 1
    and big34: k ≥ 2 ∗ ε powr (−1/2) ∗ k powr (3/4)
    and le2: ((1 + ε) ∗ (1 + ε) powr (2 ∗ ε powr (−1/4))) ≤ 2
             (1 + ε) ^ (nat ⌊2 ∗ ε powr (−1/4)⌋ + nat ⌊2 ∗ ε powr (−1/2)⌋ − 1)
≤ 2
    and bigY65B: Big-Y-6-5-Bblue l
    and R53: ⋀i. i ∈ 𝒮 ⟹ pseq (Suc i) ≥ pseq i
    using big l-le-k
    by (auto simp: Red-5-3 Big-X-7-11-def Big-X-7-11-inequalities-def 𝒮-def)
  then have Y-6-5-B: ⋀i. i ∈ ℬ ⟹ hgt (pseq (Suc i)) ≥ hgt (pseq (i−1)) − 2
∗ ε powr (−1/2)
    using bigY65B Y-6-5-Bblue unfolding ℬ-def by blast
  have big41: Big-Blue-4-1 μ l
    and hub: Big-height-upper-bound k
    and 16: k≥16
    and ok-le-k: ok-fun-26 k − ok-fun-28 k ≤ k
    using big-x75 l-le-k by (auto simp: Big-X-7-5-def)
  have oddset: {..<halted-point} \ 𝒟 = {i ∈ {..<halted-point}. odd i}
     using step-odd step-even not-halted-even-dreg halted-point-minimal by (auto
simp: 𝒟-def)
  have [simp]: finite ℛ finite ℬ finite 𝒮
    using finite-components by (auto simp: ℛ-def ℬ-def 𝒮-def)
  have [simp]: ℛ ∩ 𝒮 = {} and [simp]: (ℛ ∪ 𝒮) ∩ ℬ = {}
    by (simp-all add: ℛ-def 𝒮-def ℬ-def Step-class-def disjoint-iff)

  have hgt-qstar-le: hgt qstar ≤ 2 ∗ ε powr (−1/4)
  proof (intro real-hgt-Least)
    show 0 < 2 ∗ nat ⌊ε powr (−1/4)⌋
      using kn0 eps-gt0 by (simp add: eps-le1 powr-le1 powr-minus-divide)
    show qstar ≤ qfun (2 ∗ nat ⌊ε powr (−1/4)⌋)
      using kn0 711
      by (simp add: qstar-def alpha-def qfun-eq divide-right-mono mult.commute)
  qed auto
  then have ((1 + ε) ∗ (1 + ε) ^ hgt qstar) ≤ ((1 + ε) ∗ (1 + ε) powr (2 ∗ ε
powr (−1/4)))
    by (smt (verit) eps-ge0 mult-left-mono powr-mono powr-realpow)
  also have ((1 + ε) ∗ (1 + ε) powr (2 ∗ ε powr (−1/4))) ≤ 2
    using le2 by simp
```

```
  finally have (1 + ε) * (1 + ε) ^ hgt qstar ≤ 2 .
  moreover have card ℛ ≤ k
    by (simp add: ℛ-def less-imp-le red-step-limit)
  ultimately have §: ((1 + ε) * (1 + ε) ^ hgt qstar) * card ℛ ≤ 2 * real k
    by (intro mult-mono) auto
  have − 2 * alpha 1 * k ≤ − alpha (hgt qstar + 2) * card ℛ
    using mult-right-mono-neg [OF §, of − ε] eps-ge0
    by (simp add: alpha-eq divide-simps mult-ac)
  also have ... ≤ (∑i∈ℛ. pstar (Suc i) − pstar i)
  proof −
    { fix i
      assume i ∈ ℛ
      have − alpha (hgt qstar + 2) ≤ pstar (Suc i) − pstar i
      proof (cases hgt (pseq i) > hgt qstar + 2)
        case True
        then have hgt (pseq (Suc i)) > hgt qstar
          using Y-6-5-Red 16 ‹i ∈ ℛ› by (force simp: ℛ-def)
        then have pstar (Suc i) = pstar i
          using True hgt-mono' pstar-def by fastforce
        then show ?thesis
          by (simp add: alpha-ge0)
      next
        case False
        with ‹i ∈ ℛ› show ?thesis
          unfolding pstar-def ℛ-def
             by (smt (verit, del-insts) Y-6-4-Red alpha-ge0 alpha-mono hgt-gt0
linorder-not-less)
      qed
    }
    then show ?thesis
      by (smt (verit, ccfv-SIG) mult-of-nat-commute sum-constant sum-mono)
  qed
  finally have − 2 * alpha 1 * k ≤ (∑i∈ℛ. pstar (Suc i) − pstar i) .
  moreover have 0 ≤ (∑i∈𝒮. pstar (Suc i) − pstar i)
    using R53 by (intro sum-nonneg) (force simp: pstar-def)
  ultimately have RS-half: − 2 * alpha 1 * k ≤ (∑i∈ℛ∪𝒮. pstar (Suc i) −
pstar i)
    by (simp add: sum.union-disjoint)

  let ?e12 = ε powr (−1/2)
  define h' where h' ≡ hgt qstar + nat ⌊2 * ?e12⌋
  have − alpha 1 * k ≤ −2 * ?e12 * alpha 1 * k powr (3/4)
    using mult-right-mono-neg [OF big34, of − alpha 1] alpha-ge0 [of 1]
    by (simp add: mult-ac)
  also have ... ≤ −?e12 * alpha (h') * card ℬ
  proof −
    have card ℬ ≤ l powr (3/4)
      using big41 bblue-step-limit by (simp add: ℬ-def)
    also have ... ≤ k powr (3/4)
```

```
      by (simp add: powr-mono2 l-le-k)
    finally have 1: card ℬ ≤ k powr (3/4) .
    have alpha (h′) ≤ alpha (nat ⌊2 ∗ ε powr (−1/4)⌋ + nat ⌊2 ∗ ?e12⌋)
    proof (rule alpha-mono)
      show h′ ≤ nat ⌊2 ∗ ε powr (−1/4)⌋ + nat ⌊2 ∗ ?e12⌋
        using h′-def hgt-qstar-le le-nat-floor by auto
    qed (simp add: hgt-gt0 h′-def)
    also have . . . ≤ 2 ∗ alpha 1
    proof −
      have ∗: (1 + ε) ^ (nat ⌊2 ∗ ε powr (−1/4)⌋ + nat ⌊2 ∗ ?e12⌋ − 1) ≤ 2
        using le2 by simp
      have 1 ≤ 2 ∗ ε powr (−1/4)
       by (smt (verit) hgt-qstar-le Suc-leI divide-minus-left hgt-gt0 numeral-nat(7)
real-of-nat-ge-one-iff)
      then show ?thesis
        using mult-right-mono [OF ∗, of ε] eps-ge0
        by (simp add: alpha-eq hgt-gt0 divide-right-mono mult.commute)
    qed
    finally have 2: 2 ∗ alpha 1 ≥ alpha (h′) .
    show ?thesis
     using mult-right-mono-neg [OF mult-mono [OF 1 2], of −?e12] alpha-ge0
by (simp add: mult-ac)
  qed
  also have . . . ≤ (∑i∈ℬ. pstar (Suc i) − pstar (i−1))
  proof −
    { fix i
      assume i ∈ ℬ
      have −?e12 ∗ alpha (h′) ≤ pstar (Suc i) − pstar (i−1)
      proof (cases hgt (pseq (i−1)) > hgt qstar + 2 ∗ ?e12)
        case True
        then have hgt (pseq (Suc i)) > hgt qstar
          using Y-6-5-B ‹i ∈ ℬ› by (force simp: ℛ-def)
        then have pstar (i−1) = pstar(Suc i)
          unfolding pstar-def
          by (smt (verit) True hgt-mono′ of-nat-less-iff powr-non-neg)
        then show ?thesis
          by (simp add: alpha-ge0)
      next
        case False
        then have hgt (pseq (i−1)) ≤ h′
          by (simp add: h′-def) linarith
        then have †: alpha (hgt (pseq (i−1))) ≤ alpha h′
          by (intro alpha-mono hgt-gt0)
        have pseq (Suc i) ≥ pseq (i−1) − ?e12 ∗ alpha (hgt (pseq (i−1)))
          using Y-6-4-Bblue ‹i ∈ ℬ› unfolding ℬ-def by blast
        with mult-left-mono [OF †, of ?e12] show ?thesis
          unfolding pstar-def
          by (smt (verit) alpha-ge0 mult-minus-left powr-non-neg mult-le-0-iff)
      qed
```

}
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *mult-of-nat-commute sum-constant sum-mono*)
qed
finally have *B*: − *alpha 1* ∗ *k* ≤ (∑*i*∈ℬ. *pstar* (*Suc i*) − *pstar* (*i*−*1*)) **.**

have *ε powr* (−*1*/*4*) ∗ *alpha 1* ∗ *card* ((ℛ∪𝒮) ∩ *C*) ≤ (∑*i*∈ℛ∪𝒮. *if i* ∈ *C then ε powr* (−*1*/*4*) ∗ *alpha 1 else 0*)
by (*simp add*: *flip*: *sum.inter-restrict*)
also have (∑*i*∈ℛ∪𝒮. *if i* ∈ *C then ε powr* (−*1*/*4*) ∗ *alpha 1 else 0*) ≤ (∑*i*∈ℛ∪𝒮. *pstar i* − *pstar* (*i*−*1*))
proof (*intro sum-mono*)
fix *i*
assume *i*: *i* ∈ ℛ ∪ 𝒮
then obtain *i*−*1* ∈ 𝒟 *i*>*0*
unfolding *ℛ-def 𝒮-def 𝒟-def* **by** (*metis dreg-before-step1 dreg-before-gt0 Step-class-insert Un-iff*)
then have *pseq* (*i*−*1*) ≤ *pseq i*
by (*metis Suc-pred′ Y-6-4-DegreeReg 𝒟-def*)
then have *pstar* (*i*−*1*) ≤ *pstar i*
by (*fastforce simp*: *pstar-def*)
then show (*if i* ∈ *C then ε powr* (−*1*/*4*) ∗ *alpha 1 else 0*) ≤ *pstar i* − *pstar* (*i*−*1*)
using *C-def pstar-def qstar-def* **by** *auto*
qed
finally have §: *ε powr* (−*1*/*4*) ∗ *alpha 1* ∗ *card* ((ℛ∪𝒮) ∩ *C*) ≤ (∑*i*∈ℛ∪𝒮. *pstar i* − *pstar* (*i*−*1*)) **.**

have *psplit*: *pstar* (*Suc i*) − *pstar* (*i*−*1*) = (*pstar* (*Suc i*) − *pstar i*) + (*pstar i* − *pstar* (*i*−*1*)) **for** *i*
by *simp*
have *RS*: *ε powr* (−*1*/*4*) ∗ *alpha 1* ∗ *card* ((ℛ∪𝒮) ∩ *C*) + (− *2* ∗ *alpha 1* ∗ *k*) ≤ (∑*i*∈ℛ∪𝒮. *pstar* (*Suc i*) − *pstar* (*i*−*1*))
unfolding *psplit sum.distrib* **using** *RS-half* § **by** *linarith*

have *k16*: *k powr* (*1*/*16*) ≤ *k powr 1*
using *kn0* **by** (*intro powr-mono*) *auto*

have *meq*: {..<*halted-point*} \ 𝒟 = (ℛ∪𝒮) ∪ ℬ
using *Step-class-cases halted-point-minimal′* **by**(*fastforce simp*: *ℛ-def 𝒮-def 𝒟-def ℬ-def Step-class-def*)

have (*ε powr* (−*1*/*4*) ∗ *alpha 1* ∗ *card* ((ℛ∪𝒮) ∩ *C*) + (− *2* ∗ *alpha 1* ∗ *k*))
\+ (− *alpha 1* ∗ *k*)
≤ (∑*i* ∈ ℛ∪𝒮. *pstar*(*Suc i*) − *pstar*(*i*−*1*)) + (∑*i*∈ℬ. *pstar*(*Suc i*) − *pstar*(*i*−*1*))
using *RS B* **by** *linarith*
also have ... = (∑*i* ∈ {..<*halted-point*} \ 𝒟. *pstar*(*Suc i*) − *pstar*(*i*−*1*))
by (*simp add*: *meq sum.union-disjoint*)

also have ... $\leq$ *pstar halted-point* $-$ *pstar 0*
proof (*cases even halted-point*)
case *False*
have *pseq* (*halted-point* $-$ *Suc 0*) $\leq$ *pseq halted-point*
using *Y-6-4-DegreeReg* [*of halted-point*$-$*1*] *False not-halted-even-dreg odd-pos*

by (*auto simp*: *halted-point-minimal*)
then have *pstar*(*halted-point* $-$ *Suc 0*) $\leq$ *pstar halted-point*
by (*simp add*: *pstar-def*)
with *False* **show** *?thesis*
by (*simp add*: *oddset sum-odds-odd*)
qed (*simp add*: *oddset sum-odds-even*)
also have ... = ($\sum$ *i* < *halted-point*. *pstar*(*Suc i*) $-$ *pstar i*)
by (*simp add*: *sum-lessThan-telescope*)
also have ... = *pstar halted-point* $-$ *pstar 0*
by (*simp add*: *sum-lessThan-telescope*)
also have ... $\leq$ *alpha 1* $*$ ε *powr* ($-$*1/4*)
using *alpha-ge0* **by** (*simp add*: *mult.commute pee-eq-p0 pstar-def qstar-def*)
also have ... $\leq$ *alpha 1* $*$ *k*
using *alpha-ge0 k16* **by** (*intro powr-mono mult-left-mono*) (*auto simp*: *eps-def powr-powr*)
finally have ε *powr* ($-$*1/4*) $*$ *card* (($\mathcal{R} \cup \mathcal{S}$) $\cap$ *C*) $*$ *alpha 1* $\leq$ *4* $*$ *k* $*$ *alpha 1*
by (*simp add*: *mult-ac*)
then have ε *powr* ($-$*1/4*) $*$ *real* (*card* (($\mathcal{R} \cup \mathcal{S}$) $\cap$ *C*)) $\leq$ *4* $*$ *k*
using *kn0* **by** (*simp add*: *divide-simps alpha-eq eps-gt0*)
then show *?thesis*
using *alpha-ge0*[*of 1*] *kn0 eps-gt0* **by** (*simp add*: *powr-minus divide-simps mult-ac split*: *if-split-asm*)
qed

6.11 Lemma 7.12

definition *Big-X-7-12* $\equiv$
$\lambda\mu$ *l*. *Big-X-7-11* μ *l* $\wedge$ *Big-X-7-10* μ *l* $\wedge$ ($\forall$*k*. *l*$\leq$*k* $\longrightarrow$ *Big-X-7-9 k*)

establishing the size requirements for 7.12

lemma *Big-X-7-12*:
assumes *0*<μ*0* μ*1*<*1*
shows $\forall^\infty$*l*. $\forall\mu$. $\mu \in$ {μ*0*..μ*1*} $\longrightarrow$ *Big-X-7-12* μ *l*
using *assms Big-X-7-11 Big-X-7-10 Big-X-7-9*
unfolding *Big-X-7-12-def eventually-conj-iff*
apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib eventually-frequently-const-simps*)
using *eventually-all-ge-at-top* **by** *blast*

lemma (**in** *Book*) *X-7-12*:
defines $\mathcal{R} \equiv$ *Step-class* {*red-step*}
defines $\mathcal{S} \equiv$ *Step-class* {*dboost-step*}
defines *C* $\equiv$ {*i*. *card* (*Xseq i*) < (*1* $-$ *2* $*$ ε *powr* (*1/4*)) $*$ *card* (*Xseq* (*i*$-$*1*))}
assumes *big*: *Big-X-7-12* μ *l*
shows *card* (($\mathcal{R}\cup\mathcal{S}$) $\cap$ *C*) $\leq$ *7* $*$ ε *powr* (*1/4*) $*$ *k*

proof –
define $\mathcal{D}$ **where** $\mathcal{D} \equiv$ *Step-class* {*dreg-step*}
have *big-711*: *Big-X-7-11* μ *l* **and** *big-710*: *Big-X-7-10* μ *l*
using *big* **by** (*auto simp*: *Big-X-7-12-def*)
have [*simp*]: *finite* $\mathcal{R}$ *finite* $\mathcal{S}$
using *finite-components* **by** (*auto simp*: $\mathcal{R}$*-def* $\mathcal{S}$*-def*)
— now the conditions for Lemmas 7.10 and 7.11
define *C10* **where** $C10 \equiv \{i.\ hgt\ (pseq\ i) \ge hgt\ (pseq\ (i-1)) + \varepsilon\ powr\ (-1/4)\}$
define *C11* **where** $C11 \equiv \{i.\ pseq\ i \ge pseq\ (i-1) + \varepsilon\ powr\ (-1/4) * alpha\ 1$
$\wedge\ pseq\ (i-1) \le p0\}$
have $(\mathcal{R}\cup\mathcal{S}) \cap C \cap \{i.\ pseq\ (i-1) \le p0\} \subseteq (\mathcal{R}\cup\mathcal{S}) \cap C11$
proof
fix *i*
assume *i*: $i \in (\mathcal{R}\cup\mathcal{S}) \cap C \cap \{i.\ pseq\ (i-1) \le p0\}$
then have *iRS*: $i \in \mathcal{R} \cup \mathcal{S}$ **and** *iC*: $i \in C$
by *auto*
then obtain *i1*: $i-1 \in \mathcal{D}$ $i>0$
unfolding $\mathcal{R}$*-def* $\mathcal{S}$*-def* $\mathcal{D}$*-def* **by** (*metis Step-class-insert Un-iff dreg-before-step1*
dreg-before-gt0)
then have *77*: $card\ (Xseq\ (i-1) \setminus Xseq\ i)\ /\ card\ (Xseq\ i) * (\varepsilon\ powr\ (-1/2)$
$*\ alpha\ (hgt\ (pseq\ (i-1))))$
$\le pseq\ i\ -\ pseq\ (i-1)$
by (*metis Suc-diff-1 X-7-7* $\mathcal{D}$*-def*)
have *card-Xm1*: $card\ (Xseq\ (i-1)) = card\ (Xseq\ i) + card\ (Xseq\ (i-1) \setminus Xseq$
$i)$
by (*metis Xseq-antimono add-diff-inverse-nat card-Diff-subset card-mono*
diff-le-self
finite-Xseq linorder-not-less)
have $card\ (Xseq\ i) > 0$
by (*metis Step-class-insert card-Xseq-pos* $\mathcal{R}$*-def* $\mathcal{S}$*-def iRS*)
have $card\ (Xseq\ (i-1)) > 0$
using *C-def iC less-irrefl* **by** *fastforce*
moreover have $2 * (card\ (Xseq\ (i-1)) * \varepsilon\ powr\ (1/4)) < card\ (Xseq\ (i-1)$
$\setminus Xseq\ i)$
using *iC card-Xm1* **by** (*simp add*: *algebra-simps C-def*)
moreover have $card\ (Xseq\ i) \le 2 * card\ (Xseq\ (i-1))$
using *card-Xm1* **by** *linarith*
ultimately have $\varepsilon\ powr\ (1/4) \le card\ (Xseq\ (i-1) \setminus Xseq\ i)\ /\ card\ (Xseq$
$(i-1))$
by (*simp add*: *divide-simps mult.commute*)
moreover have $real\ (card\ (Xseq\ i)) \le card\ (Xseq\ (i-1))$
using *card-Xm1* **by** *linarith*
ultimately have *1*: $\varepsilon\ powr\ (1/4) \le card\ (Xseq\ (i-1) \setminus Xseq\ i)\ /\ card\ (Xseq$
$i)$
by (*smt* (*verit*) ‹$0 < card\ (Xseq\ i)$› *frac-le of-nat-0-le-iff of-nat-0-less-iff*)
have $\varepsilon\ powr\ (-1/4) * alpha\ 1$
$\le card\ (Xseq\ (i-1) \setminus Xseq\ i)\ /\ card\ (Xseq\ i) * (\varepsilon\ powr\ (-1/2) * alpha\ 1)$
using *alpha-ge0 mult-right-mono* [*OF 1*, *of* ε *powr* $(-1/2) * alpha\ 1$]
by (*simp add*: *mult-ac flip*: *powr-add*)

```
    also have ... ≤ card (Xseq (i−1) \ Xseq i) / card (Xseq i) ∗ (ε powr (−1/2)
∗ alpha (hgt (pseq (i−1))))
      by (intro mult-left-mono alpha-mono) (auto simp: Suc-leI hgt-gt0)
    also have ... ≤ pseq i − pseq (i−1)
      using 77 by simp
    finally have ε powr (−1/4) ∗ alpha 1 ≤ pseq i − pseq (i−1) .
    with i show i ∈ (R ∪ S) ∩ C11
      by (simp add: C11-def)
  qed
  then have real (card ((R∪S) ∩ C ∩ {i. pseq (i−1) ≤ p0})) ≤ real (card ((R∪S)
∩ C11))
    by (simp add: card-mono)
  also have ... ≤ 4 ∗ ε powr (1/4) ∗ k
   using X-7-11 big-711 by (simp add: R-def S-def C11-def Step-class-insert-NO-MATCH)
  finally have card ((R∪S) ∩ C ∩ {i. pseq (i−1) ≤ p0}) ≤ 4 ∗ ε powr (1/4) ∗
k .
  moreover
  have card ((R∪S) ∩ C \ {i. pseq (i−1) ≤ p0}) ≤ 3 ∗ ε powr (1/4) ∗ k
  proof −
    have Big-X-7-9 k
      using Big-X-7-12-def big l-le-k by presburger
    then have X79: card (Xseq (Suc i)) ≥ (1 − 2 ∗ ε powr (1/4)) ∗ card (Xseq
i)
      if i ∈ Step-class {dreg-step} and pseq i ≥ p0
          and hgt (pseq (Suc i)) ≤ hgt (pseq i) + ε powr (−1/4) for i
      using X-7-9 that by blast
    have (R∪S) ∩ C \ {i. pseq (i−1) ≤ p0} ⊆ (R∪S) ∩ C10
      unfolding C10-def C-def
    proof clarify
      fix i
      assume i ∈ R ∪ S
        and §: card (Xseq i) < (1 − 2 ∗ ε powr (1/4)) ∗ card (Xseq (i−1)) ¬ pseq
(i−1) ≤ p0
      then obtain i−1 ∈ D i>0
        unfolding D-def R-def S-def
       by (metis dreg-before-step1 dreg-before-gt0 Step-class-Un Un-iff insert-is-Un)
      with X79 § show hgt (pseq (i − 1)) + ε powr (−1/4) ≤ hgt (pseq i)
        by (force simp: D-def)
    qed
    then have card ((R∪S) ∩ C \ {i. pseq (i−1) ≤ p0}) ≤ real (card ((R∪S) ∩
C10))
      by (simp add: card-mono)
    also have card ((R∪S) ∩ C10) ≤ 3 ∗ ε powr (1/4) ∗ k
      unfolding R-def S-def C10-def by (intro X-7-10 assms big-710)
    finally show ?thesis .
  qed
  moreover
  have card ((R∪S) ∩ C)
      = real (card ((R∪S) ∩ C ∩ {i. pseq (i−1) ≤ p0})) + real (card ((R∪S) ∩
```

$C \setminus \{i.\ pseq\ (i-1) \le p0\})$)
  by (*metis card-Int-Diff of-nat-add ‹finite $\mathcal{R}$› ‹finite $\mathcal{S}$› finite-Int infinite-Un*)
 ultimately show *?thesis*
  by *linarith*
qed

6.12 Lemma 7.6

definition *Big-X-7-6* $\equiv$
 $\lambda\mu\ l.\ Big\text{-}Blue\text{-}4\text{-}1\ \mu\ l \wedge Big\text{-}X\text{-}7\text{-}12\ \mu\ l \wedge (\forall k.\ k{\ge}l \longrightarrow Big\text{-}X\text{-}7\text{-}8\ k \wedge 1 - 2 * eps\ k\ powr\ (1/4) > 0)$

lemma *Big-X-7-6*:
 assumes $0{<}\mu 0\ \ \mu 1{<}1$
 shows $\forall^\infty l.\ \forall \mu.\ \mu \in \{\mu 0..\mu 1\} \longrightarrow Big\text{-}X\text{-}7\text{-}6\ \mu\ l$
 using *assms Big-Blue-4-1 Big-X-7-8 Big-X-7-12*
 unfolding *Big-X-7-6-def eps-def*
 apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib eventually-all-ge-at-top*)

 apply (*intro conjI strip eventually-all-geI0 eventually-all-ge-at-top*; *real-asymp*)
 done

definition *ok-fun-76* $\equiv$
 $\lambda k.\ ((1 + 2 * real\ k) * ln\ (1 - 2 * eps\ k\ powr\ (1/4))$
   $- (k\ powr\ (3/4) + 7 * eps\ k\ powr\ (1/4) * k + 1) * (2 * ln\ k))\ /\ ln\ 2$

lemma *ok-fun-76*: $ok\text{-}fun\text{-}76 \in o(real)$
 unfolding *eps-def ok-fun-76-def* **by** *real-asymp*

lemma (**in** *Book*) *X-7-6*:
 assumes *big*: $Big\text{-}X\text{-}7\text{-}6\ \mu\ l$
 defines $\mathcal{D} \equiv Step\text{-}class\ \{dreg\text{-}step\}$
 shows $(\prod i{\in}\mathcal{D}.\ card(Xseq(Suc\ i))\ /\ card\ (Xseq\ i)) \ge 2\ powr\ ok\text{-}fun\text{-}76\ k$
proof –
 define $\mathcal{R}$ **where** $\mathcal{R} \equiv Step\text{-}class\ \{red\text{-}step\}$
 define $\mathcal{B}$ **where** $\mathcal{B} \equiv Step\text{-}class\ \{bblue\text{-}step\}$
 define $\mathcal{S}$ **where** $\mathcal{S} \equiv Step\text{-}class\ \{dboost\text{-}step\}$
 define C **where** $C \equiv \{i.\ card\ (Xseq\ i) < (1 - 2 * \varepsilon\ powr\ (1/4)) * card\ (Xseq\ (i-1))\}$
 define C' **where** $C' \equiv Suc\ -`\ C$
 have *big41*: $Big\text{-}Blue\text{-}4\text{-}1\ \mu\ l$
  and *712*: $card\ ((\mathcal{R}{\cup}\mathcal{S}) \cap C) \le 7 * \varepsilon\ powr\ (1/4) * k$
  using *big X-7-12 l-le-k* **by** (*auto simp*: *Big-X-7-6-def $\mathcal{R}$-def $\mathcal{S}$-def C-def*)

 have [*simp*]: *finite $\mathcal{D}$ finite $\mathcal{R}$ finite $\mathcal{B}$ finite $\mathcal{S}$*
  using *finite-components* **by** (*auto simp*: *$\mathcal{D}$-def $\mathcal{R}$-def $\mathcal{B}$-def $\mathcal{S}$-def*)
 have $card\ \mathcal{R} < k$
  using *$\mathcal{R}$-def assms red-step-limit* **by** *blast+*
 have $card\ \mathcal{B} \le l\ powr\ (3/4)$

```
    using big41 bblue-step-limit by (auto simp: B-def)
  then have card (B ∩ C) ≤ l powr (3/4)
    using card-mono [OF - Int-lower1] by (smt (verit) ‹finite B› of-nat-mono)
  also have ... ≤ k powr (3/4)
    by (simp add: l-le-k powr-mono2)
  finally have Bk-34: card (B ∩ C) ≤ k powr (3/4) .

  have less-l: card B + card S < l
    using bblue-dboost-step-limit big41 by (auto simp: B-def S-def)
  have [simp]: (B ∪ (R ∪ S)) ∩ {halted-point} = {} R ∩ S = {} B ∩ (R ∪ S) =
{}
            halted-point ∉ B halted-point ∉ R halted-point ∉ S
            B ∩ C ∩ (R ∩ C ∪ S ∩ C) = {} for C
   using halted-point-minimal' by (force simp: B-def R-def S-def Step-class-def)+

  have Big-X-7-8 k and one-minus-gt0: 1 − 2 ∗ ε powr (1/4) > 0
    using big l-le-k by (auto simp: Big-X-7-6-def)
  then have X78: card (Xseq (Suc i)) ≥ card (Xseq i) / k^2 if i ∈ D for i
    using X-7-8 that by (force simp: D-def)

  let ?DC = λk. k powr (3/4) + 7 ∗ eps k powr (1/4) ∗ k + 1
  have dc-pos: ?DC k > 0 for k
    by (smt (verit) of-nat-less-0-iff powr-ge-zero zero-le-mult-iff)
  have X-pos: card (Xseq i) > 0 if i ∈ D for i
  proof −
    have card (Xseq (Suc i)) > 0
      using that X-7-7 kn0 unfolding D-def by blast
    then show ?thesis
      by (metis Xseq-Suc-subset card-mono finite-Xseq gr0I leD)
  qed
  have ok-fun-76 k ≤ log 2 ((1 / (real k)^2) powr ?DC k ∗ (1 − 2 ∗ ε powr (1/4))
^ (k + l + 1))
    unfolding ok-fun-76-def log-def
    using kn0 l-le-k one-minus-gt0
     by (simp add: ln-mult ln-div ln-realpow divide-right-mono mult-le-cancel-right
flip: power-Suc mult.assoc)
  then have 2 powr ok-fun-76 k ≤ (1 / (real k)^2) powr ?DC k ∗ (1 − 2 ∗ ε powr
(1/4)) ^ (k + l + 1)
    using powr-eq-iff kn0 one-minus-gt0 by (simp add: le-log-iff)
  also have ... ≤ (1 / (real k)^2) powr card (D ∩ C') ∗ (1 − 2 ∗ ε powr (1/4))
^ card (D\C')
  proof (intro mult-mono powr-mono')
    have Suc i ∈ R if i ∈ D Suc i ≠ halted-point Suc i ∉ B Suc i ∉ S for i
    proof −
      have Suc i ∉ D
        by (metis D-def ‹i ∈ D› even-Suc step-even)
      moreover
      have stepper-kind i ≠ halted
        using D-def ‹i ∈ D› Step-class-def by force
```

ultimately show *Suc i* $\in \mathcal{R}$
using *that halted-point-minimal′ halted-point-minimal Step-class-cases Suc-lessI*
$\mathcal{B}$*-def* $\mathcal{D}$*-def* $\mathcal{R}$*-def* $\mathcal{S}$*-def* **by** *blast*
qed
then have *Suc* ‘ $\mathcal{D} \subseteq \mathcal{B} \cup (\mathcal{R} \cup \mathcal{S}) \cup \{$*halted-point*$\}$
by *auto*
then have *ifD*: *Suc i* $\in \mathcal{B} \vee$ *Suc i* $\in \mathcal{R} \vee$ *Suc i* $\in \mathcal{S} \vee$ *Suc i* = *halted-point* **if** $i \in \mathcal{D}$ **for** *i*
using *that* **by** *force*
then have *card* $\mathcal{D} \leq$ *card* $(\mathcal{B} \cup (\mathcal{R} \cup \mathcal{S}) \cup \{$*halted-point*$\})$
by (*intro card-inj-on-le* [*of Suc*]) *auto*
also have ... = *card* $\mathcal{B}$ + *card* $\mathcal{R}$ + *card* $\mathcal{S}$ + *1*
by (*simp add*: *card-Un-disjoint card-insert-if*)
also have ... $\leq k + l + 1$
using ‹*card* $\mathcal{R} < k$› *less-l* **by** *linarith*
finally have *card-D*: *card* $\mathcal{D} \leq k + l + 1$ **.**

have (*1* − *2* ∗ ε *powr* (*1/4*)) ∗ *card* (*Xseq 0*) ≤ *1* ∗ *real* (*card* (*Xseq 0*))
by (*intro mult-right-mono*; *force*)
then have *0* ∉ *C*
by (*force simp*: *C-def*)
then have *C-eq-C′*: *C* = *Suc* ‘ *C′*
using *nat.exhaust* **by** (*auto simp*: *C′-def set-eq-iff image-iff*)
have *card* $(\mathcal{D} \cap C') \leq$ *real* (*card* $((\mathcal{B} \cup (\mathcal{R} \cup \mathcal{S}) \cup \{$*halted-point*$\}) \cap C)$)
using *ifD*
by (*intro of-nat-mono card-inj-on-le* [*of Suc*]) (*force simp*: *Int-insert-left C-eq-C′*)+
also have ... ≤ *card* $(\mathcal{B} \cap C)$ + *real* (*card* $((\mathcal{R} \cup \mathcal{S}) \cap C)$) + *1*
by (*simp add*: *Int-insert-left Int-Un-distrib2 card-Un-disjoint card-insert-if*)
also have ... ≤ *?DC k*
using *Bk-34 712* **by** *force*
finally show *card* $(\mathcal{D} \cap C') \leq$ *?DC k* **.**
have *card* $(\mathcal{D} \setminus C') \leq$ *card* $\mathcal{D}$
using ‹*finite* $\mathcal{D}$› **by** (*simp add*: *card-mono*)
then show (*1* − *2* ∗ ε *powr* (*1/4*)) ^ (*k*+*l*+*1*) ≤ (*1* − *2* ∗ ε *powr* (*1/4*)) ^ *card* $(\mathcal{D} \setminus C')$
by (*smt* (*verit*) *card-D add-leD2 one-minus-gt0 power-decreasing powr-ge-zero*)
qed (*use one-minus-gt0 kn0* **in** *auto*)
also have ... = (∏*i*∈$\mathcal{D}$. *if i* ∈ *C′ then 1* / *real k* ^ *2 else 1* − *2* ∗ ε *powr* (*1/4*))
by (*simp add*: *kn0 powr-realpow prod.If-cases Diff-eq*)
also have ... ≤ (∏ *i* ∈ $\mathcal{D}$. *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*))
using *X-pos X78 one-minus-gt0 kn0* **by** (*simp add*: *divide-simps C′-def C-def prod-mono*)
finally show *?thesis* **.**
qed

6.13 Lemma 7.1

definition *Big-X-7-1* ≡
  λμ l. Big-Blue-4-1 μ l ∧ Big-X-7-2 μ l ∧ Big-X-7-4 μ l ∧ Big-X-7-6 μ l

establishing the size requirements for 7.11

lemma *Big-X-7-1*:
  assumes *0<μ0 μ1<1*
  shows ∀∞*l*. ∀*μ*. *μ* ∈ {*μ0*..*μ1*} ⟶ *Big-X-7-1 μ l*
  unfolding *Big-X-7-1-def*
  using *assms Big-Blue-4-1 Big-X-7-2 Big-X-7-4 Big-X-7-6*
  by (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)

definition *ok-fun-71* ≡ *λμ k. ok-fun-72 μ k* + *ok-fun-73 k* + *ok-fun-74 k* + *ok-fun-76 k*

lemma *ok-fun-71*:
  assumes *0<μ μ<1*
  shows *ok-fun-71 μ* ∈ *o*(*real*)
  using *ok-fun-72 ok-fun-73 ok-fun-74 ok-fun-76*
  by (*simp add*: *assms ok-fun-71-def sum-in-smallo*)

lemma (**in** *Book*) *X-7-1*:
  assumes *big*: *Big-X-7-1 μ l*
  defines 𝒟 ≡ *Step-class* {*dreg-step*}
  defines ℛ ≡ *Step-class* {*red-step*} **and** 𝒮 ≡ *Step-class* {*dboost-step*}
  shows *card* (*Xseq halted-point*) ≥
    2 powr ok-fun-71 μ k ∗ *μ*^*l* ∗ (*1*−*μ*) ^ *card* ℛ ∗ (*bigbeta* / *μ*) ^ *card* 𝒮 ∗ *card X0*
proof −
  define ℬ **where** ℬ ≡ *Step-class* {*bblue-step*}
  have *72*: *Big-X-7-2 μ l* **and** *74*: *Big-X-7-4 μ l*
    and *76*: *Big-X-7-6 μ l*
    and *big41*: *Big-Blue-4-1 μ l*
    using *big* **by** (*auto simp*: *Big-X-7-1-def*)
  then have [*simp*]: *finite* ℛ *finite* ℬ *finite* 𝒮 *finite* 𝒟
      ℛ∩ℬ = {} 𝒮∩𝒟 = {} (ℛ∪ℬ)∩(𝒮∪𝒟) = {}
   using *finite-components* **by** (*auto simp*: ℛ*-def* ℬ*-def* 𝒮*-def* 𝒟*-def Step-class-def*)
  have *BS-le-l*: *card* ℬ + *card* 𝒮 < *l*
    using *big41 bblue-dboost-step-limit* **by** (*auto simp*: 𝒮*-def* ℬ*-def*)

  have *R*: (∏*i*∈ℛ. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr* (*ok-fun-72 μ k*) ∗ (*1*−*μ*) ^ *card* ℛ
    unfolding ℛ*-def* **using** *72 X-7-2* **by** *meson*
  have *B*: (∏*i*∈ℬ. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr* (*ok-fun-73 k*) ∗ *μ* ^ (*l* − *card* 𝒮)
    unfolding ℬ*-def* 𝒮*-def* **using** *big41 X-7-3* **by** *meson*
  have *S*: (∏*i*∈𝒮. *card* (*Xseq* (*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr ok-fun-74 k* ∗ *bigbeta* ^ *card* 𝒮
    unfolding 𝒮*-def* **using** *74 X-7-4* **by** *meson*

have *D*: ($\prod$ *i*∈$\mathcal{D}$. *card*(*Xseq*(*Suc i*)) / *card* (*Xseq i*)) ≥ *2 powr ok-fun-76 k*
  unfolding *$\mathcal{D}$-def* **using** *76 X-7-6* **by** *meson*
have *below-m*: $\mathcal{R}$∪$\mathcal{B}$∪$\mathcal{S}$∪$\mathcal{D}$ = {..<*halted-point*}
  using *assms* **by** (*auto simp*: *$\mathcal{R}$-def $\mathcal{B}$-def $\mathcal{S}$-def $\mathcal{D}$-def before-halted-eq Step-class-insert-NO-MATCH*)
have *X-nz*: ⋀*i*. *i* < *halted-point* ⟹ *card* (*Xseq i*) ≠ *0*
  using *assms below-halted-point-cardX* **by** *blast*
have *tele*: *card* (*Xseq halted-point*) = ($\prod$ *i* < *halted-point*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*)) ∗ *card* (*Xseq 0*)
proof (*cases halted-point=0*)
  case *False*
  with *X-nz prod-lessThan-telescope-mult* [**where** *f* = λ*i*. *real* (*card* (*Xseq i*))]
  show *?thesis* **by** *simp*
qed *auto*
have *X0-nz*: *card* (*Xseq 0*) > *0*
  by (*simp add*: *card-XY0*)
have *2 powr ok-fun-71 μ k* ∗ *μ^l* ∗ (*1−μ*) ^ *card* $\mathcal{R}$ ∗ (*bigbeta* / *μ*) ^ *card* $\mathcal{S}$
    ≤ *2 powr ok-fun-71 μ k* ∗ *μ* ^ (*l* − *card* $\mathcal{S}$) ∗ (*1−μ*) ^ *card* $\mathcal{R}$ ∗ (*bigbeta* ^ *card* $\mathcal{S}$)
  using *μ01 BS-le-l* **by** (*simp add*: *power-diff power-divide*)
also have ... ≤ ($\prod$ *i*∈$\mathcal{R}$∪$\mathcal{B}$∪$\mathcal{S}$∪$\mathcal{D}$. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
proof −
  have ($\prod$ *i*∈($\mathcal{R}$∪$\mathcal{B}$)∪($\mathcal{S}$∪$\mathcal{D}$). *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
      ≥ ((*2 powr* (*ok-fun-72 μ k*) ∗ (*1−μ*) ^ *card* $\mathcal{R}$) ∗ (*2 powr* (*ok-fun-73 k*) ∗ *μ* ^ (*l* − *card* $\mathcal{S}$)))
      ∗ ((*2 powr ok-fun-74 k* ∗ *bigbeta* ^ *card* $\mathcal{S}$) ∗ (*2 powr ok-fun-76 k*))
   using *μ01* **by** (*auto simp*: *R B S D prod.union-disjoint prod-nonneg bigbeta-ge0 intro*!: *mult-mono*)
  then show *?thesis*
    by (*simp add*: *Un-assoc mult-ac powr-add ok-fun-71-def*)
qed
also have ... ≤ ($\prod$ *i* < *halted-point*. *card* (*Xseq*(*Suc i*)) / *card* (*Xseq i*))
  using *below-m* **by** *auto*
finally show *?thesis*
  using *X0-nz μ01* **unfolding** *tele* **by** (*simp add*: *divide-simps*)
qed

end

7 The Zigzag Lemma

theory *Zigzag* **imports** *Bounding-X*

begin

7.1 Lemma 8.1 (the actual Zigzag Lemma)

definition *Big-ZZ-8-2* ≡ λ*k*. (*1* + *eps k powr* (*1/2*)) ≥ (*1* + *eps k*) *powr* (*eps k powr* (−*1/4*))

An inequality that pops up in the proof of (39)

definition *Big39* ≡ λk. 1/2 ≤ (1 + eps k) powr (−2 ∗ eps k powr (−1/2))

Two inequalities that pops up in the proof of (42)

definition *Big42a* ≡ λk. (1 + eps k)² / (1 − eps k powr (1/2)) ≤ 1 + 2 ∗ k powr (−1/16)

definition *Big42b* ≡ λk. 2 ∗ k powr (−1/16) ∗ k
+ (1 + 2 ∗ ln k / eps k + 2 ∗ k powr (7/8)) / (1 − eps k powr (1/2))
≤ real k powr (19/20)

definition *Big-ZZ-8-1* ≡
λμ l. Big-Blue-4-1 μ l ∧ Big-Red-5-1 μ l ∧ Big-Red-5-3 μ l ∧ Big-Y-6-5-Bblue l
∧ (∀ k. k≥l ⟶ Big-height-upper-bound k ∧ Big-ZZ-8-2 k ∧ k≥16 ∧ Big39 k
∧ Big42a k ∧ Big42b k)

(16::'a) ≤ k is for *Y-6-5-Red*

lemma *Big-ZZ-8-1*:
assumes 0<μ0 μ1<1
shows ∀[∞]l. ∀μ. μ ∈ {μ0..μ1} ⟶ Big-ZZ-8-1 μ l
using *assms Big-Blue-4-1 Big-Red-5-1 Big-Red-5-3 Big-Y-6-5-Bblue*
unfolding *Big-ZZ-8-1-def Big-ZZ-8-2-def Big39-def Big42a-def Big42b-def eventually-conj-iff all-imp-conj-distrib eps-def*
apply (*simp add*: *eventually-conj-iff eventually-frequently-const-simps*)
apply (*intro conjI strip eventually-all-ge-at-top Big-height-upper-bound*; *real-asymp*)
done

lemma (**in** *Book*) *ZZ-8-1*:
assumes *big*: *Big-ZZ-8-1 μ l*
defines 𝓡 ≡ *Step-class* {*red-step*}
defines *sum-SS* ≡ (∑ i∈dboost-star. (1 − beta i) / beta i)
shows *sum-SS* ≤ *card* 𝓡 + k powr (19/20)
proof −
define *pp* **where** *pp* ≡ λi h. if h=1 then min (pseq i) (qfun 1)
else if pseq i ≤ qfun (h−1) then qfun (h−1)
else if pseq i ≥ qfun h then qfun h
else pseq i
define Δ **where** Δ ≡ λi. pseq (Suc i) − pseq i
define ΔΔ **where** ΔΔ ≡ λi h. pp (Suc i) h − pp i h
have *pp-eq*: pp i h = (if h=1 then min (pseq i) (qfun 1)
else max (qfun (h−1)) (min (pseq i) (qfun h))) **for** i h
using *qfun-mono* [*of h−1 h*] **by** (*auto simp*: *pp-def max-def*)

define *maxh* **where** *maxh* ≡ nat⌊2 ∗ ln k / ε⌋ + 1
have *maxh*: ⋀pseq. pseq≤1 ⟹ hgt pseq ≤ 2 ∗ ln k / ε **and** k≥16
using *big l-le-k* **by** (*auto simp*: *Big-ZZ-8-1-def height-upper-bound*)

```
then have 1 ≤ 2 ∗ ln k / ε
  using hgt-gt0 [of 1] by force
then have maxh > 1
  by (simp add: maxh-def eps-gt0)
have hgt pseq < maxh if pseq≤1 for pseq
  using that kn0 maxh[of pseq] unfolding maxh-def by linarith
then have hgt-le-maxh: hgt (pseq i) < maxh for i
  using pee-le1 by auto

have pp-eq-hgt [simp]: pp i (hgt (pseq i)) = pseq i for i
  using hgt-less-imp-qfun-less [of hgt (pseq i) − 1 pseq i]
  using hgt-works [of pseq i] hgt-gt0 [of pseq i] kn0 pp-eq by force

have pp-less-hgt [simp]: pp i h = qfun h if 0<h h < hgt (pseq i) for h i
proof (cases h=1)
  case True
  then show ?thesis
    using hgt-less-imp-qfun-less pp-def that by auto
next
  case False
  with that show ?thesis
    using alpha-def alpha-ge0 hgt-less-imp-qfun-less pp-eq by force
qed

have pp-gt-hgt [simp]: pp i h = qfun (h−1) if h > hgt (pseq i) for h i
  using hgt-gt0 [of pseq i] kn0 that
  by (simp add: pp-def hgt-le-imp-qfun-ge)

have Δ0: Δ i ≥ 0 ⟷ (∀h>0. ΔΔ i h ≥ 0) for i
proof (intro iffI strip)
  fix h::nat
  assume 0 ≤ Δ i 0 < h then show 0 ≤ ΔΔ i h
    using qfun-mono [of h−1 h] kn0 by (auto simp: Δ-def ΔΔ-def pp-def)
next
  assume ∀h>0. 0 ≤ ΔΔ i h
  then have pseq i ≤ pp (Suc i) (hgt (pseq i))
    unfolding ΔΔ-def
    by (smt (verit, best) hgt-gt0 pp-eq-hgt)
  then show 0 ≤ Δ i
    using hgt-less-imp-qfun-less [of hgt (pseq i) − 1 pseq i]
    using hgt-gt0 [of pseq i] kn0
    by (simp add: Δ-def pp-def split: if-split-asm)
qed

have sum-pp-aux: (∑h=Suc 0..n. pp i h)
                  = (if hgt (pseq i) ≤ n then pseq i + (∑h=1..<n. qfun h) else
(∑h=1..n. qfun h))
  if n>0 for n i
  using that
```

proof (*induction n*)
case (*Suc n*)
show *?case*
proof (*cases n=0*)
case *True*
then show *?thesis*
using *kn0 hgt-Least* [*of 1 pseq i*]
by (*simp add*: *pp-def hgt-le-imp-qfun-ge min-def*)
next
case *False*
with *Suc* **show** *?thesis*
by (*simp split*: *if-split-asm*) (*smt* (*verit*) *le-Suc-eq not-less-eq pp-eq-hgt sum.head-if*)
qed
qed *auto*
have *sum-pp*: (∑*h=Suc 0..maxh. pp i h*) = *pseq i* + (∑*h=1..<maxh. qfun h*) **for** *i*
using ‹*1* < *maxh*› **by** (*simp add*: *hgt-le-maxh less-or-eq-imp-le sum-pp-aux*)
have *33*: Δ *i* = (∑*h=1..maxh.* ΔΔ *i h*) **for** *i*
by (*simp add*: ΔΔ*-def* Δ*-def sum-subtractf sum-pp*)

have (∑*i<halted-point.* ΔΔ *i h*) = *0*
if ⋀*i. i≤halted-point* ⟹ *h* > *hgt* (*pseq i*) **for** *h*
using *that* **by** (*simp add*: *sum.neutral* ΔΔ*-def*)
then have *B*: (∑*i<halted-point.* ΔΔ *i h*) = *0* **if** *h* ≥ *maxh* **for** *h*
by (*meson hgt-le-maxh le-simps le-trans not-less-eq that*)
have (∑*h=Suc 0..maxh.* ∑*i<halted-point.* ΔΔ *i h* / *alpha h*) ≤ (∑*h=Suc 0..maxh. 1*)
proof (*intro sum-mono*)
fix *h*
assume *h* ∈ {*Suc 0..maxh*}
have (∑*i<halted-point.* ΔΔ *i h*) ≤ *alpha h*
using *qfun-mono* [*of h−1 h*] *kn0*
unfolding ΔΔ*-def alpha-def sum-lessThan-telescope* [**where** *f* = λ*i. pp i h*]
by (*auto simp*: *pp-def pee-eq-p0*)
then show (∑*i<halted-point.* ΔΔ *i h* / *alpha h*) ≤ *1*
using *alpha-ge0* [*of h*] **by** (*simp add*: *divide-simps flip*: *sum-divide-distrib*)
qed
also have ... ≤ *1* + *2* ∗ *ln k* / ε
using ‹*maxh* > *1*› **by** (*simp add*: *maxh-def*)
finally have *34*: (∑*h=Suc 0..maxh.* ∑*i<halted-point.* ΔΔ *i h* / *alpha h*) ≤ *1* + *2* ∗ *ln k* / ε **.**

define 𝒟 **where** 𝒟 ≡ *Step-class* {*dreg-step*}
define ℬ **where** ℬ ≡ *Step-class* {*bblue-step*}
define 𝒮 **where** 𝒮 ≡ *Step-class* {*dboost-step*}
have *dboost-star* ⊆ 𝒮
unfolding *dboost-star-def* 𝒮*-def dboost-star-def* **by** *auto*
have *BD-disj*: ℬ∩𝒟 = {} **and** *disj*: ℛ∩ℬ = {} 𝒮∩ℬ = {} ℛ∩𝒟 = {} 𝒮∩𝒟 =

{} $\mathcal{R} \cap \mathcal{S} = \{\}$
by (*auto simp*: $\mathcal{D}$*-def* $\mathcal{R}$*-def* $\mathcal{B}$*-def* $\mathcal{S}$*-def Step-class-def*)

have [*simp*]: *finite* $\mathcal{D}$ *finite* $\mathcal{B}$ *finite* $\mathcal{R}$ *finite* $\mathcal{S}$
using *finite-components assms*
by (*auto simp*: $\mathcal{D}$*-def* $\mathcal{B}$*-def* $\mathcal{R}$*-def* $\mathcal{S}$*-def Step-class-insert-NO-MATCH*)
have *card* $\mathcal{R} < k$
using *red-step-limit* **by** (*auto simp*: $\mathcal{R}$*-def*)

have *R52*: *pseq* (*Suc i*) − *pseq i* ≥ (*1* − ε) ∗ ((*1* − *beta i*) / *beta i*) ∗ *alpha* (*hgt* (*pseq i*))
and *beta-gt0*: *beta i* > *0*
and *R53*: *pseq* (*Suc i*) ≥ *pseq i* ∧ *beta i* ≥ *1* / (*real k*)^2
if $i \in \mathcal{S}$ **for** *i*
using *big Red-5-2 that* **by** (*auto simp*: *Big-ZZ-8-1-def Red-5-3* $\mathcal{B}$*-def* $\mathcal{S}$*-def*)
have *card*$\mathcal{B}$: *card* $\mathcal{B}$ ≤ *l powr* (*3/4*) **and** *bigY65B*: *Big-Y-6-5-Bblue l*
using *big bblue-step-limit* **by** (*auto simp*: *Big-ZZ-8-1-def* $\mathcal{B}$*-def*)

have ΔΔ*-ge0*: ΔΔ *i h* ≥ *0* **if** $i \in \mathcal{S}$ *h* ≥ *1* **for** *i h*
using *that R53* [*OF* ‹$i \in \mathcal{S}$›] **by** (*fastforce simp*: ΔΔ*-def pp-eq*)
have ΔΔ*-eq-0*: ΔΔ *i h* = *0* **if** *hgt* (*pseq i*) ≤ *hgt* (*pseq* (*Suc i*)) *hgt* (*pseq* (*Suc i*)) < *h* **for** *h i*
using ΔΔ*-def that* **by** *fastforce*
define *oneminus* **where** *oneminus* ≡ *1* − ε *powr* (*1/2*)
have *35*: *oneminus* ∗ ((*1* − *beta i*) / *beta i*)
≤ (∑*h*=*1*..*maxh*. ΔΔ *i h* / *alpha h*) (**is** *?L* ≤ *?R*)
if *i* ∈ *dboost-star* **for** *i*
proof −
have $i \in \mathcal{S}$
using ‹*dboost-star* ⊆ $\mathcal{S}$› *that* **by** *blast*
have [*simp*]: *real* (*hgt x* − *Suc 0*) = *real* (*hgt x*) − *1* **for** *x*
using *hgt-gt0* [*of x*] **by** *linarith*
have *36*: (*1* − ε) ∗ ((*1* − *beta i*) / *beta i*) ≤ Δ *i* / *alpha* (*hgt* (*pseq i*))
using *R52 alpha-gt0* [*OF hgt-gt0*] *beta-gt0 that* ‹*dboost-star* ⊆ $\mathcal{S}$› **by** (*force simp*: Δ*-def divide-simps*)
have *k-big*: (*1* + ε *powr* (*1/2*)) ≥ (*1* + ε) *powr* (ε *powr* (−*1/4*))
using *big l-le-k* **by** (*auto simp*: *Big-ZZ-8-1-def Big-ZZ-8-2-def*)
have ∗: ⋀*x*::*real*. *x* > *0* ⟹ (*1* − *x powr* (*1/2*)) ∗ (*1* + *x powr* (*1/2*)) = *1* − *x*
by (*simp add*: *algebra-simps flip*: *powr-add*)
have *?L* = (*1* − ε) ∗ ((*1* − *beta i*) / *beta i*) / (*1* + ε *powr* (*1/2*))
using *beta-gt0* [*OF* ‹$i \in \mathcal{S}$›] *eps-gt0 k-big*
by (*force simp*: *oneminus-def divide-simps* ∗)
also have . . . ≤ Δ *i* / *alpha* (*hgt* (*pseq i*)) / (*1* + ε *powr* (*1/2*))
by (*intro 36 divide-right-mono*) *auto*
also have . . . ≤ Δ *i* / *alpha* (*hgt* (*pseq i*)) / (*1* + ε) *powr* (*real* (*hgt* (*pseq* (*Suc i*))) − *hgt* (*pseq i*))
proof (*intro divide-left-mono mult-pos-pos*)
have *real* (*hgt* (*pseq* (*Suc i*))) − *hgt* (*pseq i*) ≤ ε *powr* (−*1/4*)

```
      using that by (simp add: dboost-star-def)
    then show (1 + ε) powr (real (hgt (pseq (Suc i))) − real (hgt (pseq i))) ≤
1 + ε powr (1/2)
      using k-big by (smt (verit) eps-ge0 powr-mono)
    show 0 ≤ Δ i / alpha (hgt (pseq i))
      by (simp add: Δ0 ΔΔ-ge0 ‹i ∈ S› alpha-ge0)
    show 0 < (1 + ε) powr (real (hgt (pseq (Suc i))) − real (hgt (pseq i)))
      using eps-gt0 by auto
  qed (auto simp: add-strict-increasing)
  also have ... ≤ Δ i / alpha (hgt (pseq (Suc i)))
  proof −
    have alpha (hgt (pseq (Suc i))) ≤ alpha (hgt (pseq i)) ∗ (1 + ε) powr (real
(hgt (pseq (Suc i))) − real (hgt (pseq i)))
      using eps-gt0 hgt-gt0
      by (simp add: alpha-eq divide-right-mono flip: powr-realpow powr-add)
    moreover have 0 ≤ Δ i
      by (simp add: Δ0 ΔΔ-ge0 ‹i ∈ S›)
    moreover have 0 < alpha (hgt (pseq (Suc i)))
      by (simp add: alpha-gt0 hgt-gt0 kn0)
    ultimately show ?thesis
      by (simp add: divide-left-mono)
  qed
  also have ... ≤ ?R
    unfolding 33 sum-divide-distrib
  proof (intro sum-mono)
    fix h
    assume h: h ∈ {1..maxh}
    show ΔΔ i h / alpha (hgt (pseq (Suc i))) ≤ ΔΔ i h / alpha h
    proof (cases  hgt (pseq i) ≤ hgt (pseq (Suc i)) ∧ hgt (pseq (Suc i)) < h)
      case False
      then consider hgt (pseq i) > hgt (pseq (Suc i)) | hgt (pseq (Suc i)) ≥ h
        by linarith
      then show ?thesis
      proof cases
        case 1
        then show ?thesis
          using R53 ‹i ∈ S› hgt-mono' kn0 by force
      next
        case 2
        have alpha h ≤ alpha (hgt (pseq (Suc i)))
          using 2 alpha-mono h by auto
        moreover have 0 ≤ ΔΔ i h
          using ΔΔ-ge0 ‹i ∈ S› h by presburger
        moreover have 0 < alpha h
          using h kn0 by (simp add: alpha-gt0 hgt-gt0)
        ultimately show ?thesis
          by (simp add: divide-left-mono)
      qed
    qed (auto simp: ΔΔ-eq-0)
```

qed
finally show *?thesis* **.**
qed
— now we are able to prove claim 8.2
have *oneminus* ∗ *sum-SS* = (∑ *i*∈*dboost-star*. *oneminus* ∗ ((*1* − *beta i*) / *beta i*))
using *sum-distrib-left sum-SS-def* **by** *blast*
also have ... ≤ (∑ *i*∈*dboost-star*. ∑ *h*=*1*..*maxh*. ΔΔ *i h* / *alpha h*)
by (*intro sum-mono 35*)
also have ... = (∑ *h*=*1*..*maxh*. ∑ *i*∈*dboost-star*. ΔΔ *i h* / *alpha h*)
using *sum.swap* **by** *fastforce*
also have ... ≤ (∑ *h*=*1*..*maxh*. ∑ *i*∈𝒮. ΔΔ *i h* / *alpha h*)
by (*intro sum-mono sum-mono2*) (*auto simp*: ‹*dboost-star* ⊆ 𝒮› ΔΔ-*ge0 alpha-ge0*)
finally have *82*: *oneminus* ∗ *sum-SS*
≤ (∑ *h*=*1*..*maxh*. ∑ *i*∈𝒮. ΔΔ *i h* / *alpha h*) **.**
— leading onto claim 8.3
have Δ*alpha*: − *1* ≤ Δ *i* / *alpha* (*hgt* (*pseq i*)) **if** *i* ∈ ℛ **for** *i*
using *Y-6-4-Red* [*of i*] ‹*i* ∈ ℛ›
unfolding Δ-*def* ℛ-*def*
by (*smt* (*verit*, *best*) *hgt-gt0 alpha-gt0 divide-minus-left less-divide-eq-1-pos*)

have (∑ *i*∈ℛ. − (*1* + ε)²) ≤ (∑ *i*∈ℛ. ∑ *h*=*1*..*maxh*. ΔΔ *i h* / *alpha h*)
proof (*intro sum-mono*)
fix *i* :: *nat*
assume *i* ∈ ℛ
show − (*1* + ε)² ≤ (∑ *h* = *1*..*maxh*. ΔΔ *i h* / *alpha h*)
proof (*cases* Δ *i* < *0*)
case *True*
have (*1* + ε)² ∗ −*1* ≤ (*1* + ε)² ∗ (Δ *i* / *alpha* (*hgt* (*pseq i*)))
using Δ*alpha*
by (*smt* (*verit*, *best*) *power2-less-0* ‹*i* ∈ ℛ› *mult-le-cancel-left2 mult-minus-right*)
also have ... ≤ (∑ *h* = *1*..*maxh*. ΔΔ *i h* / *alpha h*)
proof −
have *le0*: ΔΔ *i h* ≤ *0* **for** *h*
using *True* **by** (*auto simp*: ΔΔ-*def* Δ-*def pp-eq*)
have *eq0*: ΔΔ *i h* = *0* **if** *1* ≤ *h h* < *hgt* (*pseq i*) − *2* **for** *h*
proof −
have *hgt* (*pseq i*) − *2* ≤ *hgt* (*pseq* (*Suc i*))
using *Y-6-5-Red* ‹*16* ≤ *k*› ‹*i* ∈ ℛ› **unfolding** ℛ-*def* **by** *blast*
then show *?thesis*
using *that pp-less-hgt*[*of h*] **by** (*auto simp*: ΔΔ-*def pp-def*)
qed
show *?thesis*
unfolding *33 sum-distrib-left sum-divide-distrib*
proof (*intro sum-mono*)
fix *h* :: *nat*
assume *h* ∈ {*1*..*maxh*}
then have *1* ≤ *h h* ≤ *maxh* **by** *auto*

```
        show (1 + ε)² ∗ (ΔΔ i h / alpha (hgt (pseq i))) ≤ ΔΔ i h / alpha h
        proof (cases h < hgt (pseq i) − 2)
          case True
          then show ?thesis
            using ‹1 ≤ h› eq0 by force
        next
          case False
          have ∗: (1 + ε) ^ (hgt (pseq i) − Suc 0) ≤ (1 + ε)² ∗ (1 + ε) ^ (h −
Suc 0)
            using False eps-ge0 unfolding power-add [symmetric]
            by (intro power-increasing) auto
          have ∗∗: (1 + ε)² ∗ alpha h ≥ alpha (hgt (pseq i))
            using ‹1 ≤ h› mult-left-mono [OF ∗, of ε] eps-ge0
            by (simp add: alpha-eq hgt-gt0 mult-ac divide-right-mono)
          show ?thesis
            using le0 alpha-gt0 ‹h≥1› hgt-gt0 mult-left-mono-neg [OF ∗∗, of ΔΔ
i h]
            by (simp add: divide-simps mult-ac)
        qed
      qed
    qed
    finally show ?thesis
      by linarith
  next
    case False
    then have ΔΔ i h ≥ 0 for h
      using ΔΔ-def Δ-def pp-eq by auto
    then have (∑h = 1..maxh. ΔΔ i h / alpha h) ≥ 0
      by (simp add: alpha-ge0 sum-nonneg)
    then show ?thesis
      by (smt (verit, ccfv-SIG) sum-power2-ge-zero)
  qed
qed
then have 83: − (1 + ε)² ∗ card ℛ ≤ (∑h=1..maxh. ∑i∈ℛ. ΔΔ i h / alpha
h)
  by (simp add: mult.commute sum.swap [of - ℛ])

— now to tackle claim 8.4

have Δ0: Δ i ≥ 0 if i ∈ 𝒟 for i
  using Y-6-4-DegreeReg that unfolding 𝒟-def Δ-def by auto

have 39: −2 ∗ ε powr(−1/2) ≤ (∑h = 1..maxh. (ΔΔ (i−1) h + ΔΔ i h) /
alpha h) (is ?L ≤ ?R)
  if i ∈ ℬ for i
proof −
  have odd i
    using step-odd that by (force simp: Step-class-insert-NO-MATCH ℬ-def)
  then have i>0
```

using *odd-pos* **by** *auto*
show *?thesis*
proof (*cases* Δ (*i*−*1*) + Δ *i* ≥ *0*)
case *True*
with ‹*i*>*0*› **have** ΔΔ (*i*−*1*) *h* + ΔΔ *i h* ≥ *0* **if** *h*≥*1* **for** *h*
by (*fastforce simp*: ΔΔ-*def* Δ-*def pp-eq*)
then have (∑ *h* = *1*..*maxh*. (ΔΔ (*i*−*1*) *h* + ΔΔ *i h*) / *alpha h*) ≥ *0*
by (*force simp*: *alpha-ge0 intro*: *sum-nonneg*)
then show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *powr-ge-zero*)
next
case *False*
then have ΔΔ-*le0*: ΔΔ (*i*−*1*) *h* + ΔΔ *i h* ≤ *0* **if** *h*≥*1* **for** *h*
by (*smt* (*verit*, *best*) *One-nat-def* ΔΔ-*def* Δ-*def* ‹*odd i*› *odd-Suc-minus-one pp-eq*)
have *hge*: *hgt* (*pseq* (*Suc i*)) ≥ *hgt* (*pseq* (*i*−*1*)) − *2* ∗ ε *powr* (−*1*/*2*)
using *bigY65B that Y-6-5-Bblue* **by** (*fastforce simp*: *ℬ-def*)
have ΔΔ*0*: ΔΔ (*i*−*1*) *h* + ΔΔ *i h* = *0* **if** *0*<*h h* < *hgt* (*pseq* (*i*−*1*)) − *2* ∗ ε *powr* (−*1*/*2*) **for** *h*
using ‹*odd i*› *that hge* **unfolding** ΔΔ-*def One-nat-def*
by (*smt* (*verit*) *of-nat-less-iff odd-Suc-minus-one powr-non-neg pp-less-hgt*)
have *big39*: *1*/*2* ≤ (*1* + ε) *powr* (−*2* ∗ ε *powr* (−*1*/*2*))
using *big l-le-k* **by** (*auto simp*: *Big-ZZ-8-1-def Big39-def*)
have *?L* ∗ *alpha* (*hgt* (*pseq* (*i*−*1*))) ∗ (*1* + ε) *powr* (−*2* ∗ ε *powr* (−*1*/*2*))
≤ − (ε *powr* (−*1*/*2*)) ∗ *alpha* (*hgt* (*pseq* (*i*−*1*)))
using *mult-left-mono-neg* [*OF big39*, *of* − (ε *powr* (−*1*/*2*)) ∗ *alpha* (*hgt* (*pseq* (*i*−*1*))) / *2*]
using *alpha-ge0* [*of hgt* (*pseq* (*i*−*1*))] *eps-ge0*
by (*simp add*: *mult-ac*)
also have . . . ≤ Δ (*i*−*1*) + Δ *i*
proof −
have *pseq* (*Suc i*) ≥ *pseq* (*i*−*1*) − (ε *powr* (−*1*/*2*)) ∗ *alpha* (*hgt* (*pseq* (*i*−*1*)))
using *Y-6-4-Bblue that ℬ-def* **by** *blast*
with ‹*i*>*0*› **show** *?thesis*
by (*simp add*: Δ-*def*)
qed
finally have *?L* ∗ *alpha* (*hgt* (*pseq* (*i*−*1*))) ∗ (*1* + ε) *powr* (−*2* ∗ ε *powr* (−*1*/*2*)) ≤ Δ (*i*−*1*) + Δ *i* **.**
then have *?L* ≤ (*1* + ε) *powr* (*2* ∗ ε *powr* (−*1*/*2*)) ∗ (Δ (*i*−*1*) + Δ *i*) / *alpha* (*hgt* (*pseq* (*i*−*1*)))
using *alpha-ge0* [*of hgt* (*pseq* (*i*−*1*))] *eps-ge0*
by (*simp add*: *powr-minus divide-simps mult-ac*)
also have . . . ≤ *?R*
proof −
have (*1* + ε) *powr* (*2* ∗ ε *powr*(−*1*/*2*)) ∗ (ΔΔ (*i* − *Suc 0*) *h* + ΔΔ *i h*) / *alpha* (*hgt* (*pseq* (*i* − *Suc 0*)))
≤ (ΔΔ (*i* − *Suc 0*) *h* + ΔΔ *i h*) / *alpha h*
if *h*: *Suc 0* ≤ *h h* ≤ *maxh* **for** *h*

proof (*cases* $h < hgt\ (pseq\ (i-1)) - 2 * \varepsilon\ powr(-1/2)$)
case *False*
then have $hgt\ (pseq\ (i-1)) - 1 \le 2 * \varepsilon\ powr(-1/2) + (h - 1)$
using *hgt-gt0* **by** (*simp add*: *nat-less-real-le*)
then have $*$: $(1 + \varepsilon)\ powr\ (2 * \varepsilon\ powr(-1/2))\ /\ alpha\ (hgt\ (pseq\ (i-1)))$
$\ge 1\ /\ alpha\ h$
using *that eps-gt0 kn0 hgt-gt0*
by (*simp add*: *alpha-eq divide-simps flip*: *powr-realpow powr-add*)
show *?thesis*
using *mult-left-mono-neg* [*OF* $*$ $\Delta\Delta$*-le0*] *that* **by** (*simp add*: *Groups.mult-ac*)

qed (*use h* $\Delta\Delta 0$ **in** *auto*)
then show *?thesis*
by (*force simp*: *33 sum-distrib-left sum-divide-distrib simp flip*: *sum.distrib*
intro: *sum-mono*)
qed
finally show *?thesis* **.**
qed
qed

have *B34*: $card\ \mathcal{B} \le k\ powr\ (3/4)$
by (*smt* (*verit*) *card*$\mathcal{B}$ *l-le-k of-nat-0-le-iff of-nat-mono powr-mono2 zero-le-divide-iff*)
have $-2 * k\ powr\ (7/8) \le -2 * \varepsilon\ powr(-1/2) * k\ powr\ (3/4)$
by (*simp add*: *eps-def powr-powr flip*: *powr-add*)
also have $\ldots \le -2 * \varepsilon\ powr(-1/2) * card\ \mathcal{B}$
using *B34* **by** (*intro mult-left-mono-neg powr-mono2*) *auto*
also have $\ldots = (\sum i \in \mathcal{B}.\ -2 * \varepsilon\ powr(-1/2))$
by *simp*
also have $\ldots \le (\sum h = 1..maxh.\ \sum i \in \mathcal{B}.\ (\Delta\Delta\ (i-1)\ h + \Delta\Delta\ i\ h)\ /\ alpha\ h)$
unfolding *sum.swap* [*of* - $\mathcal{B}$] **by** (*intro sum-mono 39*)
also have $\ldots \le (\sum h=1..maxh.\ \sum i \in \mathcal{B} \cup \mathcal{D}.\ \Delta\Delta\ i\ h\ /\ alpha\ h)$
proof (*intro sum-mono*)
fix *h*
assume $h \in \{1..maxh\}$
have $\mathcal{B} \subseteq \{0<..\}$
using *odd-pos* [*OF step-odd*] **by** (*auto simp*: $\mathcal{B}$*-def Step-class-insert-NO-MATCH*)
with *inj-on-diff-nat* [*of* $\mathcal{B}$ *1*] **have** *inj-pred*: *inj-on* $(\lambda i.\ i - Suc\ 0)\ \mathcal{B}$
by (*simp add*: *Suc-leI subset-eq*)
have $(\sum i \in \mathcal{B}.\ \Delta\Delta\ (i - Suc\ 0)\ h) = (\sum i \in (\lambda i.\ i-1)$ ‘ $\mathcal{B}.\ \Delta\Delta\ i\ h)$
by (*simp add*: *sum.reindex* [*OF inj-pred*])
also have $\ldots \le (\sum i \in \mathcal{D}.\ \Delta\Delta\ i\ h)$
proof (*intro sum-mono2*)
show $(\lambda i.\ i - 1)$ ‘ $\mathcal{B} \subseteq \mathcal{D}$
by (*force simp*: $\mathcal{D}$*-def* $\mathcal{B}$*-def Step-class-insert-NO-MATCH intro*: *dreg-before-step'*)
show $0 \le \Delta\Delta\ i\ h$ **if** $i \in \mathcal{D} \setminus (\lambda i.\ i - 1)$ ‘ $\mathcal{B}$ **for** *i*
using *that* $\Delta 0$ $\Delta\Delta$*-def* Δ*-def pp-eq* **by** *fastforce*
qed *auto*
finally have $(\sum i \in \mathcal{B}.\ \Delta\Delta\ (i - Suc\ 0)\ h) \le (\sum i \in \mathcal{D}.\ \Delta\Delta\ i\ h)$ **.**
with *alpha-ge0* [*of h*]

show (∑i∈ℬ. (ΔΔ (i − 1) h + ΔΔ i h) / alpha h) ≤ (∑i ∈ ℬ∪𝒟. ΔΔ i h / alpha h)
by (simp add: BD-disj divide-right-mono sum.distrib sum.union-disjoint flip: sum-divide-distrib)
qed
finally have 84: −2 * k powr (7/8) ≤ (∑h=1..maxh. ∑i∈ℬ∪𝒟. ΔΔ i h / alpha h) **.**

have m-eq: {..<halted-point} = ℛ ∪ 𝒮 ∪ (ℬ ∪ 𝒟)
using before-halted-eq **by** (auto simp: ℬ-def 𝒟-def 𝒮-def ℛ-def Step-class-insert-NO-MATCH)

have − $(1 + \varepsilon)^2$ * real (card ℛ)
+ oneminus * sum-SS
− 2 * real k powr (7/8) ≤ (∑h = Suc 0..maxh. ∑i∈ℛ. ΔΔ i h / alpha h)
+ (∑h = Suc 0..maxh. ∑i∈𝒮. ΔΔ i h / alpha h)
+ (∑h = Suc 0..maxh. ∑i ∈ ℬ ∪ 𝒟. ΔΔ i h / alpha h)
using 82 83 84 **by** simp
also have ... = (∑h = Suc 0..maxh. ∑i ∈ ℛ ∪ 𝒮 ∪ (ℬ ∪ 𝒟). ΔΔ i h / alpha h)
by (simp add: sum.distrib disj sum.union-disjoint Int-Un-distrib Int-Un-distrib2)
also have ... ≤ 1 + 2 * ln (real k) / ε
using 34 **by** (simp add: m-eq)
finally
have 41: oneminus * sum-SS − $(1 + \varepsilon)^2$ * card ℛ − 2 * k powr (7/8)
≤ 1 + 2 * ln k / ε
by simp
have big42: $(1 + \varepsilon)^2$ / oneminus ≤ 1 + 2 * k powr (−1/16)
2 * k powr (−1/16) * k
+ (1 + 2 * ln k / ε + 2 * k powr (7/8)) / oneminus
≤ real k powr (19/20)
using big l-le-k **by** (auto simp: Big-ZZ-8-1-def Big42a-def Big42b-def oneminus-def)
have oneminus > 0
using ‹16 ≤ k› eps-gt0 eps-less1 powr01-less-one **by** (auto simp: oneminus-def)
with 41 **have** sum-SS
≤ (1 + 2 * ln k / ε + $(1 + \varepsilon)^2$ * card ℛ + 2 * k powr (7/8)) / oneminus
by (simp add: mult-ac pos-le-divide-eq diff-le-eq)
also have ... ≤ card ℛ * (($(1 + \varepsilon)^2$) / oneminus)
+ (1 + 2 * ln k / ε + 2 * k powr (7/8)) / oneminus
by (simp add: field-simps add-divide-distrib)
also have ... ≤ card ℛ * (1 + 2 * k powr (−1/16))
+ (1 + 2 * ln k / ε + 2 * k powr (7/8)) / oneminus
using big42 ‹oneminus > 0› **by** (intro add-mono mult-mono) auto
also have ... ≤ card ℛ + 2 * k powr (−1/16) * k
+ (1 + 2 * ln k / ε + 2 * k powr (7/8)) / oneminus
using ‹card ℛ < k› **by** (intro add-mono mult-mono) (auto simp: algebra-simps)
also have ... ≤ real (card ℛ) + real k powr (19/20)
using big42 **by** force
finally show ?thesis **.**
qed

7.2 Lemma 8.5

An inequality that pops up in the proof of (39)

```
definition inequality85 ≡ λk. 3 ∗ eps k powr (1/4) ∗ k ≤ k powr (19/20)

definition Big-ZZ-8-5 ≡
  λμ l. Big-X-7-5 μ l ∧ Big-ZZ-8-1 μ l ∧ Big-Red-5-3 μ l
    ∧ (∀ k≥l. inequality85 k)

lemma Big-ZZ-8-5:
  assumes 0<μ0 μ1<1
  shows ∀∞l. ∀ μ. μ ∈ {μ0..μ1} ⟶ Big-ZZ-8-5 μ l
  using assms Big-Red-5-3 Big-X-7-5 Big-ZZ-8-1
  unfolding Big-ZZ-8-5-def inequality85-def eps-def
  apply (simp add: eventually-conj-iff all-imp-conj-distrib)
  apply (intro conjI strip eventually-all-ge-at-top; real-asymp)
  done

lemma (in Book) ZZ-8-5:
  assumes big: Big-ZZ-8-5 μ l
  defines ℛ ≡ Step-class {red-step} and 𝒮 ≡ Step-class {dboost-step}
  shows card 𝒮 ≤ (bigbeta / (1 − bigbeta)) ∗ card ℛ
      + (2 / (1−μ)) ∗ k powr (19/20)
proof −
  have [simp]: finite 𝒮
    by (simp add: 𝒮-def)
  moreover have dboost-star ⊆ 𝒮
    by (auto simp: dboost-star-def 𝒮-def)
  ultimately have real (card 𝒮) − real (card dboost-star) = card (𝒮\dboost-star)
    by (metis card-Diff-subset card-mono finite-subset of-nat-diff)
  also have ... ≤ 3 ∗ ε powr (1/4) ∗ k
    using μ01 big X-7-5 by (auto simp: Big-ZZ-8-5-def dboost-star-def 𝒮-def)
  also have ... ≤ k powr (19/20)
    using big l-le-k by (auto simp: Big-ZZ-8-5-def inequality85-def)
  finally have ∗: real (card 𝒮) − card dboost-star ≤ k powr (19/20) .
  have bigbeta-lt1: bigbeta < 1 and bigbeta-gt0: 0 < bigbeta and beta-gt0: ⋀i. i
∈ 𝒮 ⟹ beta i > 0
    using bigbeta-ge0 big by (auto simp: Big-ZZ-8-5-def 𝒮-def beta-gt0 bigbeta-gt0
bigbeta-less1)
  then have ge0: bigbeta / (1 − bigbeta) ≥ 0
    by auto
  show ?thesis
  proof (cases dboost-star = {})
    case True
    with ∗ have card 𝒮 ≤ k powr (19/20)
      by simp
    also have ... ≤ (2 / (1−μ)) ∗ k powr (19/20)
      using μ01 kn0 by (simp add: divide-simps)
    finally show ?thesis
```

```
    by (smt (verit, ccfv-SIG) mult-nonneg-nonneg of-nat-0-le-iff ge0)
  next
    case False
    have bb-le: bigbeta ≤ μ
      using big bigbeta-le by (auto simp: Big-ZZ-8-5-def)
    have (card 𝒮 − k powr (19/20)) / bigbeta ≤ card dboost-star / bigbeta
      by (smt (verit) ∗ bigbeta-ge0 divide-right-mono)
    also have ... = (∑i∈dboost-star. 1 / beta i)
    proof (cases card dboost-star = 0)
      case False
      then show ?thesis
        by (simp add: bigbeta-def Let-def inverse-eq-divide)
    qed (simp add: False card-eq-0-iff)
    also have ... ≤ real(card dboost-star) + card ℛ + k powr (19/20)
    proof −
      have (∑i∈dboost-star. (1 − beta i) / beta i)
            ≤ real (card ℛ) + k powr (19/20)
        using ZZ-8-1 big unfolding Big-ZZ-8-5-def ℛ-def by blast
      moreover have (∑i∈dboost-star. beta i / beta i) = (∑i∈dboost-star. 1)
        using ‹dboost-star ⊆ 𝒮› beta-gt0 by (intro sum.cong) force+
      ultimately show ?thesis
        by (simp add: field-simps diff-divide-distrib sum-subtractf)
    qed
    also have ... ≤ real(card 𝒮) + card ℛ + k powr (19/20)
      by (simp add: ‹dboost-star ⊆ 𝒮› card-mono)
    finally have (card 𝒮 − k powr (19/20)) / bigbeta ≤ real (card 𝒮) + card ℛ
+ k powr (19/20) .
    then have card 𝒮 − k powr (19/20) ≤ (real (card 𝒮) + card ℛ + k powr
(19/20)) ∗ bigbeta
      using bigbeta-gt0 by (simp add: field-simps)
    then have card 𝒮 ∗ (1 − bigbeta) ≤ bigbeta ∗ card ℛ + (1 + bigbeta) ∗ k
powr (19/20)
      by (simp add: algebra-simps)
    then have card 𝒮 ≤ (bigbeta ∗ card ℛ + (1 + bigbeta) ∗ k powr (19/20)) /
(1 − bigbeta)
      using bigbeta-lt1 by (simp add: field-simps)
    also have ... = (bigbeta / (1 − bigbeta)) ∗ card ℛ
                + ((1 + bigbeta) / (1 − bigbeta)) ∗ k powr (19/20)
      using bigbeta-gt0 bigbeta-lt1 by (simp add: divide-simps)
    also have ... ≤ (bigbeta / (1 − bigbeta)) ∗ card ℛ + (2 / (1−μ)) ∗ k powr
(19/20)
      using μ01 bb-le by (intro add-mono order-refl mult-right-mono frac-le) auto
    finally show ?thesis .
  qed
qed
```

7.3 Lemma 8.6

For some reason this was harder than it should have been. It does require a further small limit argument.

```
definition Big-ZZ-8-6 ≡
  λμ l. Big-ZZ-8-5 μ l ∧ (∀k≥l. 2 / (1−μ) ∗ k powr (19/20) < k powr (39/40))

lemma Big-ZZ-8-6:
  assumes 0<μ0 μ1<1
  shows ∀∞l. ∀μ. μ ∈ {μ0..μ1} ⟶ Big-ZZ-8-6 μ l
  using assms Big-ZZ-8-5
  unfolding Big-ZZ-8-6-def
  apply (simp add: eventually-conj-iff all-imp-conj-distrib)
  apply (intro conjI strip eventually-all-ge-at-top eventually-all-geI1 [where L=1])

   apply real-asymp
  by (smt (verit, ccfv-SIG) frac-le powr-ge-zero)

lemma (in Book) ZZ-8-6:
  assumes big: Big-ZZ-8-6 μ l
  defines ℛ ≡ Step-class {red-step} and 𝒮 ≡ Step-class {dboost-step}
    and a ≡ 2 / (1−μ)
  assumes s-ge: card 𝒮 ≥ k powr (39/40)
  shows bigbeta ≥ (1 − a ∗ k powr (−1/40)) ∗ (card 𝒮 / (card 𝒮 + card ℛ))
proof −
  have bigbeta-lt1: bigbeta < 1 and bigbeta-gt0: 0 < bigbeta
    using bigbeta-ge0 big
    by (auto simp: Big-ZZ-8-6-def Big-ZZ-8-5-def bigbeta-less1 bigbeta-gt0 𝒮-def)
  have a > 0
    using μ01 by (simp add: a-def)
  have s-gt-a: a ∗ k powr (19/20) < card 𝒮
       and 85: card 𝒮 ≤ (bigbeta / (1 − bigbeta)) ∗ card ℛ + a ∗ k powr (19/20)
    using big l-le-k assms
    unfolding ℛ-def 𝒮-def a-def Big-ZZ-8-6-def by (fastforce intro: ZZ-8-5)+
  then have t-non0: card ℛ ≠ 0 — seemingly not provable without our assumption
    using mult-eq-0-iff by fastforce
  then have (card 𝒮 − a ∗ k powr (19/20)) / card ℛ ≤ bigbeta / (1 − bigbeta)
    using 85 bigbeta-gt0 bigbeta-lt1 t-non0 by (simp add: pos-divide-le-eq)
  then have bigbeta ≥ (1 − bigbeta) ∗ (card 𝒮 − a ∗ k powr (19/20)) / card ℛ
   by (smt (verit, ccfv-threshold) bigbeta-lt1 mult.commute le-divide-eq times-divide-eq-left)
  then have ∗: bigbeta ∗ (card ℛ + card 𝒮 − a ∗ k powr (19/20)) ≥ card 𝒮 − a
∗ k powr (19/20)
    using t-non0 by (simp add: field-simps)
  have (1 − a ∗ k powr − (1/40)) ∗ card 𝒮 ≤ card 𝒮 − a ∗ k powr (19/20)
      using s-ge kn0 ‹a>0› t-non0 by (simp add: powr-minus field-simps flip:
powr-add)
  then have (1 − a ∗ k powr (−1/40)) ∗ (card 𝒮 / (card 𝒮 + card ℛ))
         ≤ (card 𝒮 − a ∗ k powr (19/20)) / (card 𝒮 + card ℛ)
    by (force simp: divide-right-mono)
```

```
  also have ... ≤ (card 𝒮 − a ∗ k powr (19/20)) / (card ℛ + card 𝒮 − a ∗ k powr (19/20))
    using s-gt-a ‹a>0› t-non0 by (intro divide-left-mono) auto
  also have ... ≤ bigbeta
    using ∗ s-gt-a
    by (simp add: divide-simps split: if-split-asm)
  finally show ?thesis .
qed

end
```

8 An exponential improvement far from the diagonal

```
theory Far-From-Diagonal
  imports Zigzag Stirling-Formula.Stirling-Formula

begin
```

8.1 An asymptotic form for binomial coefficients via Stirling's formula

From Appendix D.3, page 56

```
lemma const-smallo-real: (λn. x) ∈ o(real)
  by real-asymp

lemma o-real-shift:
  assumes f ∈ o(real)
  shows (λi. f(i+j)) ∈ o(real)
  unfolding smallo-def
proof clarify
  fix c :: real
  assume (0::real) < c
  then have ∗: ∀F i in sequentially. norm (f i) ≤ c/2 ∗ norm i
    using assms half-gt-zero landau-o.smallD by blast
  have ∀F i in sequentially. norm (f (i + j)) ≤ c/2 ∗ norm (i + j)
    using eventually-all-ge-at-top [OF ∗]
    by (metis (mono-tags, lifting) eventually-sequentially le-add1)
  then have ∀F i in sequentially. i≥j ⟶ norm (f (i + j)) ≤ c ∗ norm i
    apply eventually-elim
    apply clarsimp
    by (smt (verit, best) ‹0 < c› mult-left-mono nat-distrib(2) of-nat-mono)
  then show ∀F i in sequentially. norm (f (i + j)) ≤ c ∗ norm i
    using eventually-mp by fastforce
qed

lemma tendsto-zero-imp-o1:
  fixes a :: nat ⇒ real
```

assumes *a* ⟶ *0*
shows *a* ∈ *o*(*1*)
proof –
have ∀$_F$ *n in sequentially*. |*a n*| ≤ *c* **if** *c*>*0* **for** *c*
using *assms order-tendstoD*(*2*) *tendsto-rabs-zero-iff eventually-sequentially less-eq-real-def that*
by *metis*
then show *?thesis*
by (*auto simp*: *smallo-def*)
qed

8.2 Fact D.3 from the Appendix

And hence, Fact 9.4

definition *stir* ≡ λ*n*. *fact n* / (*sqrt* (*2*∗*pi*∗*n*) ∗ (*n* / *exp 1*) ˆ *n*) − *1*

Generalised to the reals to allow derivatives

definition *stirG* ≡ λ*n*. *Gamma* (*n*+*1*) / (*sqrt* (*2*∗*pi*∗*n*) ∗ (*n* / *exp 1*) *powr n*) − *1*

lemma *stir-eq-stirG*: *n*>*0* ⟹ *stir n* = *stirG* (*real n*)
by (*simp add*: *stirG-def stir-def add.commute powr-realpow Gamma-fact*)

lemma *stir-ge0*: *n*>*0* ⟹ *stir n* ≥ *0*
using *fact-bounds*[*of n*] **by** (*simp add*: *stir-def*)

lemma *stir-to-0*: *stir* ⟶ *0*
using *fact-asymp-equiv* **by** (*simp add*: *asymp-equiv-def stir-def LIM-zero*)

lemma *stir-o1*: *stir* ∈ *o*(*1*)
using *stir-to-0 tendsto-zero-imp-o1* **by** *presburger*

lemma *fact-eq-stir-times*: *n* ≠ *0* ⟹ *fact n* = (*1* + *stir n*) ∗ (*sqrt* (*2*∗*pi*∗*n*) ∗ (*n* / *exp 1*) ˆ *n*)
by (*simp add*: *stir-def*)

definition *logstir* ≡ λ*n*. *if n*=*0 then 0 else log 2* ((*1* + *stir n*) ∗ *sqrt* (*2*∗*pi*∗*n*))

lemma *logstir-o-real*: *logstir* ∈ *o*(*real*)
proof –
have ∀$^\infty$*n*. *0* < *n* ⟶ |*log 2* ((*1* + *stir n*) ∗ *sqrt* (*2*∗*pi*∗*n*))| ≤ *c* ∗ *real n* **if** *c*>*0* **for** *c*
proof –
have ∀$^\infty$*n*. *2 powr* (*c*∗*n*) / *sqrt* (*2*∗*pi*∗*n*) ≥ *c*+*1*
using *that* **by** *real-asymp*
moreover have ∀$^\infty$*n*. |*stir n*| ≤ *c*
using *stir-o1 that* **by** (*auto simp*: *smallo-def*)
ultimately have ∀$^\infty$*n*. ((*1* + *stir n*) ∗ *sqrt* (*2*∗*pi*∗*n*)) ≤ *2 powr* (*c* ∗ *n*)
proof *eventually-elim*

fix *n*
assume *c1*: *c+1* ≤ *2 powr* (*c* ∗ *n*) / *sqrt* (*2*∗*pi*∗*n*) **and** *lec*: |*stir n*| ≤ *c*
then have *stir n* ≤ *c*
by *auto*
then show (*1* + *stir n*) ∗ *sqrt* (*2*∗*pi*∗*n*) ≤ *2 powr* (*c*∗*n*)
using *mult-right-mono* [*OF c1*, *of sqrt* (*2*∗*pi*∗*n*)] *lec*
by (*smt* (*verit*, *ccfv-SIG*) *c1 mult-right-mono nonzero-eq-divide-eq pos-prod-le powr-gt-zero*)
qed
then show *?thesis*
proof (*eventually-elim*, *clarify*)
fix *n*
assume *n*: (*1* + *stir n*) ∗ *sqrt* (*2* ∗ *pi* ∗ *n*) ≤ *2 powr* (*c* ∗ *n*)
and *n*>*0*
have (*1* + *stir n*) ∗ *sqrt* (*2* ∗ *pi* ∗ *real n*) ≥ *1*
using *stir-ge0* ‹*0* < *n*› *mult-ge1-I pi-ge-two* **by** *auto*
with *n* **show** |*log 2* ((*1* + *stir n*) ∗ *sqrt* (*2* ∗ *pi* ∗ *n*))| ≤ *c* ∗ *n*
by (*simp add*: *abs-if le-powr-iff*)
qed
qed
then show *?thesis*
by (*auto simp*: *smallo-def logstir-def*)
qed

lemma *logfact-eq-stir-times*:
fact n = *2 powr* (*logstir n*) ∗ (*n* / *exp 1*) ^ *n*
proof−
have *1* + *stir n* > *0* **if** *n*≠*0*
using *that* **by** (*simp add*: *stir-def*)
then show *?thesis*
by (*simp add*: *logstir-def fact-eq-stir-times*)
qed

lemma *mono-G*:
defines *G* ≡ (λ*x*::*real*. *Gamma* (*x* + *1*) / (*x* / *exp 1*) *powr x*)
shows *mono-on* {*0*<..} *G*
unfolding *monotone-on-def*
proof (*intro strip*)
fix *x y*::*real*
assume *x*: *x* ∈ {*0*<..} *x* ≤ *y*
define *GD* **where** *GD* ≡ λ*u*::*real*. *Gamma*(*u*+*1*) ∗ (*Digamma*(*u*+*1*) − *ln*(*u*)) / (*u* / *exp 1*) *powr u*
have ∗: ∃*D*. (*G has-real-derivative D*) (*at u*) ∧ *D* > *0* **if** *0* < *u* **for** *u*
proof (*intro exI conjI*)
show (*G has-real-derivative GD u*) (*at u*)
unfolding *G-def GD-def*
using *that*
by (*force intro*!: *derivative-eq-intros has-real-derivative-powr′ simp*: *ln-div pos-prod-lt field-simps*)

show *GD u* > *0*
using *that* **by** (*auto simp*: *GD-def Digamma-plus-1-gt-ln*) — Thank you, Manuel!
qed
show *G x* ≤ *G y*
using *x DERIV-pos-imp-increasing* [*OF* - ∗] **by** (*force simp*: *less-eq-real-def*)
qed

lemma *mono-logstir*: *mono logstir*
unfolding *monotone-on-def*
proof (*intro strip*)
fix *i j*::*nat*
assume *i*≤*j*
show *logstir i* ≤ *logstir j*
proof (*cases j*=*0*)
case *True*
with ‹*i*≤*j*› **show** *?thesis*
by *auto*
next
case *False*
with *pi-ge-two* **have** *1* ∗ *1* ≤ *2* ∗ *pi* ∗ *j*
by (*intro mult-mono*) *auto*
with *False stir-ge0* [*of j*] **have** ∗: *1* ∗ *1* ≤ (*1* + *stir j*) ∗ *sqrt* (*2* ∗ *pi* ∗ *real j*)
by (*intro mult-mono*) *auto*
with ‹*i* ≤ *j*› *mono-G* **show** *?thesis*
by (*auto simp*: *logstir-def stir-eq-stirG stirG-def monotone-on-def*)
qed
qed

definition *ok-fun-94* ≡ λ*k*. − *logstir k*

lemma *ok-fun-94*: *ok-fun-94* ∈ *o*(*real*)
unfolding *ok-fun-94-def*
using *logstir-o-real* **by** *simp*

lemma *fact-9-4*:
assumes *l*: *0* < *l l* ≤ *k*
defines γ ≡ *l* / (*real k* + *real l*)
shows *k*+*l choose l* ≥ *2 powr ok-fun-94 k* ∗ γ *powr* (−*l*) ∗ (*1*−γ) *powr* (−*k*)
proof −
have ∗: *ok-fun-94 k* ≤ *logstir* (*k*+*l*) − (*logstir k* + *logstir l*)
using *mono-logstir* **by** (*auto simp*: *ok-fun-94-def monotone-def*)
have *2 powr ok-fun-94 k* ∗ γ *powr* (− *real l*) ∗ (*1*−γ) *powr* (− *real k*)
= (*2 powr ok-fun-94 k*) ∗ (*k*+*l*) *powr*(*k*+*l*) / (*k powr k* ∗ *l powr l*)
by (*simp add*: γ*-def powr-minus powr-add powr-divide divide-simps*)
also have . . . ≤ (*2 powr* (*logstir* (*k*+*l*)) / (*2 powr* (*logstir k*) ∗ *2 powr* (*logstir l*)))
∗ (*k*+*l*) *powr* (*k*+*l*) / (*k powr k* ∗ *l powr l*)
by (*smt* (*verit*, *del-insts*) ∗ *divide-right-mono mult-less-0-iff mult-right-mono*

powr-add powr-diff powr-ge-zero powr-mono)
also have ... = *fact*(*k*+*l*) / (*fact k* ∗ *fact l*)
using *l* **by** (*simp add*: *logfact-eq-stir-times powr-add divide-simps flip*: *powr-realpow*)
also have ... = *real* (*k*+*l choose l*)
by (*simp add*: *binomial-fact*)
finally show *?thesis* **.**
qed

8.3 Fact D.2

For Fact 9.6

lemma *D2*:
fixes *k l*
assumes *t*: *0*<*t t* ≤ *k*
defines γ ≡ *l* / (*real k* + *real l*)
shows (*k*+*l*−*t choose l*) ≤ *exp* (− γ ∗ (*t*−*1*) ^*2* / (*2*∗*k*)) ∗ (*k* / (*k*+*l*)) ^*t* ∗ (*k*+*l choose l*)
proof −
have (*k*+*l*−*t choose l*) ∗ *inverse* (*k*+*l choose l*) = (∏ *i*<*t*. (*k*−*i*) / (*k*+*l*−*i*))
using ‹*t* ≤ *k*›
proof (*induction t*)
case (*Suc t*)
then have *t* ≤ *k*
by *simp*
have (*k* + *l* − *t*) ∗ (*k* + *l* − *Suc t choose l*) = (*k* − *t*) ∗ (*k* + *l* − *t choose l*)
by (*metis binomial-absorb-comp diff-Suc-eq-diff-pred diff-add-inverse2 diff-commute*)
with *Suc.IH* [*symmetric*] *Suc*(*2*) **show** *?case*
by (*simp add*: *field-simps flip*: *of-nat-mult of-nat-diff*)
qed *auto*
also have ... = (*real k* / (*k*+*l*)) ^*t* ∗ (∏ *i*<*t*. *1* − *real i* ∗ *real l* / (*real k* ∗ (*k*+*l*−*i*)))
proof −
have *1* − *i* ∗ *real l* / (*real k* ∗ (*k*+*l*−*i*)) = ((*k*−*i*)/(*k*+*l*−*i*)) ∗ ((*k*+*l*) / *k*)
if *i*<*t* **for** *i*
using *that* ‹*t* ≤ *k*› **by** (*simp add*: *divide-simps*) *argo*
then have ∗: (∏ *i*<*t*. *1* − *real i* ∗ *real l* / (*real k* ∗ (*k*+*l*−*i*))) = (∏ *i*<*t*. ((*k*−*i*)/(*k*+*l*−*i*)) ∗ ((*k*+*l*) / *k*))
by *auto*
show *?thesis*
unfolding ∗ *prod.distrib* **by** (*simp add*: *power-divide*)
qed
also have ... ≤ (*real k* / (*k*+*l*)) ^*t* ∗ *exp* (− (∑ *i*<*t*. *real i* ∗ *real l* / (*real k* ∗ (*k*+*l*))))
proof (*intro mult-left-mono*)
have *real i* ∗ *real l* / (*real k* ∗ *real* (*k*+*l*−*i*)) ≤ *1*
if *i* < *t* **for** *i*
using *that* ‹*t* ≤ *k*› **by** (*simp add*: *divide-simps mult-mono*)
moreover have *1* − *i* ∗ *l* / (*k* ∗ *real* (*k*+*l*−*i*)) ≤ *exp* (− (*i* ∗ *real l* / (*k* ∗ (*k* + *real l*)))) (**is** - ≤ *?R*)

```
    if i < t for i
   proof -
    have exp (- (i*l / (k * real (k+l-i)))) ≤ ?R
     using that ‹t ≤ k› by (simp add: frac-le-eq divide-le-0-iff mult-mono)
    with exp-minus-ge show ?thesis
     by (smt (verit, best))
   qed
   ultimately show (∏ i<t. 1 - i * real l / (k * real (k+l-i))) ≤ exp (-
(∑ i<t. i * real l / (k * real (k+l))))
    by (force simp: exp-sum simp flip: sum-negf intro!: prod-mono)
 qed auto
 finally have 1: (k+l-t choose l) * inverse (k+l choose l)
            ≤ (real k / (k+l))^t * exp (- (∑ i<t. i * γ / k))
   by (simp add: γ-def mult.commute)
 have **: γ * (t - 1)^2 / (2*k) ≤ (∑ i<t. i * γ / k)
 proof -
   have g: (∑ i<t. real i) = real (t*(t-1)) / 2
    by (induction t) (auto simp: algebra-simps eval-nat-numeral)
   have γ * (t-1)^2 / (2*k) ≤ real(t*(t-1)) / 2 * γ/k
      by (simp add: field-simps eval-nat-numeral divide-right-mono mult-mono
γ-def)
   also have ... = (∑ i<t. i * γ / k)
    unfolding g [symmetric] by (simp add: sum-distrib-right sum-divide-distrib)
   finally show ?thesis .
 qed
 have 0: 0 ≤ real (k + l choose l)
   by simp
 have *: (k+l-t choose l) ≤ (k / (k+l))^t * exp (- (∑ i<t. i * γ / k)) * (k+l
choose l)
   using order-trans [OF -  mult-right-mono [OF 1 0]]
   by (simp add: less-eq-real-def)
 also have ... ≤ (k / (k+l))^t * exp (- γ * (t-1)^2 / (2*k)) *(k+l choose l)
   using ** by (intro mult-mono) auto
 also have ... ≤ exp (- γ * (t-1)^2 / (2 * real k)) * (k / (k+l))^t * (k+l
choose l)
   by (simp add: mult-ac)
 finally show ?thesis
   using t by simp
qed
```

Statement borrowed from Bhavik; no o(k) function

```
corollary Far-9-6:
 fixes k l
 assumes t: 0<t t ≤ k
 defines γ ≡ l / (k + real l)
 shows exp (-1) * (1-γ) powr (- real t) * exp (γ * (real t)^2 / real(2*k)) *
(k-t+l choose l) ≤ (k+l choose l)
proof -
 have kkl: k / (k + real l) = 1 - γ k+l-t = k-t+l
```

using *t* **by** (*auto simp*: *γ-def divide-simps*)
have [*simp*]: *t* + *t* ≤ *Suc* (*t* ∗ *t*)
using *t*
by (*metis One-nat-def Suc-leI mult-2 mult-right-mono nle-le not-less-eq-eq numeral-2-eq-2 mult-1-right*)
have *0* ≤ *γ* *γ* < *1*
using *t* **by** (*auto simp*: *γ-def*)
then have *γ* ∗ (*real t* ∗ *2*) ≤ *γ* + *real k* ∗ *2*
using *t* **by** (*smt* (*verit*, *best*) *mult-less-cancel-right2 of-nat-0-less-iff of-nat-mono*)
then have ∗: *γ* ∗ *t*^*2* / (*2*∗*k*) − *1* ≤ *γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*)
using *t*
apply (*simp add*: *power2-eq-square pos-divide-le-eq divide-simps*)
apply (*simp add*: *algebra-simps*)
done
then have ∗: *exp* (−*1*) ∗ *exp* (*γ* ∗ *t*^*2* / (*2*∗*k*)) ≤ *exp* (*γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*))
by (*metis exp-add exp-le-cancel-iff uminus-add-conv-diff*)
have *1*: *exp* (*γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*)) ∗ (*k*+*l*−*t choose l*) ≤ (*k* / (*k*+*l*))^*t* ∗ (*k*+*l choose l*)
using *mult-right-mono* [*OF D2* [*OF t*], *of exp* (*γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*)) *l*] *t*
by (*simp add*: *γ-def exp-minus field-simps*)
have *2*: (*k* / (*k*+*l*)) *powr* (− *real t*) ∗ *exp* (*γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*)) ∗ (*k*+*l*−*t choose l*) ≤ (*k*+*l choose l*)
using *mult-right-mono* [*OF 1*, *of* (*1*−*γ*) *powr* (− *real t*)] *t*
by (*simp add*: *powr-minus γ-def powr-realpow mult-ac divide-simps*)
then have *3*: (*1*−*γ*) *powr* (− *real t*) ∗ *exp* (*γ* ∗ (*t*−*1*)^*2* / (*2*∗*k*)) ∗ (*k*−*t*+*l choose l*) ≤ (*k*+*l choose l*)
by (*simp add*: *kkl*)
show *?thesis*
apply (*rule order-trans* [*OF - 3*])
using ∗ *less-eq-real-def* **by** *fastforce*
qed

8.4 Lemma 9.3

definition *ok-fun-93g* ≡ *λγ k*. (*nat* ⌈*k powr* (*3*/*4*)⌉) ∗ *log 2 k* − (*ok-fun-71 γ k* + *ok-fun-94 k*) + *1*

lemma *ok-fun-93g*:
assumes *0* < *γ* *γ* < *1*
shows *ok-fun-93g γ* ∈ *o*(*real*)
proof −
have (*λk*. (*nat* ⌈*k powr* (*3*/*4*)⌉) ∗ *log 2 k*) ∈ *o*(*real*)
by *real-asymp*
then show *?thesis*
unfolding *ok-fun-93g-def*
by (*intro ok-fun-71* [*OF assms*] *ok-fun-94 sum-in-smallo const-smallo-real*)
qed

definition *ok-fun-93h* ≡ *λγ k*. (*2* / (*1*−*γ*)) ∗ *k powr* (*19*/*20*) ∗ (*ln γ* + *2* ∗ *ln k*)

```
                              + ok-fun-93g γ k ∗ ln 2

lemma ok-fun-93h:
  assumes 0 < γ γ < 1
  shows  ok-fun-93h γ ∈ o(real)
proof −
  have (λk. (2 / (1−γ)) ∗ k powr (19/20) ∗ (ln γ + 2 ∗ ln k)) ∈ o(real)
    by real-asymp
  then show ?thesis
   unfolding ok-fun-93h-def by (metis (mono-tags) ok-fun-93g assms sum-in-smallo(1)
cmult-in-smallo-iff ′)
qed

lemma ok-fun-93h-uniform:
  assumes μ01: 0<μ0 μ1<1
  assumes e>0
  shows  ∀∞k. ∀μ. μ ∈ {μ0..μ1} ⟶ |ok-fun-93h μ k| / k ≤ e
proof −
  define f where f ≡ λk. ok-fun-73 k + ok-fun-74 k + ok-fun-76 k + ok-fun-94 k
  define g where g ≡ λμ k. 2 ∗ real k powr (19/20) ∗ (ln μ + 2 ∗ ln k) / (1−μ)
  have g: ∀∞k. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ |g μ k| / k ≤ e if e>0 for e
  proof (intro eventually-all-geI1 [where L = nat⌈1 / μ0⌉])
    show ∀∞k. |g μ1 k| / real k ≤ e
      using assms that unfolding g-def by real-asymp
  next
    fix k μ
    assume le-e: |g μ1 k| / k ≤ e and μ: μ0 ≤ μ μ ≤ μ1 and k: nat ⌈1/μ0⌉ ≤ k
    then have k>0
      using assms gr0I by force
    have ln-k: ln k ≥ ln (1/μ0)
      using k ‹0<μ0› ln-mono by fastforce
    with μ μ01
    have |ln μ + 2 ∗ ln (real k)| ≤ |ln μ1 + 2 ∗ ln (real k)|
      by (smt (verit) ln-div ln-mono ln-one)
    with μ k ‹μ1 < 1›
    have |g μ k| ≤ |g μ1 k|
      by (simp add: g-def abs-mult frac-le mult-mono)
    then show |g μ k| / real k ≤ e
      by (smt (verit, best) divide-right-mono le-e of-nat-less-0-iff)
  qed
  have eq93: ok-fun-93h μ k = g μ k +
         ⌈k powr (3/4)⌉ ∗ ln k − ((ok-fun-72 μ k + f k) − 1) ∗ ln 2 for μ k
     by (simp add: ok-fun-93h-def g-def ok-fun-71-def ok-fun-93g-def f-def log-def
field-simps)

  have ln2: ln 2 ≥ (0::real)
    by simp
  have le93: |ok-fun-93h μ k|
     ≤ |g μ k| + |⌈k powr (3/4)⌉ ∗ ln k| + (|ok-fun-72 μ k| + |f k| + 1) ∗ ln 2
```

```
for μ k
    unfolding eq93
   by (smt (verit, best) mult.commute ln-gt-zero-iff mult-le-cancel-left-pos mult-minus-left)
  define e5 where e5 ≡ e/5
  have e5 > 0
    by (simp add: ‹e>0› e5-def)
  then have A: ∀∞k. ∀μ. μ ∈ {μ0..μ1} ⟶ |g μ k| / k ≤ e5
    using g by simp
  have B: ∀∞k. |⌈k powr (3/4)⌉ ∗ ln k| / k ≤ e5
    using ‹0 < e5› by real-asymp
  have C: ∀∞k. ∀μ. μ ∈ {μ0..μ1} ⟶ |ok-fun-72 μ k| ∗ ln 2 / k ≤ e5
    using ln2 assms ok-fun-72-uniform[OF μ01, of e5 / ln 2] ‹e5 > 0›
    by (simp add: divide-simps)
  have f ∈ o(real)
    by (simp add: f-def ok-fun-73 ok-fun-74 ok-fun-76 ok-fun-94 sum-in-smallo(1))
  then have D: ∀∞k. |f k| ∗ ln 2 / k ≤ e5
    using ‹e5 > 0› ln2
   by (force simp: smallo-def field-simps eventually-at-top-dense dest!: spec [where
x=e5 / ln 2])
  have E: ∀∞k.  ln 2 / k ≤ e5
    using ‹e5 > 0› ln2 by real-asymp
  have ∀∞k. ∀μ. μ ∈ {μ0..μ1} ⟶ |ok-fun-93h μ k| / real k ≤ e5+e5+e5+e5+e5
    using A B C D E
    apply eventually-elim
    by (fastforce simp: add-divide-distrib distrib-right
         intro!: order-trans [OF  divide-right-mono [OF le93]])
  then show ?thesis
    by (simp add: e5-def)
qed

context P0-min
begin

definition Big-Far-9-3 ≡
   λμ l. Big-ZZ-8-5 μ l ∧ Big-X-7-1 μ l ∧ Big-Y-6-2 μ l ∧ Big-Red-5-3 μ l
      ∧ (∀k≥l. p0-min − 3 ∗ eps k > 1/k ∧ k≥2
           ∧ |ok-fun-93h μ k / (μ ∗ (1 + 1 / (exp 1 ∗ (1−μ))))| / k ≤ 0.667 −
2/3)

lemma Big-Far-9-3:
  assumes 0<μ0 μ0≤μ1 μ1<1
  shows ∀∞l. ∀μ. μ ∈ {μ0..μ1} ⟶ Big-Far-9-3 μ l
proof −
  define d where d ≡ λμ::real. μ ∗ (1 + 1 / (exp 1 ∗ (1−μ)))
  have d μ0 > 0
    using assms by (auto simp: d-def divide-simps add-pos-pos)
  then have dgt: d μ ≥ d μ0 if μ ∈ {μ0..μ1} for μ
    using that assms by (auto simp: d-def frac-le mult-mono)
```

define *e*::*real* **where** *e* ≡ *0.667* − *2/3*
have *e*>*0*
by (*simp add*: *e-def*)
have ∗: ∀∞*l*. ∀ *μ*. *μ* ∈ {*μ0*..*μ1*} ⟶ (∀ *k*≥*l*. |*ok-fun-93h* *μ* *k* / *d* *μ*| / *k* ≤ *e*)
proof −
have ∀∞*l*. ∀ *k*≥*l*. (∀ *μ*. *μ* ∈ {*μ0*..*μ1*} ⟶ |*ok-fun-93h* *μ* *k*| / *k* ≤ *d* *μ0* ∗ *e*)
using *mult-pos-pos*[*OF* ‹*d* *μ0* > *0*› ‹*e*>*0*›] *assms*
using *ok-fun-93h-uniform* *eventually-all-ge-at-top*
by *blast*
then show *?thesis*
apply *eventually-elim*
using *dgt* ‹*0* < *d* *μ0*› ‹*0* < *e*›
by (*auto simp*: *mult-ac divide-simps mult-less-0-iff zero-less-mult-iff split*: *if-split-asm*)
(*smt* (*verit*) *mult-less-cancel-left nat-neq-iff of-nat-0-le-iff*)
qed
with *p0-min* **show** *?thesis*
unfolding *Big-Far-9-3-def eps-def d-def e-def*
using *assms Big-ZZ-8-5 Big-X-7-1 Big-Y-6-2 Big-Red-5-3*
apply (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)
apply (*intro conjI strip eventually-all-ge-at-top*; *real-asymp*)
done
qed

end

lemma (*λk*. (*nat* ⌈*real k powr* (*3/4*)⌉) ∗ *log 2 k*) ∈ *o*(*real*)
by *real-asymp*

lemma *RN34-le-2powr-ok*:
fixes *l k*::*nat*
assumes *l* ≤ *k* *0*<*k*
defines *l34* ≡ *nat* ⌈*real l powr* (*3/4*)⌉
shows *RN k l34* ≤ *2 powr* (⌈*k powr* (*3/4*)⌉ ∗ *log 2 k*)
proof −
have §: ⌈*l powr* (*3/4*)⌉ ≤ ⌈*k powr* (*3/4*)⌉
by (*simp add*: *assms*(*1*) *ceiling-mono powr-mono2*)
have *RN k l34* ≤ *k powr* (*l34*−*1*)
— Bhavik's off-diagonal Ramsey upper bound; can't use $(2::'a)^{k + l34}$
using *RN-le-argpower'* ‹*k*>*0*› *powr-realpow* **by** *auto*
also have ... ≤ *k powr l34*
using ‹*k*>*0*› *powr-mono* **by** *force*
also have ... ≤ *2 powr* (*l34* ∗ *log 2 k*)
by (*smt* (*verit*, *best*) *mult.commute* ‹*k*>*0*› *of-nat-0-less-iff powr-log-cancel powr-powr*)
also have ... ≤ *2 powr* (⌈*real k powr* (*3/4*)⌉ ∗ *log 2 k*)
unfolding *l34-def*
proof (*intro powr-mono powr-mono2 mult-mono ceiling-mono of-nat-mono nat-mono* ‹*l* ≤ *k*›)

```
    show 0 ≤ real-of-int ⌈k powr (3/4)⌉
      by (meson le-of-int-ceiling order.trans powr-ge-zero)
  qed (use assms § in auto)
  finally show ?thesis .
qed
```

Here n really refers to the cardinality of V, so actually nV

```
lemma (in Book') Far-9-3:
  defines δ ≡ min (1/200) (γ/20)
  defines R ≡ Step-class {red-step}
  defines t ≡ card R
  assumes γ15: γ ≤ 1/5 and p0: p0 ≥ 1/4
    and nge: n ≥ exp (−δ ∗ real k) ∗ (k+l choose l)
    and X0ge: card X0 ≥ n/2
        — Because n / 2 ≤ real (card X0) makes the proof harder
  assumes big: Big-Far-9-3 γ l
  shows t ≥ 2∗k / 3
proof −
  define S where S ≡ Step-class {dboost-step}
  have k≥2 and big85: Big-ZZ-8-5 γ l and big71: Big-X-7-1 γ l
    and big62: Big-Y-6-2 γ l and big53: Big-Red-5-3 γ l
    using big l-le-k by (auto simp: Big-Far-9-3-def)
  define l34 where l34 ≡ nat ⌈real l powr (3/4)⌉
  have l34 > 0
    using l34-def ln0 by fastforce
  have γ01: 0 < γ γ < 1
    using ln0 l-le-k by (auto simp: γ-def)
  then have bigbeta01: 0 < bigbeta bigbeta < 1
    using big53 assms bigbeta-gt0 bigbeta-less1 by (auto simp: bigbeta-def)
  have one-minus: 1−γ = real k / (real k + real l)
    using ln0 by (simp add: γ-def divide-simps)
  have t < k
    using red-step-limit by (auto simp: R-def t-def)
  have f: 2 powr ok-fun-94 k ∗ γ powr (− real l) ∗ (1−γ) powr (− real k)
        ≤ k+l choose l
    unfolding γ-def using fact-9-4 l-le-k ln0 by blast
  have powr-combine-right: x powr a ∗ (x powr b ∗ y) = x powr (a+b) ∗ y for x
y a b::real
    by (simp add: powr-add)
  have (2 powr ok-fun-71 γ k ∗ 2 powr ok-fun-94 k) ∗ (bigbeta/γ) ^ card S ∗ (exp
(−δ∗k) ∗ (1−γ) powr (− real k + t) / 2)
      ≤ 2 powr ok-fun-71 γ k ∗ γ^l ∗ (1−γ) ^ t ∗ (bigbeta/γ) ^ card S ∗ (exp
(−δ∗k) ∗ (k+l choose l) / 2)
    using γ01 ‹0<bigbeta› mult-right-mono [OF f, of 2 powr ok-fun-71 γ k ∗ γ^l
∗ (1−γ) ^ t ∗ (bigbeta/γ) ^ card S ∗ (exp (−δ∗k)) / 2]
   by (simp add: mult-ac zero-le-mult-iff powr-minus powr-diff divide-simps powr-realpow)
  also have ... ≤ 2 powr ok-fun-71 γ k ∗ γ^l ∗ (1−γ) ^ t ∗ (bigbeta/γ) ^ card
S ∗ card X0
  proof (intro mult-left-mono order-refl)
```

```
    show exp (−δ ∗ k) ∗ real (k+l choose l) / 2 ≤ real (card X0)
      using X0ge nge by force
    show 0 ≤ 2 powr ok-fun-71 γ k ∗ γ ^ l ∗ (1−γ) ^ t ∗ (bigbeta/γ) ^ card S
      using γ01 bigbeta-ge0 by (force simp: bigbeta-def)
  qed
  also have ... ≤ card (Xseq halted-point)
    unfolding R-def S-def t-def using big
    by (intro X-7-1) (auto simp: Big-Far-9-3-def)
  also have ... ≤ RN k l34
  proof −
    have p0 − 3 ∗ ε > 1/k and pseq halted-point ≥ p0 − 3 ∗ ε
      using l-le-k big p0-ge Y-6-2-halted by (auto simp: Big-Far-9-3-def γ-def)
    then show ?thesis
      using halted-point-halted γ01
        by (fastforce simp: step-terminating-iff termination-condition-def pseq-def
l34-def)
  qed
  also have ... ≤ 2 powr (⌈k powr (3/4)⌉ ∗ log 2 k)
    using RN34-le-2powr-ok l34-def l-le-k ln0 by blast
  finally have 2 powr (ok-fun-71 γ k + ok-fun-94 k) ∗ (bigbeta/γ) ^ card S
              ∗ exp (−δ∗k) ∗ (1−γ) powr (− real k + t) / 2
             ≤ 2 powr (⌈k powr (3/4)⌉ ∗ log 2 k)
    by (simp add: powr-add)
  then have le-2-powr-g: exp (−δ∗k) ∗ (1−γ) powr (− real k + t) ∗ (bigbeta/γ)
^ card S
            ≤ 2 powr ok-fun-93g γ k
    using ‹k≥2› by (simp add: ok-fun-93g-def field-simps powr-add powr-diff flip:
powr-realpow)

  let ?ξ = bigbeta ∗ t / (1−γ) + (2 / (1−γ)) ∗ k powr (19/20)
  have bigbeta-le: bigbeta ≤ γ and bigbeta-ge: bigbeta ≥ 1 / (real k)²
    using bigbeta-def γ01 big53 bigbeta-le bigbeta-ge-square by blast+

  define φ where φ ≡ λu. (u / (1−γ)) ∗ ln (γ/u)  — finding the maximum via
derivatives
  have ln-eq: ln (γ / (γ / exp 1)) / (1−γ) = 1/(1−γ)
    using γ01 by simp
  have φ: φ (γ / exp 1) ≥ φ bigbeta
  proof (cases γ / exp 1 ≤ bigbeta)    — Could perhaps avoid case analysis via
2nd derivatives
    case True
    show ?thesis
    proof (intro DERIV-nonpos-imp-nonincreasing [where f = φ])
      fix x
      assume x: γ / exp 1 ≤ x x ≤ bigbeta
      with γ01 have x>0
        by (smt (verit, best) divide-pos-pos exp-gt-zero)
      with γ01 x have ln (γ/x) / (1−γ) − 1 / (1−γ) ≤ 0
        by (smt (verit, ccfv-SIG) divide-pos-pos exp-gt-zero frac-le ln-eq ln-mono)
```

```
      with x ‹x>0› γ01 show ∃D. (φ has-real-derivative D) (at x) ∧ D ≤ 0
        unfolding φ-def by (intro exI conjI derivative-eq-intros | force)+
    qed (simp add: True)
  next
    case False
    show ?thesis
    proof (intro DERIV-nonneg-imp-nondecreasing [where f = φ])
      fix x
      assume x: bigbeta ≤ x x ≤ γ / exp 1
      with bigbeta01 γ01 have x>0 by linarith
      with γ01 x have ln (γ/x) / (1−γ) − 1 / (1−γ) ≥ 0
        by (smt (verit, best) frac-le ln-eq ln-mono zero-less-divide-iff)
      with x ‹x>0› γ01 show ∃D. (φ has-real-derivative D) (at x) ∧ D ≥ 0
        unfolding φ-def
        by (intro exI conjI derivative-eq-intros | force)+
    qed (use False in force)
  qed

  define c where c ≡ λx::real. 1 + 1 / (exp 1 ∗ (1−x))
  have mono-c: mono-on {0<..<1} c
    by (auto simp: monotone-on-def c-def field-simps)
  have cgt0: c x > 0 if x<1 for x
    using that by (simp add: add-pos-nonneg c-def)

  have card 𝒮 ≤ bigbeta ∗ t / (1−bigbeta) + (2 / (1−γ)) ∗ k powr (19/20)
    using ZZ-8-5 [OF big85] by (auto simp: ℛ-def 𝒮-def t-def)
  also have . . . ≤ ?ξ
    using bigbeta-le by (simp add: γ01 bigbeta-ge0 frac-le)
  finally have card 𝒮 ≤ ?ξ .
  with bigbeta-le bigbeta01 have ?ξ ∗ ln (bigbeta/γ) ≤ card 𝒮 ∗ ln (bigbeta/γ)
    by (simp add: mult-right-mono-neg)
  then have −?ξ ∗ ln (γ/bigbeta) ≤ card 𝒮 ∗ ln (bigbeta/γ)
    using bigbeta01 γ01 by (smt (verit) ln-div minus-mult-minus)
  then have γ ∗ (real k − t) − δ∗k − ?ξ ∗ ln (γ/bigbeta) ≤ γ ∗ (real k − t) −
δ∗k + card 𝒮 ∗ ln (bigbeta/γ)
    by linarith
  also have . . . ≤ (t − real k) ∗ ln (1−γ) − δ∗k + card 𝒮 ∗ ln (bigbeta/γ)
    using ‹t < k› γ01 mult-right-mono [OF ln-add-one-self-le-self2 [of −γ], of real
k − t]
    by (simp add: algebra-simps)
  also have . . . = ln (exp (−δ∗k) ∗ (1−γ) powr (− real k + t) ∗ (bigbeta/γ) ^
card 𝒮)
    using γ01 bigbeta01 by (simp add: ln-mult ln-div ln-realpow)
  also have . . . ≤ ln (2 powr ok-fun-93g γ k)
    using le-2-powr-g γ01 bigbeta01 by (simp del: ln-powr)
  also have . . . = ok-fun-93g γ k ∗ ln 2
    by auto
  finally have γ ∗ (real k − t) − δ∗k − ?ξ ∗ ln (γ/bigbeta) ≤ ok-fun-93g γ k ∗
ln 2 .
```

then have *γ* ∗ (*real k* − *t*) ≤ *?ξ* ∗ *ln* (*γ*/*bigbeta*) + *δ*∗*k* + *ok-fun-93g γ k* ∗ *ln 2*
  by *simp*
also have . . . ≤ (*bigbeta* ∗ *t* / (*1*−*γ*)) ∗ *ln* (*γ*/*bigbeta*) + *δ*∗*k* + *ok-fun-93h γ k*
proof −
  have *γ*/*bigbeta* ≤ *γ* ∗ (*real k*)^2
    using *kn0 bigbeta-le bigbeta-ge* ‹*bigbeta*>*0*› **by** (*simp add*: *field-simps*)
  then have *X*: *ln* (*γ*/*bigbeta*) ≤ *ln γ* + *2* ∗ *ln k*
    using ‹*bigbeta*>*0*› ‹*γ*>*0*› *kn0*
      by (*metis ln-mult-pos ln-realpow of-nat-numeral of-nat-zero-less-power-iff divide-pos-pos ln-mono*)
  show *?thesis*
    using *mult-right-mono* [*OF X*, *of 2* ∗ *k powr* (*19*/*20*) / (*1*−*γ*)] ‹*γ*<*1*›
    by (*simp add*: *ok-fun-93h-def algebra-simps*)
qed
also have . . . ≤ ((*γ* / *exp 1*) ∗ *t* / (*1*−*γ*)) + *δ*∗*k* + *ok-fun-93h γ k*
  using *γ01 mult-right-mono* [*OF φ*, *of t*] **by** (*simp add*: *φ-def mult-ac*)
finally have *γ* ∗ (*real k* − *t*) ≤ ((*γ* / *exp 1*) ∗ *t* / (*1*−*γ*)) + *δ*∗*k* + *ok-fun-93h γ k* **.**
then have (*γ*−*δ*) ∗ *k* − *ok-fun-93h γ k* ≤ *t* ∗ *γ* ∗ *c γ*
  by (*simp add*: *c-def algebra-simps*)
then have ((*γ*−*δ*) ∗ *k* − *ok-fun-93h γ k*) / (*γ* ∗ *c γ*) ≤ *t*
  using *γ01 cgt0* **by** (*simp add*: *pos-divide-le-eq*)
then have ∗: *t* ≥ (*1*−*δ* / *γ*) ∗ *inverse* (*c γ*) ∗ *k* − *ok-fun-93h γ k* / (*γ* ∗ *c γ*)
  using *γ01 cgt0*[*of γ*] **by** (*simp add*: *divide-simps*)
define *f47* **where** *f47* ≡ *λx*. (*1* − *1*/(*200*∗*x*)) ∗ *inverse* (*c x*)
have *concave-on* {*1*/*10*..*1*/*5*} *f47*
  unfolding *f47-def*
proof (*intro concave-on-mul*)
  show *concave-on* {*1*/*10*..*1*/*5*} (*λx*. *1* − *1*/(*200*∗*x*))
  proof (*intro f''-le0-imp-concave*)
    fix *x*::*real*
    assume *x* ∈ {*1*/*10*..*1*/*5*}
    then have *x01*: *0*<*x x*<*1* **by** *auto*
    show ((*λx*. (*1* − *1*/(*200*∗*x*))) *has-real-derivative 1*/(*200*∗*x*^*2*)) (*at x*)
      using *x01* **by** (*intro derivative-eq-intros* | *force simp*: *eval-nat-numeral*)+
    show ((*λx*. *1*/(*200*∗*x*^*2*)) *has-real-derivative* −*1*/(*100*∗*x*^*3*)) (*at x*)
      using *x01* **by** (*intro derivative-eq-intros* | *force simp*: *eval-nat-numeral*)+
    show −*1*/(*100*∗*x*^*3*) ≤ *0*
      using *x01* **by** (*simp add*: *divide-simps*)
  qed *auto*
  show *concave-on* {*1*/*10*..*1*/*5*} (*λx*. *inverse* (*c x*))
  proof (*intro f''-le0-imp-concave*)
    fix *x*::*real*
    assume *x* ∈ {*1*/*10*..*1*/*5*}
    then have *x01*: *0*<*x x*<*1* **by** *auto*
    have *swap*: *u* ∗ (*x*−*1*) = (−*u*) ∗ (*1*−*x*) **for** *u*
      by (*metis minus-diff-eq minus-mult-commute*)
    have §: *exp 1* ∗ (*x* − *1*) < *0*
      using *x01* **by** (*meson exp-gt-zero less-iff-diff-less-0 mult-less-0-iff*)

```
      then have non0: 1 + 1 / (exp 1 * (1−x)) ≠ 0
        using x01 by (smt (verit) exp-gt-zero mult-pos-pos zero-less-divide-iff)
      let ?f1 = λx. −exp 1 /(− 1 + exp 1 * (− 1 + x))^2
      let ?f2 = λx. 2*exp(1)^2/(−1 + exp(1)*(−1 + x))^3
      show ((λx. inverse (c x)) has-real-derivative ?f1 x) (at x)
        unfolding c-def power2-eq-square
        using x01 § non0
        apply (intro exI conjI derivative-eq-intros | force)+
        apply (simp add: divide-simps square-eq-iff swap)
        done
      show (?f1 has-real-derivative ?f2 x) (at x)
        using x01 §
        by (intro derivative-eq-intros | force simp: divide-simps eval-nat-numeral)+
      show ?f2 (x::real) ≤ 0
        using x01 § by (simp add: divide-simps)
    qed auto
    show mono-on {(1::real)/10..1/5} (λx. 1 − 1 / (200 * x))
      by (auto simp: monotone-on-def frac-le)
    show monotone-on {1/10..1/5} (≤) (λx y. y ≤ x) (λx. inverse (c x))
      using mono-c cgt0 by (auto simp: monotone-on-def divide-simps)
  qed (auto simp: c-def)
  moreover have f47(1/10) > 0.667
    unfolding f47-def c-def by (approximation 15)
  moreover have f47(1/5) > 0.667
    unfolding f47-def c-def by (approximation 15)
  ultimately have 47: f47 x > 0.667 if x ∈ {1/10..1/5} for x
    using concave-on-ge-min that by fastforce

  define f48 where f48 ≡ λx. (1 − 1/20) * inverse (c x)
  have 48: f48 x > 0.667 if x ∈ {0<..<1/10} for x
  proof −
    have (0.667::real) < (1 − 1/20) * inverse(c(1/10))
      unfolding c-def by (approximation 15)
    also have ... ≤ f48 x
      using that unfolding f48-def c-def
     by (intro mult-mono le-imp-inverse-le add-mono divide-left-mono) (auto simp:
add-pos-pos)
    finally show ?thesis .
  qed
  define e::real where e ≡ 0.667 − 2/3
  have BIGH: abs (ok-fun-93h γ k / (γ * c γ)) / k ≤ e
   using big l-le-k unfolding Big-Far-9-3-def all-imp-conj-distrib e-def [symmetric]
c-def
    by auto
  consider γ ∈ {0<..<1/10} | γ ∈ {1/10..1/5}
    using δ-def ‹γ ≤ 1/5› γ01 by fastforce
  then show ?thesis
  proof cases
    case 1
```

```
    then have δγ: δ / γ = 1/20
      by (auto simp: δ-def)
    have (2/3::real) ≤ f48 γ − e
      using 48[OF 1] e-def by force
    also have ... ≤ (1−δ / γ) ∗ inverse (c γ) − ok-fun-93h γ k / (γ ∗ c γ) / k
      unfolding f48-def δγ using BIGH
      by (smt (verit, best) divide-nonneg-nonneg of-nat-0-le-iff zero-less-divide-iff)
    finally
    have A: 2/3 ≤ (1−δ / γ) ∗ inverse (c γ) − ok-fun-93h γ k / (γ ∗ c γ) / k .
    have real (2 ∗ k) / 3 ≤ (1 − δ / γ) ∗ inverse (c γ) ∗ k − ok-fun-93h γ k /
(γ ∗ c γ)
      using mult-left-mono [OF A, of k] cgt0 [of γ] γ01 kn0
      by (simp add: divide-simps mult-ac)
    with ∗ show ?thesis
      by linarith
  next
    case 2
    then have δγ: δ / γ = 1/(200∗γ)
      by (auto simp: δ-def)
    have (2/3::real) ≤ f47 γ − e
      using 47[OF 2] e-def by force
    also have ... ≤ (1 − δ / γ) ∗ inverse (c γ) − ok-fun-93h γ k / (γ ∗ c γ) / k
      unfolding f47-def δγ using BIGH
      by (smt (verit, best) divide-right-mono of-nat-0-le-iff)
    finally
    have 2/3 ≤ (1 − δ / γ) ∗ inverse (c γ) − ok-fun-93h γ k / (γ ∗ c γ) / k .
    from mult-left-mono [OF this, of k] cgt0 [of γ] γ01 kn0
    have real (2 ∗ k) / 3 ≤ (1 − δ / γ) ∗ inverse (c γ) ∗ k − ok-fun-93h γ k /
(γ ∗ c γ)
      by (simp add: divide-simps mult-ac)
    with ∗ show ?thesis
      by linarith
  qed
qed
```

8.5 Lemma 9.5

```
context P0-min
begin
```

Again stolen from Bhavik: cannot allow a dependence on γ

```
definition ok-fun-95a ≡ λk. ok-fun-61 k − (2 + 4 ∗ k powr (19/20))

definition ok-fun-95b ≡ λk. ln 2 ∗ ok-fun-95a k − 1

lemma ok-fun-95a: ok-fun-95a ∈ o(real)
proof −
  have (λk. 2 + 4 ∗ k powr (19/20)) ∈ o(real)
    by real-asymp
  then show ?thesis
```

unfolding *ok-fun-95a-def* **using** *ok-fun-61 sum-in-smallo* **by** *blast*
qed

lemma *ok-fun-95b*: *ok-fun-95b* ∈ *o*(*real*)
using *ok-fun-95a* **by** (*auto simp*: *ok-fun-95b-def sum-in-smallo const-smallo-real*)

definition *Big-Far-9-5* ≡ λμ *l*. *Big-Red-5-3* μ *l* ∧ *Big-Y-6-1* μ *l* ∧ *Big-ZZ-8-5* μ *l*

lemma *Big-Far-9-5*:
assumes *0*<μ*0* μ*1*<*1*
shows ∀$^{\infty}$*l*. ∀μ. μ*0* ≤ μ ∧ μ ≤ μ*1* ⟶ *Big-Far-9-5* μ *l*
using *assms Big-Red-5-3 Big-Y-6-1 Big-ZZ-8-5*
unfolding *Big-Far-9-5-def eps-def*
by (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)

end

Y0 is an additional assumption found in Bhavik's version. (He had a couple of others). The first $o(k)$ function adjusts for the error in $n/2$

lemma (**in** *Book'*) *Far-9-5*:
fixes δ η::*real*
defines 𝓡 ≡ *Step-class* {*red-step*}
defines *t* ≡ *card* 𝓡
assumes *nV*: *real nV* ≥ *exp* (−δ ∗ *k*) ∗ (*k*+*l choose l*) **and** *Y0*: *card Y0* ≥ *nV div 2*
assumes *p0*: *1*/*2* ≤ *1*−γ−η *1*−γ−η ≤ *p0* **and** *0*≤η
assumes *big*: *Big-Far-9-5* γ *l*
shows *card* (*Yseq halted-point*) ≥
exp (−δ ∗ *k* + *ok-fun-95b k*) ∗ (*1*−γ−η) *powr* (γ∗*t* / (*1*−γ)) ∗ ((*1*−γ−η)/(*1*−γ)) ^*t*
∗ *exp* (γ ∗ $(real\ t)^2$ / (*2*∗*k*)) ∗ (*k*−*t*+*l choose l*)   (**is** - ≥ *?rhs*)
proof −
define 𝓢 **where** 𝓢 ≡ *Step-class* {*dboost-step*}
define *s* **where** *s* ≡ *card* 𝓢
have γ*01*: *0* < γ γ < *1*
using *ln0 l-le-k* **by** (*auto simp*: γ*-def*)
have *big85*: *Big-ZZ-8-5* γ *l* **and** *big61*: *Big-Y-6-1* γ *l* **and** *big53*: *Big-Red-5-3* γ *l*
using *big* **by** (*auto simp*: *Big-Far-9-5-def*)
have *bigbeta* ≤ γ
using *bigbeta-def* γ*01 big53 bigbeta-le* **by** *blast*
have *85*: *s* ≤ (*bigbeta* / (*1*−*bigbeta*)) ∗ *t* + (*2* / (*1*−γ)) ∗ *k powr* (*19*/*20*)
unfolding *s-def t-def* 𝓡*-def* 𝓢*-def* **using** *ZZ-8-5* γ*01 big85* **by** *blast*
also have ... ≤ (γ / (*1*−γ)) ∗ *t* + (*2* / (*1*−γ)) ∗ *k powr* (*19*/*20*)
using γ*01* ‹*bigbeta* ≤ γ› **by** (*intro add-mono mult-right-mono frac-le*) *auto*
finally have *D85*: *s* ≤ γ∗*t* / (*1*−γ) + (*2* / (*1*−γ)) ∗ *k powr* (*19*/*20*)
by *auto*
have *t*<*k*

unfolding *t-def* $\mathcal{R}$*-def* **using** γ*01 red-step-limit* **by** *blast*
have *st*: *card* (*Step-class* {*red-step*,*dboost-step*}) = *t* + *s*
using γ*01*
by (*simp add*: *s-def t-def* $\mathcal{R}$*-def* $\mathcal{S}$*-def Step-class-insert-NO-MATCH card-Un-disjnt disjnt-Step-class*)
then have *61*: *2 powr* (*ok-fun-61 k*) ∗ *p0* ^ (*t*+*s*) ∗ *card Y0* ≤ *card* (*Yseq halted-point*)
using *Y-6-1*[*OF big61*] *card-XY0* γ*01* **by** (*simp add*: *divide-simps*)
have (*1*−γ−η) *powr* (*t* + γ∗*t* / (*1*−γ)) ∗ *nV* ≤ (*1*−γ−η) *powr* (*t*+*s* − *4* ∗ *k powr* (*19*/*20*)) ∗ (*4* ∗ *card Y0*)
proof (*intro mult-mono*)
show (*1*−γ−η) *powr* (*t* + γ∗*t* / (*1*−γ)) ≤ (*1*−γ−η) *powr* (*t*+*s* − *4* ∗ *k powr* (*19*/*20*))
proof (*intro powr-mono'*)
have γ ≤ *1*/*2*
using ‹*0*≤η› *p0* **by** *linarith*
then have *22*: *1* / (*1* − γ) ≤ *2*
using *divide-le-eq-1* **by** *fastforce*
show *real* (*t* + *s*) − *4* ∗ *real k powr* (*19* / *20*) ≤ *real t* + γ ∗ *real t* / (*1* − γ)
using *mult-left-mono* [*OF 22*, *of 2* ∗ *real k powr* (*19* / *20*)] *D85*
by (*simp add*: *algebra-simps*)
next
show *0* ≤ *1* − γ − η *1* − γ − η ≤ *1*
using *assms* γ*01* **by** *linarith*+
qed
have *nV* ≥ *2*
by (*metis nontriv wellformed two-edges card-mono ex-in-conv finV*)
then have *nV* ≤ *4* ∗ (*nV div 2*) **by** *linarith*
also have ... ≤ *4* ∗ *card Y0*
using *Y0 mult-le-mono2* **by** *presburger*
finally show *real nV* ≤ *real* (*4* ∗ *card Y0*)
by *force*
qed (*use Y0* **in** *auto*)
also have ... ≤ (*1*−γ−η) *powr* (*t*+*s*) / (*1*−γ−η) *powr* (*4* ∗ *k powr* (*19*/*20*)) ∗ (*4* ∗ *card Y0*)
by (*simp add*: *divide-powr-uminus powr-diff*)
also have ... ≤ (*1*−γ−η) *powr* (*t*+*s*) / (*1*/*2*) *powr* (*4* ∗ *k powr* (*19*/*20*)) ∗ (*4* ∗ *card Y0*)
proof (*intro mult-mono divide-left-mono*)
show (*1*/*2*) *powr* (*4* ∗ *k powr* (*19*/*20*)) ≤ (*1*−γ−η) *powr* (*4* ∗ *k powr* (*19*/*20*))
using γ*01 p0* ‹*0*≤η› **by** (*intro powr-mono-both'*) *auto*
qed (*use p0* **in** *auto*)
also have ... ≤ *p0 powr* (*t*+*s*) / (*1*/*2*) *powr* (*4* ∗ *k powr* (*19*/*20*)) ∗ (*4* ∗ *card Y0*)
using *p0 powr-mono2* **by** (*intro mult-mono divide-right-mono*) *auto*
also have ... = (*2 powr* (*2* + *4* ∗ *k powr* (*19*/*20*))) ∗ *p0* ^ (*t*+*s*) ∗ *card Y0*
using *p0-01* **by** (*simp add*: *powr-divide powr-add power-add powr-realpow*)
finally have *2 powr* (*ok-fun-95a k*) ∗ (*1*−γ−η) *powr* (*t* + γ∗*t* / (*1*−γ)) ∗ *nV*

≤ 2 powr (ok-fun-61 k) ∗ p0 ^ (t+s) ∗ card Y0
    by (simp add: ok-fun-95a-def powr-diff field-simps)
  with 61 **have** ∗: card (Yseq halted-point) ≥ 2 powr (ok-fun-95a k) ∗ (1−γ−η) powr (t + γ∗t / (1−γ)) ∗ nV
    by linarith

  have F: exp (ok-fun-95b k) = 2 powr ok-fun-95a k ∗ exp (− 1)
    by (simp add: ok-fun-95b-def exp-diff exp-minus powr-def field-simps)
  have ?rhs
   ≤ exp (−δ ∗ k) ∗ 2 powr (ok-fun-95a k) ∗ exp (−1) ∗ (1−γ−η) powr (γ∗t / (1−γ))
        ∗ (((1−γ−η)/(1−γ)) ^t ∗ exp (γ ∗ (real t)^2 / real(2∗k)) ∗ (k−t+l choose l))
    unfolding exp-add F **by** simp
  also have . . . ≤ exp (−δ ∗ k) ∗ 2 powr (ok-fun-95a k) ∗ (1−γ−η) powr (γ∗t / (1−γ))
          ∗ (exp (−1) ∗ ((1−γ−η)/(1−γ)) ^t ∗ exp (γ ∗ (real t)^2 / real(2∗k)) ∗ (k−t+l choose l))
    by (simp add: mult.assoc)
  also have . . . ≤ 2 powr (ok-fun-95a k) ∗ (1−γ−η) powr (t + γ∗t / (1−γ)) ∗ exp (−δ ∗ k)
             ∗ (exp (−1) ∗ (1−γ) powr (− real t) ∗ exp (γ ∗ (real t)^2 / real(2∗k)) ∗ (k−t+l choose l))
    using p0 γ01
      unfolding powr-add powr-minus **by** (simp add: mult-ac divide-simps flip: powr-realpow)
  also have . . . ≤ 2 powr (ok-fun-95a k) ∗ (1−γ−η) powr (t + γ∗t / (1−γ)) ∗ exp (−δ ∗ k) ∗ (k+l choose l)
  proof (cases t=0)
    case False
    then show ?thesis
      unfolding γ-def **using** ‹t<k› **by** (intro mult-mono order-refl Far-9-6) auto
  qed auto
  also have . . . ≤ 2 powr (ok-fun-95a k) ∗ (1−γ−η) powr (t + γ∗t / (1−γ)) ∗ nV
    using nV mult-left-mono **by** fastforce
  also have . . . ≤ card (Yseq halted-point)
    by (rule ∗)
  finally show ?thesis **.**
qed

8.6 Lemma 9.2

context P0-min
begin

lemma error-9-2:
  assumes μ>0 d > 0
  shows ∀$^\infty$k. ok-fun-95b k + μ ∗ real k / d ≥ 0

```
proof –
  have ∀∞k. |ok-fun-95b k| ≤ (μ/d) * k
    using ok-fun-95b assms unfolding smallo-def
    by (auto dest!: spec [where x = μ/d])
  then show ?thesis
    by eventually-elim force
qed

definition Big-Far-9-2 ≡ λμ l. Big-Far-9-3 μ l ∧ Big-Far-9-5 μ l ∧ (∀k≥l.
ok-fun-95b k + μ*k/60 ≥ 0)

lemma Big-Far-9-2:
  assumes 0<μ0 μ0≤μ1 μ1<1
  shows ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ Big-Far-9-2 μ l
proof –
  have ∀∞l. ∀k≥l. (∀μ. μ0 ≤ μ ∧ μ ≤ μ1 ⟶ 0 ≤ ok-fun-95b k + μ * k / 60)
    using assms
    apply (intro eventually-all-ge-at-top eventually-all-geI0 error-9-2)
     apply (auto simp: divide-right-mono mult-right-mono elim!: order-trans)
    done
  then show ?thesis
    using assms Big-Far-9-3 Big-Far-9-5
    unfolding Big-Far-9-2-def
    apply (simp add: eventually-conj-iff all-imp-conj-distrib)
    by (smt (verit, ccfv-threshold) eventually-sequentially)
qed

end
```

Used for both 9.2 and 10.2

```
lemma (in Book') Off-diagonal-conclusion:
  defines ℛ ≡ Step-class {red-step}
  defines t ≡ card ℛ
  assumes Y: (k−t+l choose l) ≤ card (Yseq halted-point)
  shows False
proof –
  have t<k
    unfolding t-def ℛ-def using red-step-limit by blast
  have RN (k−t) l ≤ card (Yseq halted-point)
    by (metis Y add.commute RN-commute RN-le-choose le-trans)
  then obtain K
    where Ksub: K ⊆ Yseq halted-point
      and K: card K = k−t ∧ clique K Red ∨ card K = l ∧ clique K Blue
    by (meson Red-Blue-RN Yseq-subset-V size-clique-def)
  show False
    using K
  proof
    assume K: card K = k − t ∧ clique K Red
    have clique (K ∪ Aseq halted-point) Red
```

```
    proof (intro clique-Un)
      show clique (Aseq halted-point) Red
        by (meson A-Red-clique valid-state-seq)
      have all-edges-betw-un (Aseq halted-point) (Yseq halted-point) ⊆ Red
        using valid-state-seq Ksub
        by (auto simp: valid-state-def RB-state-def all-edges-betw-un-Un2)
      then show all-edges-betw-un K (Aseq halted-point) ⊆ Red
        using Ksub all-edges-betw-un-commute all-edges-betw-un-mono2 by blast
      show K ⊆ V
        using Ksub Yseq-subset-V by blast
    qed (use K Aseq-subset-V in auto)
    moreover have card (K ∪ Aseq halted-point) = k
    proof -
      have eqt: card (Aseq halted-point) = t
        using red-step-eq-Aseq ℛ-def t-def by simp
      have card (K ∪ Aseq halted-point) = card K + card (Aseq halted-point)
      proof (intro card-Un-disjoint)
        show finite K
          by (meson Ksub Yseq-subset-V finV finite-subset)
        have disjnt (Yseq halted-point) (Aseq halted-point)
          using valid-state-seq by (auto simp: valid-state-def disjoint-state-def)
        with Ksub show K ∩ Aseq halted-point = {}
          by (auto simp: disjnt-def)
      qed (simp add: finite-Aseq)
      also have ... = k
        using eqt K ‹t < k› by simp
      finally show ?thesis .
    qed
    moreover have K ∪ Aseq halted-point ⊆ V
      using Aseq-subset-V Ksub Yseq-subset-V by blast
    ultimately show False
      using no-Red-clique size-clique-def by blast
  next
    assume card K = l ∧ clique K Blue
    then show False
      using Ksub Yseq-subset-V no-Blue-clique size-clique-def by blast
  qed
qed
```

A little tricky to express since the Book locale assumes that there are no cliques in the original graph (page 9). So it's a contrapositive

```
lemma (in Book') Far-9-2-aux:
  fixes δ η::real
  defines δ ≡ γ/20
  assumes 0: real (card X0) ≥ nV/2 card Y0 ≥ nV div 2 p0 ≥ 1−γ−η
    — These are the assumptions about the red density of the graph
  assumes γ: γ ≤ 1/10 and η: 0≤η η ≤ γ/15
  assumes nV: real nV ≥ exp (−δ ∗ k) ∗ (k+l choose l)
  assumes big: Big-Far-9-2 γ l
```

```
  shows False
proof -
  define R where R ≡ Step-class {red-step}
  define t where t ≡ card R
  have γ01: 0 < γ γ < 1
    using ln0 l-le-k by (auto simp: γ-def)
  have big93: Big-Far-9-3 γ l
    using big by (auto simp: Big-Far-9-2-def)
  have t23: t ≥ 2*k / 3
    unfolding t-def R-def
  proof (rule Far-9-3)
    show γ ≤ 1/5
      using γ unfolding γ-def by linarith
    have min (1/200) (γ / 20) ≥ δ
      unfolding δ-def using γ ln0 by (simp add: γ-def)
    then show exp (- min (1/200) (γ / 20) * k) * (k+l choose l) ≤ nV
      using δ-def γ-def nV by force
    show 1/4 ≤ p0
      using η γ 0 by linarith
    show Big-Far-9-3 (γ) l
      using γ-def big93 by blast
  qed (use assms in auto)
  have t<k
    unfolding t-def R-def using γ01 red-step-limit by blast

  have ge-half: 1/2 ≤ 1-γ-η
    using γ η by linarith
  have exp (-1/3 + (1/5::real)) ≤ exp (10/9 * ln (134/150))
    by (approximation 9)
  also have ... ≤ exp (1 / (1-γ) * ln (134/150))
    using γ by (auto simp: divide-simps)
  also have ... ≤ exp (1 / (1-γ) * ln (1-γ-η))
    using γ η by (auto simp: divide-simps)
  also have ... = (1-γ-η) powr (1 / (1-γ))
    using ge-half by (simp add: powr-def)
  finally have A: exp (-1/3 + 1/5) ≤ (1-γ-η) powr (1 / (1-γ)) .

  have 3*t / (10*k) ≤ (-1/3 + 1/5) + t/(2*k)
    using t23 kn0 by (simp add: divide-simps)
  from mult-right-mono [OF this, of γ*t] γ01
  have 3*γ*t^2 / (10*k) ≤ γ*t*(-1/3 + 1/5) + γ*t^2/(2*k)
    by (simp add: eval-nat-numeral algebra-simps)
  then have exp (3*γ*t^2 / (10*k)) ≤ exp (-1/3 + 1/5) powr (γ*t) * exp
(γ*t^2/(2*k))
    by (simp add: mult-exp-exp exp-powr-real)
  also have ... ≤ (1-γ-η) powr ((γ*t) / (1-γ)) * exp (γ*t^2/(2*k))
    using γ01 powr-powr powr-mono2 [of γ*t exp (-1/3 + 1/5), OF - - A]
    by (intro mult-right-mono) auto
  finally have B: exp (3*γ*t^2 / (10*k)) ≤ (1-γ-η) powr ((γ*t) / (1-γ)) * exp
```

```
(γ∗t²/(2∗k)) .

  have (2∗k / 3)^2 ≤ t²
    using t23 by auto
  from kn0 γ01 mult-right-mono [OF this, of γ/(80∗k)]
  have C: δ∗k + γ∗k/60 ≤ 3∗γ∗t² / (20∗k)
    by (simp add: field-simps δ-def eval-nat-numeral)

  have exp (− 3∗γ∗t / (20∗k)) ≤ exp (−3 ∗ η/2)
  proof −
    have 1 ≤ 3/2 ∗ t/k
      using t23 kn0 by (auto simp: divide-simps)
    from mult-right-mono [OF this, of γ/15] γ01 η
    show ?thesis
      by simp
  qed
  also have ... ≤ 1 − η / (1−γ)
  proof −
    have §: 2/3 ≤ (1 − γ − η)
      using γ η by linarith
    have 1 / (1−η / (1−γ)) = 1 + η / (1−γ−η)
      using ge-half η by (simp add: divide-simps split: if-split-asm)
    also have ... ≤ 1 + 3 ∗ η / 2
      using mult-right-mono [OF §, of η] η ge-half by (simp add: field-simps)
    also have ... ≤ exp (3 ∗ η / 2)
      using exp-minus-ge [of −3∗η/2] by simp
    finally show ?thesis
      using γ01 ge-half
      by (simp add: exp-minus divide-simps mult.commute split: if-split-asm)
  qed
  also have ... = (1−γ−η) / (1−γ)
    using γ01 by (simp add: divide-simps)
  finally have exp (− 3∗γ∗t / (20∗k)) ≤ (1−γ−η) / (1−γ) .
  from powr-mono2 [of t, OF - - this] ge-half γ01
  have D: exp (− 3∗γ∗t² / (20∗k)) ≤ ((1−γ−η) / (1−γ))^t
   by (simp add: eval-nat-numeral powr-powr exp-powr-real mult-ac flip: powr-realpow)

  have Y: (k−t+l choose l) ≤ card (Yseq halted-point)
  proof −
    have 1 ∗ real(k−t+l choose l)
            ≤ exp (ok-fun-95b k + γ∗k/60) ∗ (k−t+l choose l)
      using big l-le-k unfolding Big-Far-9-2-def
      by (intro mult-right-mono mult-ge1-I) auto
     also have ... ≤ exp (3∗γ∗t² / (20∗k) + −δ ∗ k + ok-fun-95b k) ∗ (k−t+l
choose l)
      using C by simp
    also have ... = exp (3∗γ∗t² / (10∗k)) ∗ exp (−δ ∗ k + ok-fun-95b k) ∗ exp
(− 3∗γ∗t² / (20∗k))
           ∗ (k−t+l choose l)
```

by (*simp flip*: *exp-add*)
also have ... $\leq$ *exp* ($3*\gamma*t^2$ / ($10*k$)) * *exp* ($-\delta$ * *k* + *ok-fun-95b k*) * (($1-\gamma-\eta$)/($1-\gamma$)) ^*t*
* (*k*−*t*+*l choose l*)
using γ*01 ge-half D* **by** (*intro mult-right-mono*) *auto*
also have ... $\leq$ ($1-\gamma-\eta$) *powr* ($\gamma*t$ / ($1-\gamma$)) * *exp* (γ * t^2 / ($2*k$)) * *exp* ($-\delta$ * *k* + *ok-fun-95b k*)
* (($1-\gamma-\eta$)/($1-\gamma$)) ^*t* * (*k*−*t*+*l choose l*)
using γ*01 ge-half* **by** (*intro mult-right-mono B*) *auto*
also have ... = *exp* ($-\delta$ * *k* + *ok-fun-95b k*) * ($1-\gamma-\eta$) *powr* ($\gamma*t$ / ($1-\gamma$)) * (($1-\gamma-\eta$)/($1-\gamma$)) ^*t*
* *exp* (γ * (*real t*)^2 / ($2*k$)) * (*k*−*t*+*l choose l*)
by (*simp add*: *mult-ac*)
also have *95*: ... $\leq$ *real* (*card* (*Yseq halted-point*))
unfolding *t-def* $\mathcal{R}$*-def*
proof (*rule Far-9-5*)
show $1/2 \leq 1 - \gamma - \eta$
using *ge-half* γ*-def* **by** *blast*
show *Big-Far-9-5* (γ) *l*
using *Big-Far-9-2-def big* **unfolding** γ*-def* **by** *presburger*
qed (*use assms* **in** *auto*)
finally show *?thesis* **by** *simp*
qed
then show *False*
using *Off-diagonal-conclusion* **by** (*simp flip*: $\mathcal{R}$*-def t-def*)
qed

Mediation of 9.2 (and 10.2) from locale *Book-Basis* to the book locales with the starting sets of equal size

lemma (**in** *No-Cliques*) *to-Book*:
assumes *gd*: *p0-min* $\leq$ *graph-density Red*
assumes μ*01*: $0 < \mu$ $\mu < 1$
obtains *X0 Y0* **where** $l \geq 2$ *card X0* $\geq$ *real nV* / *2 card Y0* = *gorder div 2*
and *X0* = *V* \ *Y0 Y0*$\subseteq$*V*
and *graph-density Red* $\leq$ *gen-density Red X0 Y0*
and *Book V E p0-min Red Blue l k* μ *X0 Y0*
proof −
have *Red* $\neq$ {}
using *gd p0-min* **by** (*auto simp*: *graph-density-def*)
then have *gorder* $\geq$ *2*
by (*metis Red-E card-mono equals0I finV subset-empty two-edges wellformed*)
then have *div2*: *0* < *gorder div 2 gorder div 2* < *gorder*
by *auto*
then obtain *Y0* **where** *Y0*: *card Y0* = *gorder div 2 Y0*$\subseteq$*V*
graph-density Red $\leq$ *gen-density Red* (*V**Y0*) *Y0*
by (*metis complete Red-E exists-density-edge-density gen-density-commute*)
define *X0* **where** *X0* $\equiv$ *V* \ *Y0*
interpret *Book V E p0-min Red Blue l k* μ *X0 Y0*
proof

```
    show X0⊆V disjnt X0 Y0
      by (auto simp: X0-def disjnt-iff)
    show p0-min ≤ gen-density Red X0 Y0
      using X0-def Y0 gd gen-density-commute p0-min by auto
  qed (use assms ‹Y0⊆V› in auto)
  have False if l<2
    using that unfolding less-2-cases-iff
  proof
    assume l = Suc 0
    with Y0 div2 show False
      by (metis RN-1' no-Red-clique no-Blue-clique Red-Blue-RN Suc-leI kn0)
  qed (use ln0 in auto)
  with l-le-k have l≥2
    by force
  have card-X0: card X0 ≥ nV/2
    using Y0 ‹Y0⊆V› unfolding X0-def
    by (simp add: card-Diff-subset finite-Y0)
  then show thesis
    using Book-axioms X0-def Y0 ‹2 ≤ l› that by blast
qed
```

Material that needs to be proved **outside** the book locales

As above, for *Book'*

```
lemma (in No-Cliques) to-Book':
  assumes gd: p0-min ≤ graph-density Red
  assumes l: 0<l l≤k
  obtains X0 Y0 where l≥2 card X0 ≥ real nV / 2 card Y0 = gorder div 2 and
X0 = V \ Y0 Y0⊆V
    and graph-density Red ≤ gen-density Red X0 Y0
    and Book' V E p0-min Red Blue l k (real l / (real k + real l)) X0 Y0
proof -
  define γ where γ ≡ real l / (real k + real l)
  have 0 < γ γ < 1
    using l by (auto simp: γ-def)
  with assms to-Book [of γ]
  obtain X0 Y0 where *: l≥2 card X0 ≥ real nV / 2 card Y0 = gorder div 2 X0
= V \ Y0 Y0⊆V
    graph-density Red ≤ gen-density Red X0 Y0 Book V E p0-min Red Blue l k γ
X0 Y0
    by blast
  then interpret Book V E p0-min Red Blue l k γ X0 Y0
    by blast
  have Book' V E p0-min Red Blue l k γ X0 Y0
    using Book' γ-def by auto
  with * assms show ?thesis
    using γ-def that by blast
qed

lemma (in No-Cliques) Far-9-2:
```

fixes $\delta\ \gamma\ \eta$::*real*
defines $\gamma \equiv l\ /\ (real\ k\ +\ real\ l)$
defines $\delta \equiv \gamma/20$
assumes *gd*: $graph\text{-}density\ Red \geq 1-\gamma-\eta$ **and** *p0-min-OK*: $p0\text{-}min \leq 1-\gamma-\eta$
assumes $\gamma \leq 1/10$ **and** η: $0\leq\eta\ \eta \leq \gamma/15$
assumes *nV*: $real\ nV \geq exp\ (-\delta * k) * (k+l\ choose\ l)$
assumes *big*: *Big-Far-9-2* γ *l*
shows *False*
proof −
obtain *X0 Y0* **where** $l\geq 2$ **and** *card-X0*: $card\ X0 \geq real\ nV\ /\ 2$
and *card-Y0*: $card\ Y0 = gorder\ div\ 2$
and *X0-def*: $X0 = V \setminus Y0$ **and** $Y0\subseteq V$
and *gd-le*: $graph\text{-}density\ Red \leq gen\text{-}density\ Red\ X0\ Y0$
and *Book′ V E p0-min Red Blue l k* γ *X0 Y0*
using *to-Book′ assms p0-min no-Red-clique no-Blue-clique ln0* **by** *auto*
then interpret *Book′ V E p0-min Red Blue l k* γ *X0 Y0*
by *blast*
show *False*
proof (*intro Far-9-2-aux* [*of* η])
show $1 - \gamma - \eta \leq p0$
using *X0-def* γ*-def gd gd-le gen-density-commute p0-def* **by** *auto*
qed (*use assms card-X0 card-Y0* **in** *auto*)
qed

8.7 Theorem 9.1

An arithmetical lemma proved outside of the locales

lemma *kl-choose*:
fixes *l k*::*nat*
assumes $m<l\ k>0$
defines $PM \equiv \prod i<m.\ (l - real\ i)\ /\ (k+l-real\ i)$
shows $(k+l\ choose\ l) = (k+l-m\ choose\ (l-m))\ /\ PM$
proof −
have *inj*: $inj\text{-}on\ (\lambda i.\ i-m)\ \{m..<l\}$ — relating the power and binomials; maybe easier using factorials
by (*auto simp*: *inj-on-def*)
have $(\prod i<l.\ (k+l-i)\ /\ (l-i))\ /\ (\prod i<m.\ (k+l-i)\ /\ (l-i))$
$= (\prod i = m..<l.\ (k+l-i)\ /\ (l-i))$
using *prod-divide-nat-ivl* [*of 0 m l* $\lambda i.\ (k+l-i)\ /\ (l-i)$] ‹$m < l$›
by (*simp add*: *atLeast0LessThan*)
also have … $= (\prod i<l - m.\ (k+l-m - i)\ /\ (l-m-i))$
apply (*intro prod.reindex-cong* [*OF inj*, *symmetric*])
by (*auto simp*: *image-minus-const-atLeastLessThan-nat*)
finally
have $(\prod i < l-m.\ (k+l-m - i)\ /\ (l-m-i))$
$= (\prod i < l.\ (k+l-i)\ /\ (l-i))\ /\ (\prod i<m.\ (k+l-i)\ /\ (l-i))$
by *linarith*
also have … $= (k+l\ choose\ l) * inverse\ (\prod i<m.\ (k+l-i)\ /\ (l-i))$
by (*simp add*: *field-simps atLeast0LessThan binomial-altdef-of-nat*)

```
  also have ... = (k+l choose l) * PM
    unfolding PM-def using ‹m < l› ‹k>0›
    by (simp add: atLeast0LessThan flip: prod-inversef)
  finally have (k+l−m choose (l−m)) = (k+l choose l) * PM
    by (simp add: atLeast0LessThan binomial-altdef-of-nat)
  then show real(k+l choose l) = (k+l−m choose (l−m)) / PM
    by auto
qed
```

```
context P0-min
begin
```

The proof considers a smaller graph, so l needs to be so big that the smaller l' will be big enough.

```
definition Big-Far-9-1 :: real ⇒ nat ⇒ bool where
  Big-Far-9-1 ≡ λμ l. l≥3 ∧ (∀l' γ. real l' ≥ (10/11) * μ * real l ⟶ μ² ≤ γ ∧
γ ≤ 1/10 ⟶ Big-Far-9-2 γ l')
```

The proof of theorem 10.1 requires a range of values

```
lemma Big-Far-9-1:
  assumes 0<μ0 μ0≤1/10
  shows ∀∞l. ∀μ. μ0 ≤ μ ∧ μ ≤ 1/10 ⟶ Big-Far-9-1 μ l
proof −
  have μ0² ≤ 1/10
   using assms by (smt (verit, ccfv-threshold) le-divide-eq-1 mult-left-le power2-eq-square)
  then have ∀∞l. ∀γ. μ0² ≤ γ ∧ γ ≤ 1/10 ⟶ Big-Far-9-2 γ l
    using assms by (intro Big-Far-9-2) auto
  then obtain N where N: ∀l≥N. ∀γ. μ0² ≤ γ ∧ γ ≤ 1/10 ⟶ Big-Far-9-2 γ
l
    using eventually-sequentially by auto
  define M where M ≡ nat⌈11*N / (10*μ0)⌉
  have (10/11) * μ0 * l ≥ N if l ≥ M for l
   using that by (simp add: M-def ‹μ0>0› mult-of-nat-commute pos-divide-le-eq)
  with N have ∀l≥M. ∀l' γ. (10/11) * μ0 * l ≤ l' ⟶ μ0² ≤ γ ∧ γ ≤ 1 / 10
⟶ Big-Far-9-2 γ l'
    by (smt (verit, ccfv-SIG) of-nat-le-iff)
  then have ∀∞l. ∀l' γ. (10/11) * μ0 * l ≤ l' ⟶ μ0² ≤ γ ∧ γ ≤ 1 / 10 ⟶
Big-Far-9-2 γ l'
    by (auto simp: eventually-sequentially)
  moreover have ∀∞l. l≥3
    by simp
  ultimately show ?thesis
    unfolding Big-Far-9-1-def
    apply eventually-elim
   by (smt (verit) ‹0<μ0› mult-left-mono mult-right-mono of-nat-less-0-iff power-mono
zero-less-mult-iff)
qed
```

The text claims the result for all k and l, not just those sufficiently large, but the $o(k)$ function allowed in the exponent provides a fudge factor

```
theorem Far-9-1:
  fixes l k::nat
  fixes δ γ::real
  defines γ ≡ real l / (real k + real l)
  defines δ ≡ γ/20
  assumes γ: γ ≤ 1/10
  assumes big: Big-Far-9-1 γ l
  assumes p0-min-91: p0-min ≤ 1 − (1/10) ∗ (1 + 1/15)
  shows RN k l ≤ exp (−δ∗k + 1) ∗ (k+l choose l)
proof (rule ccontr)
  assume non: ¬ RN k l ≤ exp (−δ ∗ k + 1) ∗ (k+l choose l)
  with RN-eq-0-iff have l>0 by force
  with γ have l9k: 9∗l ≤ k
    by (auto simp: γ-def divide-simps)
  have l≤k
    using γ-def γ nat-le-real-less by fastforce
  with ‹l>0› have k>0 by linarith
  define ξ::real where ξ ≡ 1/15
  define U-lower-bound-ratio where — Bhavik's name
    U-lower-bound-ratio ≡ λm. (1+ξ)^m ∗ (∏ i<m. (l − real i) / (k+l − real i))

  define n where n ≡ RN k l − 1
  have l≥3
    using big by (auto simp: Big-Far-9-1-def)
  have k≥27
    using l9k ‹l≥3› by linarith
  have exp 1 / (exp 1 − 2) < (27::real)
    by (approximation 5)
  also have RN27: . . . ≤ RN k l
    by (meson RN-3plus' ‹l≥3› ‹k≥27› le-trans numeral-le-real-of-nat-iff)
  finally have exp 1 / (exp 1 − 2) < RN k l .
  moreover have n < RN k l
    using RN27 by (simp add: n-def)
  moreover have 2 < exp (1::real)
    by (approximation 5)
  ultimately have nRNe: n/2 > RN k l / exp 1
    by (simp add: n-def field-split-simps)

  have (k+l choose l) / exp (−1 + δ∗k) < RN k l
    by (smt (verit) divide-inverse exp-minus mult-minus-left mult-of-nat-commute
non)
  then have (RN k l / exp 1) ∗ exp (δ∗k) > (k+l choose l)
    unfolding exp-add exp-minus by (simp add: field-simps)
  with nRNe have n2exp-gt: (n/2) ∗ exp (δ∗k) > (k+l choose l)
    by (smt (verit, best) exp-gt-zero mult-le-cancel-right-pos)
  then have nexp-gt: n ∗ exp (δ∗k) > (k+l choose l)
    by simp
```

```
define V where V ≡ {..<n}
define E where E ≡ all-edges V
interpret Book-Basis V E
proof qed (auto simp: V-def E-def comp-sgraph.wellformed comp-sgraph.two-edges)
have [simp]: nV = n
  by (simp add: V-def)
then obtain Red Blue
  where Red-E: Red ⊆ E and Blue-def: Blue = E−Red
    and no-Red-K: ¬ (∃K. size-clique k K Red)
    and no-Blue-K: ¬ (∃K. size-clique l K Blue)
  by (metis ‹n < RN k l› less-RN-Red-Blue)
have Blue-E: Blue ⊆ E and disjnt-Red-Blue: disjnt Red Blue
and  Blue-eq: Blue = all-edges V − Red
  using complete by (auto simp: Blue-def disjnt-iff E-def)
define is-good-clique where
  is-good-clique ≡ λi K. clique K Blue ∧ K ⊆ V ∧
                              card (V ∩ (⋂w∈K. Neighbours Blue w))
                              ≥ real i ∗ U-lower-bound-ratio (card K) − card K
have is-good-card: card K < l if is-good-clique i K for i K
  using no-Blue-K that unfolding is-good-clique-def
  by (metis nat-neq-iff size-clique-def size-clique-smaller)
define GC where GC ≡ {C. is-good-clique n C}
have GC ≠ {}
  by (auto simp: GC-def is-good-clique-def U-lower-bound-ratio-def E-def V-def)
have GC ⊆ Pow V
  by (auto simp: is-good-clique-def GC-def)
then have finite GC
  by (simp add: finV finite-subset)
then obtain W where W ∈ GC and MaxW: Max (card ` GC) = card W
  using ‹GC ≠ {}› obtains-MAX by blast
then have 49: is-good-clique n W
  using GC-def by blast
have max49: ¬ is-good-clique n (insert x W) if x∈V∖W for x
proof
  assume x: is-good-clique n (insert x W)
  then have card (insert x W) = Suc (card W)
    using finV is-good-clique-def finite-subset that by fastforce
  with x ‹finite GC› have Max (card ` GC) ≥ Suc (card W)
    by (simp add: GC-def rev-image-eqI)
  then show False
    by (simp add: MaxW)
qed

have W⊆V
  using 49 by (auto simp: is-good-clique-def)
define m where m ≡ card W
define γ' where γ' ≡ (l − real m) / (k+l−real m)
define η where η ≡ ξ ∗ γ'
```

```
  have Red-Blue-RN: ∃ K ⊆ X. size-clique m K Red ∨ size-clique n K Blue
    if card X ≥ RN m n X⊆V for m n and X
   using partn-lst-imp-is-clique-RN [OF is-Ramsey-number-RN [of m n]] finV that

    unfolding is-clique-RN-def size-clique-def clique-indep-def Blue-eq
    by (metis clique-iff-indep finite-subset subset-trans)
  define U where U ≡ V ∩ (⋂ w∈W. Neighbours Blue w)
  define EU where EU ≡ E ∩ Pow U
  define RedU where RedU ≡ Red ∩ Pow U
  define BlueU where BlueU ≡ Blue ∩ Pow U

  have RN k l > 0
    using ‹n < RN k l› by auto
  have γ' > 0
    using is-good-card [OF 49] by (simp add: γ'-def m-def)
  then have η > 0
    by (simp add: η-def ξ-def)
  have finite W
    using ‹W ⊆ V› finV finite-subset by (auto simp: V-def)
  have U ⊆ V and VUU: V ∩ U = U
    by (force simp: U-def)+
  have disjnt U W
     using Blue-E not-own-Neighbour unfolding E-def V-def U-def disjnt-iff by
blast
  have m<l
    using 49 is-good-card m-def by blast
  then have γ1516: γ' ≤ 15/16
    using γ-def γ by (simp add: γ'-def divide-simps)
  then have γ'-le1: (1+ξ) ∗ γ' ≤ 1
    by (simp add: ξ-def)

  have cardU: n ∗ U-lower-bound-ratio m ≤ m + card U
    using 49 VUU unfolding is-good-clique-def U-def m-def by force
  obtain [iff]: finite RedU finite BlueU RedU ⊆ EU
   using BlueU-def EU-def RedU-def E-def V-def Red-E Blue-E fin-edges finite-subset
by blast
  have card-RedU-le: card RedU ≤ card EU
    by (metis EU-def E-def ‹RedU ⊆ EU› card-mono fin-all-edges finite-Int)
  interpret UBB: Book-Basis U E ∩ Pow U p0-min
  proof
    fix e
    assume e ∈ E ∩ Pow U
    with two-edges show e ⊆ U card e = 2 by auto
  next
    show finite U
      using ‹U ⊆ V› by (simp add: V-def finite-subset)
    have x ∈ E if x ∈ all-edges U for x
      using ‹U ⊆ V› all-edges-mono that complete E-def by blast
    then show E ∩ Pow U = all-edges U
```

```
      using comp-sgraph.wellformed ‹U ⊆ V› by (auto intro: e-in-all-edges-ss)
  qed auto

  have clique-W: size-clique m W Blue
    using 49 is-good-clique-def size-clique-def V-def m-def by blast

  define PM where PM ≡ ∏ i<m. (l − real i) / (k+l−real i)
  then have U-lower-m: U-lower-bound-ratio m = (1+ξ)^m ∗ PM
    using U-lower-bound-ratio-def by blast
  have prod-gt0: PM > 0
    unfolding PM-def using ‹m<l› by (intro prod-pos) auto

  have kl-choose: real(k+l choose l) = (k+l−m choose (l−m)) / PM
    unfolding PM-def using kl-choose ‹0 < k› ‹m < l› by blast
  — Now a huge effort just to show that U is nontrivial. Proof probably shows its
cardinality exceeds a multiple of l
  define ekl20 where ekl20 ≡ exp (k / (20∗(k+l)))
  have ekl20-eq: exp (δ∗k) = ekl20^l
      by (simp add: δ-def γ-def ekl20-def field-simps flip: exp-of-nat2-mult)
  have ekl20 ≤ exp(1/20)
    unfolding ekl20-def using ‹m < l› by fastforce
  also have ... ≤ (1+ξ)
    unfolding ξ-def by (approximation 10)
  finally have exp120: ekl20 ≤ 1 + ξ .
  have ekl20-gt0: 0 < ekl20
    by (simp add: ekl20-def)

  have 3∗l + Suc l − q ≤ (k+q choose q) / exp(δ∗k) ∗ (1+ξ) ^ (l − q)
    if 1≤q q≤l for q
    using that
  proof (induction q rule: nat-induct-at-least)
    case base
    have ekl20^l = ekl20^(l−1) ∗ ekl20
      by (metis ‹0 < l› power-minus-mult)
    also have ... ≤ (1+ξ) ^ (l−1) ∗ ekl20
      using ekl20-def exp120 power-mono by fastforce
    also have ... ≤ 2 ∗ (1+ξ) ^ (l−1)
    proof −
      have §: ekl20 ≤ 2
        using ξ-def exp120 by linarith
      from mult-right-mono [OF this, of (1+ξ) ^ (l−1)]
      show ?thesis by (simp add: mult-ac ξ-def)
    qed
    finally have ekl20^l ≤ 2 ∗ (1+ξ) ^ (l−1)
      by argo
    then have 1/2 ≤ (1+ξ) ^ (l−1) / ekl20^l
      using ekl20-def by auto
    moreover have 4 ∗ real l / (1 + real k) ≤ 1/2
      using l9k by (simp add: divide-simps)
```

```
    ultimately have 4 * real l / (1 + real k) ≤ (1+ξ) ^ (l−1) / ekl20^l
      by linarith
    then show ?case
      by (simp add: field-simps ekl20-eq)
  next
    case (Suc q)
    then have ‡: (1+ξ) ^ (l − q) = (1+ξ) * (1+ξ) ^ (l − Suc q)
      by (metis Suc-diff-le diff-Suc-Suc power.simps(2))
    have real(k + q choose q) ≤ real(k + q choose Suc q) 0 ≤ (1+ξ) ^ (l − Suc
q)
      using ‹Suc q ≤ l› l9k by (auto simp: ξ-def binomial-mono)
    from mult-right-mono [OF this]
    have (k + q choose q) * (1+ξ) ^ (l − q) / exp (δ * k) − 1
        ≤ (real (k + q choose q) + (k + q choose Suc q)) * (1+ξ) ^ (l − Suc q) /
exp (δ * k)
      unfolding ‡ by (simp add: ξ-def field-simps add-increasing)
    with Suc show ?case by force
  qed
  from ‹m<l› this [of l−m]
  have 1 + 3*l + real m ≤ (k+l−m choose (l−m)) / exp δ ^ k * (1+ξ) ^ m
    by (simp add: Suc-leI exp-of-nat2-mult)
  also have ... ≤ (k+l−m choose (l−m)) / exp (δ * k) * (1+ξ) ^ m
    by (simp add: exp-of-nat2-mult)
  also have ... < PM * (real n * (1+ξ) ^ m)
  proof −
    have §: (k+l choose l) / exp (δ * k) < n
      by (simp add: less-eq-real-def nexp-gt pos-divide-less-eq)
    show ?thesis
      using mult-strict-left-mono [OF §, of PM * (1+ξ) ^ m] kl-choose prod-gt0
      by (auto simp: field-simps ξ-def)
  qed
  also have ... = real n * U-lower-bound-ratio m
    by (simp add: U-lower-m)
  finally have U-MINUS-M: 3*l + 1 < real n * U-lower-bound-ratio m − m
    by linarith
  then have cardU-gt: card U > 3*l + 1
    using cardU by linarith
  with UBB.complete have card EU > 0 card U > 1
    by (simp-all add: EU-def UBB.finV card-all-edges)
  have BlueU-eq: BlueU = EU \ RedU
    using Blue-eq complete by (fastforce simp: BlueU-def RedU-def EU-def V-def
E-def)
  have [simp]: UBB.graph-size = card EU
    using EU-def by blast
  have γ' ≤ γ
    using ‹m<l› ‹k>0› by (simp add: γ-def γ'-def field-simps)
  have False if UBB.graph-density RedU < 1 − γ' − η
  proof −    — by maximality, etc.
    have §: UBB.graph-density BlueU ≥ γ' + η
```

using *that* ‹*card EU* > *0*› *card-RedU-le*
by (*simp add*: *BlueU-eq UBB.graph-density-def diff-divide-distrib card-Diff-subset*)
have *Nx*: *Neighbours BlueU x* ∩ (*U* \ {*x*}) = *Neighbours BlueU x* **for** *x*
using *that* **by** (*auto simp*: *BlueU-eq EU-def Neighbours-def*)
have *BlueU* ⊆ *E* ∩ *Pow U*
using *BlueU-eq EU-def* **by** *blast*
with *UBB.exists-density-edge-density* [*of 1 BlueU*]
obtain *x* **where** *x*∈*U* **and** *x*: *UBB.graph-density BlueU* ≤ *UBB.gen-density BlueU* {*x*} (*U*\{*x*})
by (*metis UBB.complete* ‹*1* < *UBB.gorder*› *card-1-singletonE insertI1 zero-less-one subsetD*)
with § **have** γ′ + η ≤ *UBB.gen-density BlueU* (*U*\{*x*}) {*x*}
using *UBB.gen-density-commute* **by** *auto*
then have ∗: (γ′ + η) ∗ (*card U* − *1*) ≤ *card* (*Neighbours BlueU x*)
using ‹*BlueU* ⊆ *E* ∩ *Pow U*› ‹*card U* > *1*› ‹*x* ∈ *U*›
by (*simp add*: *UBB.gen-density-def UBB.edge-card-eq-sum-Neighbours UBB.finV divide-simps Nx*)

have *x*: *x* ∈ *V**W*
using ‹*x* ∈ *U*› ‹*U* ⊆ *V*› ‹*disjnt U W*› **by** (*auto simp*: *U-def disjnt-iff*)
moreover
have *is-good-clique n* (*insert x W*)
unfolding *is-good-clique-def*
proof (*intro conjI*)
show *clique* (*insert x W*) *Blue*
proof (*intro clique-insert*)
show *clique W Blue*
using *49 is-good-clique-def* **by** *blast*
show *all-edges-betw-un* {*x*} *W* ⊆ *Blue*
using ‹*x*∈*U*› **by** (*auto simp*: *U-def all-edges-betw-un-def insert-commute in-Neighbours-iff*)
qed (*use* ‹*W* ⊆ *V*› ‹*x* ∈ *V**W*› **in** *auto*)
next
show *insert x W* ⊆ *V*
using ‹*W* ⊆ *V*› ‹*x* ∈ *V**W*› **by** *auto*
next
have *NB-Int-U*: *Neighbours Blue x* ∩ *U* = *Neighbours BlueU x*
using ‹*x* ∈ *U*› **by** (*auto simp*: *BlueU-def U-def Neighbours-def*)
have *ulb-ins*: *U-lower-bound-ratio* (*card* (*insert x W*)) = *U-lower-bound-ratio m* ∗ (*1*+ξ) ∗ γ′
using ‹*x* ∈ *V**W*› ‹*finite W*› **by** (*simp add*: *U-lower-bound-ratio-def* γ′*-def m-def*)
have *n* ∗ *U-lower-bound-ratio* (*card* (*insert x W*)) = *n* ∗ *U-lower-bound-ratio m* ∗ (*1*+ξ) ∗ γ′
by (*simp add*: *ulb-ins*)
also have . . . ≤ *real* (*m* + *card U*) ∗ (*1*+ξ) ∗ γ′
using *mult-right-mono* [*OF cardU*, *of* (*1*+ξ) ∗ γ′] ‹*0* < η› ‹*0* < γ′› η*-def* **by** *argo*
also have . . . ≤ *m* + *card U* ∗ (*1*+ξ) ∗ γ′

using *mult-left-mono* [*OF* γ'*-le1*, *of m*] **by** (*simp add*: *algebra-simps*)
also have ... $\leq$ *Suc m* + (γ' + η) $*$ (*UBB.gorder* − *Suc 0*)
using $*$ ‹*x* ∈ *V**W*› ‹*finite W*› *cardU-gt* γ*1516*
apply (*simp add*: *U-lower-bound-ratio-def* ξ*-def* η*-def*)
by (*simp add*: *algebra-simps*)
also have ... $\leq$ *Suc m* + *card* (*V* ∩ ⋂ (*Neighbours Blue* ‘ *insert x W*))
using $*$ *NB-Int-U finV* **by** (*simp add*: *U-def Int-ac*)
also have ... = *real* (*card* (*insert x W*) + *card* (*V* ∩ ⋂ (*Neighbours Blue* ‘ *insert x W*)))
using *x* ‹*finite W*› *VUU* **by** (*auto simp*: *U-def m-def*)
finally show *n* $*$ *U-lower-bound-ratio* (*card*(*insert x W*)) − *card*(*insert x W*)
$\leq$ *card* (*V* ∩ ⋂ (*Neighbours Blue* ‘ *insert x W*))
by *simp*
qed
ultimately show *False*
using *max49* **by** *blast*
qed
then have *gd-RedU-ge*: *UBB.graph-density RedU* $\geq$ *1* − γ' − η **by** *force*

— Bhavik's gamma'_le_gamma_iff
have $\gamma'\gamma$*2*: $\gamma' < \gamma^2$ ⟷ (*real k* $*$ *real l*) + (*real l* $*$ *real l*) < (*real k* $*$ *real m*) + (*real l* $*$ (*real m* $*$ *2*))
using ‹*m* < *l*›
apply (*simp add*: γ'*-def* γ*-def eval-nat-numeral divide-simps*; *simp add*: *algebra-simps*)
by (*metis* ‹*k*>*0*› *mult-less-cancel-left-pos of-nat-0-less-iff distrib-left*)
also have ... ⟷ (*l* $*$ (*k*+*l*)) / (*k* + *2* $*$ *l*) < *m*
using ‹*m* < *l*› **by** (*simp add*: *field-simps*)
finally have $\gamma'\gamma$*2-iff*: $\gamma' < \gamma^2$ ⟷ (*l* $*$ (*k*+*l*)) / (*k* + *2* $*$ *l*) < *m* **.**
— in both cases below, we find a blue clique of size *l* − *m*
have *extend-Blue-clique*: ∃ *K'*. *size-clique l K' Blue*
if *K* ⊆ *U size-clique* (*l*−*m*) *K Blue* **for** *K*
proof −
have *K*: *card K* = *l*−*m clique K Blue*
using *that* **by** (*auto simp*: *size-clique-def*)
define *K'* **where** *K'* ≡ *K* ∪ *W*
have *card K'* = *l*
unfolding *K'-def*
proof (*subst card-Un-disjnt*)
show *finite K finite W*
using *UBB.finV* ‹*K* ⊆ *U*› *finite-subset* ‹*finite W*› **by** *blast*+
show *disjnt K W*
using ‹*disjnt U W*› ‹*K* ⊆ *U*› *disjnt-subset1* **by** *blast*
show *card K* + *card W* = *l*
using *K* ‹*m* < *l*› *m-def* **by** *auto*
qed
moreover have *clique K' Blue*
using ‹*clique K Blue*› *clique-W* ‹*K* ⊆ *U*›
unfolding *K'-def size-clique-def U-def*
by (*force simp*: *in-Neighbours-iff insert-commute intro*: *Ramsey.clique-Un*)

```
      ultimately show ?thesis
        unfolding K'-def size-clique-def using ‹K ⊆ U› ‹U ⊆ V› ‹W ⊆ V› by
auto
    qed

    show False
    proof (cases γ' < γ²)
      case True
      with γ'γ2 have YKK: γ*k ≤ m
        using ‹0<k› ‹m < l›
        apply (simp add: γ-def field-simps)
        by (smt (verit, best) distrib-left mult-left-mono of-nat-0-le-iff)
      have ln1ξ: ln (1+ξ) * 20 ≥ 1
        unfolding ξ-def by (approximation 10)
      with YKK have §: m * ln (1+ξ) ≥ δ * k
        unfolding δ-def using zero-le-one mult-mono by fastforce
      have powerm: (1+ξ)^m ≥ exp (δ * k)
        using exp-mono [OF §]
      by (smt (verit) η-def ‹0 < η› ‹0 < γ'› exp-ln-iff exp-of-nat-mult zero-le-mult-iff)
      have n * (1+ξ)^m ≥ (k+l choose l)
        by (smt (verit, best) mult-left-mono nexp-gt of-nat-0-le-iff powerm)
      then have **: n * U-lower-bound-ratio m ≥ (k+l−m choose (l−m))
        using ‹m<l› prod-gt0 kl-choose by (auto simp: U-lower-m field-simps)

      have m-le-choose: m ≤ (k+l−m−1 choose (l−m))
      proof (cases m=0)
        case False
        have m ≤ (k+l−m−1 choose 1)
          using ‹l≤k› ‹m<l› by simp
        also have ... ≤ (k+l−m−1 choose (l−m))
          using False ‹l≤k› ‹m<l› by (intro binomial-mono) auto
        finally have m-le-choose: m ≤ (k+l−m−1 choose (l−m)) .
        then show ?thesis .
      qed auto
      have RN k (l−m) ≤ k + (l−m) − 2 choose (k − 1)
        by (rule RN-le-choose-strong)
      also have ... ≤ (k+l−m−1 choose k)
        using ‹l≤k› ‹m<l› choose-reduce-nat by simp
      also have ... = (k+l−m−1 choose (l−m−1))
        using ‹m<l› by (simp add: binomial-symmetric [of k])
      also have ... = (k+l−m choose (l−m)) − (k+l−m−1 choose (l−m))
        using ‹l≤k› ‹m<l› choose-reduce-nat by simp
      also have ... ≤ (k+l−m choose (l−m)) − m
        using m-le-choose by linarith
      finally have RN k (l−m) ≤ (k+l−m choose (l−m)) − m .
      then have card U ≥ RN k (l−m)
        using 49 ** VUU by (force simp: is-good-clique-def U-def m-def)
      with Red-Blue-RN no-Red-K ‹U ⊆ V›
      obtain K where K ⊆ U size-clique (l−m) K Blue by meson
```

```
    then show False
      using no-Blue-K extend-Blue-clique by blast
  next
    case False
    have YMK: γ−γ′ ≤ m/k
      using ‹m<l›
      apply (simp add: γ-def γ′-def divide-simps)
      apply (simp add: algebra-simps)
     by (smt (verit) mult-left-mono mult-right-mono nat-less-real-le of-nat-0-le-iff)

    define δ′ where δ′ ≡ γ′/20
    have no-RedU-K: ¬ (∃K. UBB.size-clique k K RedU)
      unfolding UBB.size-clique-def RedU-def
     by (metis Int-subset-iff VUU all-edges-subset-iff-clique no-Red-K size-clique-def)
     have (∃K. UBB.size-clique k K RedU) ∨ (∃K. UBB.size-clique (l−m) K
BlueU)
    proof (rule ccontr)
      assume neg: ¬ ((∃K. UBB.size-clique k K RedU) ∨ (∃K. UBB.size-clique
(l−m) K BlueU))
      interpret UBB-NC: No-Cliques U E ∩ Pow U p0-min RedU BlueU l−m k
      proof
        show BlueU = E ∩ Pow U \ RedU
          using BlueU-eq EU-def by fastforce
      qed (use neg EU-def ‹RedU ⊆ EU› no-RedU-K ‹l≤k› in auto)
      show False
      proof (intro UBB-NC.Far-9-2)
        have exp (δ∗k) ∗ exp (−δ′∗k) = exp (γ∗k/20 − γ′∗k/20)
          unfolding δ-def δ′-def by (simp add: mult-exp-exp)
        also have ... ≤ exp (m/20)
          using YMK ‹0 < k› by (simp add: left-diff-distrib divide-simps)
        also have ... ≤ (1+ξ)^m
        proof −
          have ln (16 / 15) ∗ 20 ≥ (1::real)
            by (approximation 5)
          from mult-left-mono [OF this]
          show ?thesis
            by (simp add: ξ-def powr-def mult-ac flip: powr-realpow)
        qed
        finally have expexp: exp (δ∗k) ∗ exp (−δ′∗k) ≤ (1+ξ) ^ m .

        have exp (−δ′∗k) ∗ (k + (l−m) choose (l−m)) = exp (−δ′∗k) ∗ PM ∗ (k+l
choose l)
          using ‹m < l› kl-choose by force
        also have ... < (n/2) ∗ exp (δ∗k) ∗ exp (−δ′∗k) ∗ PM
          using n2exp-gt prod-gt0 by auto
        also have ... ≤ (n/2) ∗ (1+ξ) ^ m ∗ PM
          using expexp less-eq-real-def prod-gt0 by fastforce
        also have ... ≤ n ∗ U-lower-bound-ratio m − m  — where I was stuck: the
"minus m"
```

using *PM-def U-MINUS-M U-lower-bound-ratio-def* ‹$m < l$› **by** *fastforce*
finally have $exp\ (-\delta'*k) * (k + (l-m)\ choose\ (l-m)) \leq n * U\text{-}lower\text{-}bound\text{-}ratio\ m - m$
by *linarith*
also have ... $\leq UBB.nV$
using *cardU* **by** *linarith*
finally have $exp\ (-\delta'*k) * (k + (l-m)\ choose\ (l-m)) \leq UBB.nV$ **.**
then show $exp\ (-\ ((l-m)\ /\ (k + real\ (l-m))\ /\ 20) * k) * (k + (l-m)\ choose\ (l-m)) \leq UBB.nV$
using ‹$m < l$› **by** (*simp add*: δ'*-def* γ'*-def*) *argo*
next
show $1 - real\ (l-m)\ /\ (real\ k + real\ (l-m)) - \eta \leq UBB.graph\text{-}density\ RedU$
using *gd-RedU-ge* ‹$\gamma' \leq \gamma$› ‹$m < l$› **unfolding** γ*-def* γ'*-def*
by (*smt* (*verit*) *less-or-eq-imp-le of-nat-add of-nat-diff*)
have $p0\text{-}min \leq 1 - \gamma - \eta$
using ‹$\gamma' \leq \gamma$› γ *p0-min-91* **by** (*auto simp*: η*-def* ξ*-def*)
also have ... $\leq 1 - (l-m)\ /\ (real\ k + real\ (l-m)) - \eta$
using ‹$\gamma' \leq \gamma$› ‹$m<l$› **by** (*simp add*: γ*-def* γ'*-def algebra-simps*)
finally show $p0\text{-}min \leq 1 - (l-m)\ /\ (real\ k + real\ (l-m)) - \eta$ **.**
next
have $m \leq l * (k + real\ l)\ /\ (k + 2 * real\ l)$
using *False* $\gamma'\gamma$*2-iff* **by** *auto*
also have ... $\leq l * (1 - (10/11)*\gamma)$
using γ ‹$l>0$› **by** (*simp add*: γ*-def field-split-simps*)
finally have $m \leq real\ l * (1 - (10/11)*\gamma)$
by *force*
then have $real\ l - real\ m \geq (10/11) * \gamma * l$
by (*simp add*: *algebra-simps*)
then have *Big-Far-9-2* γ' $(l-m)$
using *False big* ‹$\gamma' \leq \gamma$› γ ‹$m<l$›
by (*simp add*: *Big-Far-9-1-def*)
then show *Big-Far-9-2* $((l-m)\ /\ (real\ k + real\ (l-m)))\ (l-m)$
by (*simp add*: γ'*-def* ‹$m < l$› *add-diff-eq less-or-eq-imp-le*)
show $(l-m)\ /\ (real\ k + real\ (l-m)) \leq 1/10$
using γ γ*-def* ‹$m < l$› **by** *fastforce*
show $0 \leq \eta$
using ‹$0 < \eta$› **by** *linarith*
show $\eta \leq (l-m)\ /\ (real\ k + real\ (l-m))\ /\ 15$
using *mult-right-mono* [*OF* ‹$\gamma' \leq \gamma$›, *of* ξ]
by (*simp add*: η*-def* γ'*-def* ‹$m < l$› ξ*-def add-diff-eq less-or-eq-imp-le mult.commute*)
qed
qed
with *no-RedU-K* **obtain** K **where** $K \subseteq U$ *UBB.size-clique* $(l-m)$ K *BlueU*
by (*meson UBB.size-clique-def*)
then show *False*
using *no-Blue-K extend-Blue-clique VUU*
unfolding *UBB.size-clique-def size-clique-def BlueU-def*

```
    by (metis Int-subset-iff all-edges-subset-iff-clique)
  qed
qed

end

end
```

9 An exponential improvement closer to the diagonal

```
theory Closer-To-Diagonal
  imports Far-From-Diagonal

begin
```

9.1 Lemma 10.2

```
context P0-min
begin

lemma error-10-2:
  assumes μ / real d > 1/200
  shows ∀∞k. ok-fun-95b k + μ ∗ real k / real d ≥ k/200
proof −
  have d>0 μ>0
    using assms by (auto simp: divide-simps split: if-split-asm)
  then have ∗: real k ≤ μ ∗ (real k ∗ 200) / real d for k
    using assms by (fastforce simp: divide-simps less-eq-real-def)
  have ∀∞k. |ok-fun-95b k| ≤ (μ/d − 1/200) ∗ k
    using ok-fun-95b assms unfolding smallo-def
    by (auto dest!: spec [where x = μ/d])
  then show ?thesis
    apply eventually-elim
    using assms ‹d>0› ∗
    by (simp add: algebra-simps not-less abs-if add-increasing split: if-split-asm)
qed
```

The "sufficiently large" assumptions are problematical. The proof's calculation for $(3::'a) \;/\; (20::'a) < \gamma$ is sharp. We need a finite gap for the limit to exist. We can get away with 1/300.

```
definition x320::real where x320 ≡ 3/20 + 1/300

lemma error-10-2-True: ∀∞k. ok-fun-95b k + x320 ∗ real k / real 30 ≥ k/200
  unfolding x320-def
  by (intro error-10-2) auto

lemma error-10-2-False: ∀∞k. ok-fun-95b k + (1/10) ∗ real k / real 15 ≥ k/200
```

by (*intro error-10-2*) *auto*

definition *Big-Closer-10-2* $\equiv$ $\lambda\mu$ *l*. *Big-Far-9-3* μ *l* $\wedge$ *Big-Far-9-5* μ *l*
$\wedge$ ($\forall k\geq l$. *ok-fun-95b k* + (*if* μ > *x320 then* μ**k*/*30 else* μ**k*/*15*) $\geq$ *k*/*200*)

lemma *Big-Closer-10-2*:
assumes *1*/*10*$\leq\mu$*1* μ*1*<*1*
shows $\forall^\infty l$. $\forall\mu$. *1*/*10* $\leq \mu \wedge \mu \leq \mu$*1* $\longrightarrow$ *Big-Closer-10-2* μ *l*
proof –
have *T*: $\forall^\infty l$. $\forall k\geq l$. ($\forall\mu$. *x320* $\leq \mu \wedge \mu \leq \mu$*1* $\longrightarrow$ *k*/*200* $\leq$ *ok-fun-95b k* + μ**k* / *real 30*)
using *assms*
apply (*intro eventually-all-ge-at-top eventually-all-geI0 error-10-2-True*)
apply (*auto simp*: *mult-right-mono elim*!: *order-trans*)
done
have *F*: $\forall^\infty l$. $\forall k\geq l$. ($\forall\mu$. *1*/*10* $\leq \mu \wedge \mu \leq \mu$*1* $\longrightarrow$ *k*/*200* $\leq$ *ok-fun-95b k* + μ**k* / *real 15*)
using *assms*
apply (*intro eventually-all-ge-at-top eventually-all-geI0 error-10-2-False*)
by (*smt* (*verit*, *ccfv-SIG*) *divide-right-mono mult-right-mono of-nat-0-le-iff*)
have $\forall^\infty l$. $\forall k\geq l$. ($\forall\mu$. *1*/*10* $\leq \mu \wedge \mu \leq \mu$*1* $\longrightarrow$ *k*/*200* $\leq$ *ok-fun-95b k* + (*if* μ > *x320 then* μ**k*/*30 else* μ**k*/*15*))
using *assms*
apply (*split if-split*)
unfolding *eventually-conj-iff all-imp-conj-distrib all-conj-distrib*
by (*force intro*: *eventually-mono* [*OF T*] *eventually-mono* [*OF F*])
then show *?thesis*
using *assms Big-Far-9-3*[*of 1*/*10*] *Big-Far-9-5*[*of 1*/*10*]
unfolding *Big-Closer-10-2-def eventually-conj-iff all-imp-conj-distrib*
by (*force simp*: *elim*!: *eventually-mono*)
qed

end

A little tricky to express since the Book locale assumes that there are no cliques in the original graph (page 10). So it's a contrapositive

lemma (**in** *Book'*) *Closer-10-2-aux*:
assumes *0*: *real* (*card X0*) $\geq$ *nV*/*2 card Y0* $\geq$ *nV div 2 p0* $\geq$ *1*$-\gamma$
— These are the assumptions about the red density of the graph
assumes γ: *1*/*10* $\leq \gamma$ $\gamma \leq$ *1*/*5*
assumes *nV*: *real nV* $\geq$ *exp* ($-$*k*/*200*) * (*k*+*l choose l*)
assumes *big*: *Big-Closer-10-2* γ *l*
shows *False*
proof –
define $\mathcal{R}$ **where** $\mathcal{R}$ $\equiv$ *Step-class* {*red-step*}
define *t* **where** *t* $\equiv$ *card* $\mathcal{R}$
define δ::*real* **where** δ $\equiv$ *1*/*200*
have γ*01*: *0* < γ γ < *1*

using *ln0 l-le-k* **by** (*auto simp*: *γ-def*)
have *t*<*k*
unfolding *t-def* *ℛ-def* **using** *γ01 red-step-limit* **by** *blast*
have *big93*: *Big-Far-9-3 γ l*
using *big* **by** (*auto simp*: *Big-Closer-10-2-def Big-Far-9-2-def*)
have *t23*: $t \geq 2*k\ /\ 3$
unfolding *t-def* *ℛ-def*
proof (*rule Far-9-3*)
have *min* (*1/200*) (*l* / (*real k* + *real l*) / *20*) = *1/200*
using *γ ln0* **by** (*simp add*: *γ-def*)
then show *exp* (− *min* (*1/200*) (*γ* / *20*) ∗ *real k*) ∗ *real* (*k*+*l choose l*) ≤ *nV*
using *nV divide-real-def inverse-eq-divide minus-mult-right mult.commute γ-def*
by (*metis of-int-of-nat-eq of-int-minus*)
show *1/4* ≤ *p0*
using *γ 0* **by** *linarith*
show *Big-Far-9-3 γ l*
using *γ-def big93* **by** *blast*
qed (*use assms γ-def* **in** *auto*)

have *card* (*Yseq halted-point*) ≥
exp (−*δ* ∗ *k* + *ok-fun-95b k*) ∗ (*1*−*γ*) *powr* (*γ*∗*t* / (*1*−*γ*)) ∗ ((*1*−*γ*)/(*1*−*γ*))^*t*
∗ *exp* (*γ* ∗ (*real t*)^2 / (*2*∗*k*)) ∗ (*k*−*t*+*l choose l*)
proof (*rule order-trans* [*OF* - *Far-9-5*])
show *exp* (−*δ* ∗ *k*) ∗ *real* (*k*+*l choose l*) ≤ *real nV*
using *nV* **by** (*auto simp*: *δ-def*)
show *1/2* ≤ *1* − *γ* − *0*
using *divide-le-eq-1 l-le-k γ-def* **by** *fastforce*
next
show *Big-Far-9-5 γ l*
using *big* **by** (*simp add*: *Big-Closer-10-2-def Big-Far-9-2-def γ-def*)
qed (*use 0 kn0* **in** ‹*auto simp flip*: *t-def γ-def ℛ-def*›)
then have *52*: *card* (*Yseq halted-point*) ≥
exp (−*δ* ∗ *k* + *ok-fun-95b k*) ∗ (*1*−*γ*) *powr* (*γ*∗*t* / (*1*−*γ*)) ∗ *exp* (*γ* ∗ (*real t*)^2 / (*2*∗*k*)) ∗ (*k*−*t*+*l choose l*)
using *γ* **by** *simp*

define *gamf* **where** *gamf* ≡ *λx*::*real*. (*1*−*x*) *powr* (*1*/(*1*−*x*))
have *deriv-gamf*: ∃ *y*. *DERIV gamf x* :> *y* ∧ *y* ≤ *0* **if** *0*<*a* *a*≤*x* *x*≤*b* *b*<*1* **for** *a b x*
unfolding *gamf-def*
using *that ln-less-self*[*of 1*−*x*]
by (*force intro*!: *DERIV-powr derivative-eq-intros simp*: *divide-simps mult-le-0-iff simp del*: *ln-less-self*)
have (*1*−*γ*) *powr* (*γ*∗*t* / (*1*−*γ*)) ∗ *exp* (*γ* ∗ (*real t*)^2 / (*2*∗*k*)) ≥ *exp* (*δ*∗*k* − *ok-fun-95b k*)
proof (*cases γ* > *x320*)
case *True*

then have *ok-fun-95b k* + γ**k* / *30* ≥ *k*/*200*
  using *big l-le-k* **by** (*auto simp*: *Big-Closer-10-2-def Big-Far-9-2-def*)
with *True kn0* **have** δ ∗ *k* − *ok-fun-95b k* ≤ (γ/*30*) ∗ *k*
  by (*simp add*: δ-*def*)
also have ... ≤ *3* ∗ γ ∗ (*real t*)^2 / (*40*∗*k*)
  using *True mult-right-mono* [*OF mult-mono* [*OF t23 t23*], *of 3*∗γ / (*40*∗*k*)] ‹*k*>*0*›
  by (*simp add*: *power2-eq-square x320-def*)
finally have †: δ∗*k* − *ok-fun-95b k* ≤ *3* ∗ γ ∗ (*real t*)^2 / (*40*∗*k*) .

have *gamf* γ ≥ *gamf* (*1*/*5*)
  by (*smt* (*verit*, *best*) *DERIV-nonpos-imp-nonincreasing*[*of* γ *1*/*5 gamf*] γ γ*01 deriv-gamf divide-less-eq-1*)
moreover have *ln* (*gamf* (*1*/*5*)) ≥ −*1*/*3* + *1*/*20*
  unfolding *gamf-def* **by** (*approximation 10*)
moreover have *gamf* (*1*/*5*) > *0*
  by (*simp add*: *gamf-def*)
ultimately have *gamf* γ ≥ *exp* (−*1*/*3* + *1*/*20*)
  using *ln-ge-iff* **by** *auto*
from *powr-mono2* [*OF - - this*]
have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ≥ *exp* (−*17*/*60*) *powr* (γ∗*t*)
  unfolding *gamf-def* **using** γ*01 powr-powr* **by** *fastforce*
from *mult-left-mono* [*OF this*, *of exp* (γ ∗ (*real t*)^2 / (*2*∗*k*))]
have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2*∗*k*)) ≥ *exp* (−*17*/*60* ∗ (γ∗*t*) + (γ ∗ (*real t*)^2 / (*2*∗*k*)))
  by (*smt* (*verit*) *mult.commute exp-add exp-ge-zero exp-powr-real*)
moreover have (−*17*/*60* ∗ (γ∗*t*) + (γ ∗ (*real t*)^2 / (*2*∗*k*))) ≥ (*3*∗γ ∗ (*real t*)^2 / (*40*∗*k*))
  using *t23* ‹*k*>*0*› ‹γ>*0*› **by** (*simp add*: *divide-simps eval-nat-numeral*)
ultimately have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2*∗*k*)) ≥ *exp* (*3*∗γ ∗ (*real t*)^2 / (*40*∗*k*))
  by (*smt* (*verit*) *exp-mono*)
with † **show** *?thesis*
  by (*smt* (*verit*, *best*) *exp-le-cancel-iff*)
next
case *False*
then have *ok-fun-95b k* + γ∗*k*/*15* ≥ *k*/*200*
  using *big l-le-k* **by** (*auto simp*: *Big-Closer-10-2-def Big-Far-9-2-def*)
with *kn0* **have** δ ∗ *k* − *ok-fun-95b k* ≤ (γ/*15*) ∗ *k*
  by (*simp add*: δ-*def x320-def*)
also have ... ≤ *3* ∗ γ ∗ (*real t*)^2 / (*20*∗*k*)
  using γ *mult-right-mono* [*OF mult-mono* [*OF t23 t23*], *of 3*∗γ / (*40*∗*k*)] *kn0*
  by (*simp add*: *power2-eq-square field-simps*)
finally have †: δ∗*k* − *ok-fun-95b k* ≤ *3* ∗ γ ∗ (*real t*)^2 / (*20*∗*k*) .

have *gamf* γ ≥ *gamf x320*
  using *False* γ
  by (*intro DERIV-nonpos-imp-nonincreasing*[*of* γ *x320 gamf*] *deriv-gamf*)
    (*auto simp*: *x320-def*)

moreover have *ln* (*gamf x320*) ≥ −*1*/*3* + *1*/*10*
unfolding *gamf-def x320-def* **by** (*approximation 6*)
moreover have *gamf x320* > *0*
by (*simp add*: *gamf-def x320-def*)
ultimately have *gamf* γ ≥ *exp* (−*1*/*3* + *1*/*10*)
using *ln-ge-iff* **by** *auto*
from *powr-mono2* [*OF - - this*]
have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ≥ *exp* (−*7*/*30*) *powr* (γ∗*t*)
unfolding *gamf-def* **using** γ*01 powr-powr* **by** *fastforce*
from *mult-left-mono* [*OF this*, *of exp* (γ ∗ (*real t*)^2 / (*2*∗*k*))]
have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2*∗*k*)) ≥ *exp* (−*7*/*30* ∗ (γ∗*t*) + (γ ∗ (*real t*)^2 / (*2*∗*k*)))
by (*smt* (*verit*) *mult.commute exp-add exp-ge-zero exp-powr-real*)
moreover have (−*7*/*30* ∗ (γ∗*t*) + (γ ∗ (*real t*)^2 / (*2*∗*k*))) ≥ (*3*∗γ ∗ (*real t*)^2 / (*20*∗*k*))
using *t23* ‹*k*>*0*› ‹γ>*0*› **by** (*simp add*: *divide-simps eval-nat-numeral*)
ultimately have (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2*∗*k*)) ≥ *exp* (*3*∗γ ∗ (*real t*)^2 / (*20*∗*k*))
by (*smt* (*verit*) *exp-mono*)
with † **show** *?thesis*
by (*smt* (*verit*, *best*) *exp-le-cancel-iff*)
qed
then have *1* ≤ *exp* (−δ∗*k* + *ok-fun-95b k*) ∗ (*1*−γ) *powr* (γ ∗ *t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2* ∗ *k*))
by (*simp add*: *exp-add exp-diff mult-ac pos-divide-le-eq*)
then have (*k*−*t*+*l choose l*) ≤
exp (−δ ∗ *k* + *ok-fun-95b k*) ∗ (*1*−γ) *powr* (γ∗*t* / (*1*−γ)) ∗ *exp* (γ ∗ (*real t*)^2 / (*2*∗*k*)) ∗ (*k*−*t*+*l choose l*)
by *auto*
with *52* **have** (*k*−*t*+*l choose l*) ≤ *card* (*Yseq halted-point*) **by** *linarith*
then show *False*
using *Off-diagonal-conclusion* **by** (*simp flip*: *R-def t-def*)
qed

Material that needs to be proved **outside** the book locales

lemma (**in** *No-Cliques*) *Closer-10-2*:
fixes γ::*real*
defines γ ≡ *l* / (*real k* + *real l*)
assumes *nV*: *real nV* ≥ *exp* (− *real k*/*200*) ∗ (*k*+*l choose l*)
assumes *gd*: *graph-density Red* ≥ *1*−γ **and** *p0-min-OK*: *p0-min* ≤ *1*−γ
assumes *big*: *Big-Closer-10-2* γ *l* **and** *l*≤*k*
assumes γ: *1*/*10* ≤ γ γ ≤ *1*/*5*
shows *False*
proof −
obtain *X0 Y0* **where** *l*≥*2* **and** *card-X0*: *card X0* ≥ *nV*/*2*
and *card-Y0*: *card Y0* = *gorder div 2*
and *X0-def*: *X0* = *V* \ *Y0* **and** *Y0*⊆*V*
and *gd-le*: *graph-density Red* ≤ *gen-density Red X0 Y0*
and *Book′ V E p0-min Red Blue l k* γ *X0 Y0*

using *to-Book′ assms order.trans ln0* **by** *blast*
then interpret *Book′ V E p0-min Red Blue l k* γ *X0 Y0*
by *blast*
show *False*
proof (*intro Closer-10-2-aux*)
show $1 - \gamma \le p0$
using *X0-def* γ*-def gd gd-le gen-density-commute p0-def* **by** *auto*
qed (*use assms card-X0 card-Y0* **in** *auto*)
qed

9.2 Theorem 10.1

context *P0-min*
begin

definition *Big101a* $\equiv \lambda k.\ 2 + real\ k\ /\ 2 \le exp\ (of\text{-}int\lfloor k/10 \rfloor * 2 - k/200)$

definition *Big101b* $\equiv \lambda k.\ (real\ k)^2 - 10 * real\ k > (k/10) * real(10 + 9*k)$

The proof considers a smaller graph, so l needs to be so big that the smaller l' will be big enough.

definition *Big101c* $\equiv \lambda\gamma 0\ l.\ \forall l'\ \gamma.\ l' \ge nat\ \lfloor 2/5 * l \rfloor \longrightarrow \gamma 0 \le \gamma \longrightarrow \gamma \le 1/10 \longrightarrow Big\text{-}Far\text{-}9\text{-}1\ \gamma\ l'$

definition *Big101d* $\equiv \lambda l.\ (\forall l'\ \gamma.\ l' \ge nat\ \lfloor 2/5 * l \rfloor \longrightarrow 1/10 \le \gamma \longrightarrow \gamma \le 1/5 \longrightarrow Big\text{-}Closer\text{-}10\text{-}2\ \gamma\ l')$

definition *Big-Closer-10-1* $\equiv \lambda\gamma 0\ l.\ l \ge 9 \wedge (\forall k \ge l.\ Big101c\ \gamma 0\ k \wedge Big101d\ k \wedge Big101a\ k \wedge Big101b\ k)$

lemma *Big-Closer-10-1-upward*: $[\![Big\text{-}Closer\text{-}10\text{-}1\ \gamma 0\ l;\ l \le k;\ \gamma 0 \le \gamma]\!] \Longrightarrow Big\text{-}Closer\text{-}10\text{-}1\ \gamma\ k$
unfolding *Big-Closer-10-1-def Big101c-def* **by** (*meson order.trans*)

The need for $\gamma 0$ is unfortunate, but it seems simpler to hide the precise value of this term in the main proof.

lemma *Big-Closer-10-1*:
fixes $\gamma 0$*::real*
assumes $\gamma 0 > 0$
shows $\forall^\infty l.\ Big\text{-}Closer\text{-}10\text{-}1\ \gamma 0\ l$
proof −
have *a*: $\forall^\infty k.\ Big101a\ k$
unfolding *Big101a-def* **by** *real-asymp*
have *b*: $\forall^\infty k.\ Big101b\ k$
unfolding *Big101b-def* **by** *real-asymp*
have *c*: $\forall^\infty l.\ Big101c\ \gamma 0\ l$
proof −
have $\forall^\infty l.\ \forall \gamma.\ \gamma 0 \le \gamma \wedge \gamma \le 1/10 \longrightarrow Big\text{-}Far\text{-}9\text{-}1\ \gamma\ l$
using *Big-Far-9-1* ‹$\gamma 0 > 0$› *eventually-sequentially order.trans* **by** *blast*

then obtain *N* **where** *N*: $\forall l \geq N.\ \forall \gamma.\ \gamma 0 \leq \gamma \wedge \gamma \leq 1/10 \longrightarrow$ *Big-Far-9-1* $\gamma\ l$
using *eventually-sequentially* **by** *auto*
define *M* **where** $M \equiv nat\lceil 5*N\ /\ 2\rceil$
have $nat\lfloor (2/5) * l\rfloor \geq N$ **if** $l \geq M$ **for** *l*
using *that assms* **by** (*simp add*: *M-def le-nat-floor*)
with *N* **have** $\forall l \geq M.\ \forall l'\ \gamma.\ nat\lfloor (2/5) * l\rfloor \leq l' \longrightarrow \gamma 0 \leq \gamma \wedge \gamma \leq 1/10 \longrightarrow$ *Big-Far-9-1* $\gamma\ l'$
by (*meson order.trans*)
then show *?thesis*
by (*auto simp*: *Big101c-def eventually-sequentially*)
qed
have *d*: $\forall^{\infty} l.$ *Big101d l*
proof −
have $\forall^{\infty} l.\ \forall \gamma.\ 1/10 \leq \gamma \wedge \gamma \leq 1/5 \longrightarrow$ *Big-Closer-10-2* $\gamma\ l$
using *assms Big-Closer-10-2* [*of 1/5*] **by** *linarith*
then obtain *N* **where** *N*: $\forall l \geq N.\ \forall \gamma.\ 1/10 \leq \gamma \wedge \gamma \leq 1/5 \longrightarrow$ *Big-Closer-10-2* $\gamma\ l$
using *eventually-sequentially* **by** *auto*
define *M* **where** $M \equiv nat\lceil 5*N\ /\ 2\rceil$
have $nat\lfloor (2/5) * l\rfloor \geq N$ **if** $l \geq M$ **for** *l*
using *that assms* **by** (*simp add*: *M-def le-nat-floor*)
with *N* **have** $\forall l \geq M.\ \forall l'\ \gamma.\ l' \geq nat\ \lfloor 2/5 * l\rfloor \longrightarrow 1/10 \leq \gamma \wedge \gamma \leq 1/5 \longrightarrow$ *Big-Closer-10-2* $\gamma\ l'$
by (*smt* (*verit*, *ccfv-SIG*) *of-nat-le-iff*)
then show *?thesis*
by (*auto simp*: *eventually-sequentially Big101d-def*)
qed
show *?thesis*
using *a b c d eventually-all-ge-at-top eventually-ge-at-top*
unfolding *Big-Closer-10-1-def eventually-conj-iff all-imp-conj-distrib*
by *blast*
qed

The strange constant $\gamma 0$ is needed for the case where we consider a subgraph; see near the end of this proof

theorem *Closer-10-1*:
fixes *l k*::*nat*
fixes $\delta\ \gamma$::*real*
defines $\gamma \equiv real\ l\ /\ (real\ k + real\ l)$
defines $\delta \equiv \gamma/40$
defines $\gamma 0 \equiv min\ \gamma\ (0.07)$ — Since $36 \leq k$, the lower bound $1\ /\ (10::'a) - 1\ /\ (36::'a)$ works
assumes *big*: *Big-Closer-10-1* $\gamma 0\ l$
assumes γ: $\gamma \leq 1/5$
assumes *p0-min-101*: $p0\text{-}min \leq 1 - 1/5$
shows $RN\ k\ l \leq exp\ (-\delta * k + 3) * (k+l\ choose\ l)$
proof (*rule ccontr*)
assume *non*: $\neg\ RN\ k\ l \leq exp\ (-\delta * k + 3) * (k+l\ choose\ l)$

have *l≤k*
using *γ-def γ nat-le-real-less* **by** *fastforce*
moreover have *l≥9*
using *big* **by** (*simp add*: *Big-Closer-10-1-def*)
ultimately have *l>0 k>0 l≥3* **by** *linarith+*
then have *l4k*: *4∗l ≤ k*
using *γ* **by** (*auto simp*: *γ-def divide-simps*)
have *k≥36*
using ‹*l≥9*› *l4k* **by** *linarith*
have *exp-gt21*: *exp* (*x* + *2*) > *exp* (*x* + *1*) **for** *x*::*real*
by *auto*
have *exp2*: *exp* (*2*::*real*) = *exp 1* ∗ *exp 1*
by (*simp add*: *mult-exp-exp*)
have *Big91-I*:⋀*l′ μ*. ⟦*l′* ≥ *nat* ⌊*2*/*5* ∗ *l*⌋; *γ0* ≤ *μ*; *μ* ≤ *1*/*10*⟧ ⟹ *Big-Far-9-1 μ l′*
using *big* **by** (*meson Big101c-def Big-Closer-10-1-def order.refl*)
show *False*
proof (*cases γ* ≤ *1*/*10*)
case *True*
have *γ>0*
using ‹*0* < *l*› *γ-def* **by** *auto*
have *RN k l* ≤ *exp* (−*δ*∗*k* + *1*) ∗ (*k*+*l choose l*)
proof (*intro order.trans* [*OF Far-9-1*] *strip*)
show *Big-Far-9-1* (*l* / (*real k* + *real l*)) *l*
proof (*intro Big91-I*)
show *l* ≥ *nat* ⌊*2*/*5* ∗ *l*⌋
by *linarith*
qed (*use True γ0-def γ-def* **in** *auto*)
next
show *exp* (− (*l* / (*k* + *real l*) / *20*) ∗ *k* + *1*) ∗ (*k*+*l choose l*) ≤ *exp* (−*δ*∗*k* + *1*) ∗ (*k*+*l choose l*)
by (*smt* (*verit*, *best*) ‹*0* < *γ*› *γ-def δ-def exp-mono frac-le mult-right-mono of-nat-0-le-iff*)
qed (*use* ‹*l≥9*› *p0-min-101 True γ-def* **in** *auto*)
then show *False*
using *non exp-gt21* **by** (*smt* (*verit*, *ccfv-SIG*) *mult-right-mono of-nat-0-le-iff*)
next
case *False*
with ‹*l>0*› **have** *γ>0 γ>1*/*10* **and** *k9l*: *k* < *9*∗*l*
by (*auto simp*: *γ-def*)
— Much overlap with the proof of 9.2, but key differences too
define *U-lower-bound-ratio* **where**
U-lower-bound-ratio ≡ *λm*. (∏ *i*<*m*. (*l* − *real i*) / (*k*+*l* − *real i*))
define *n* **where** *n* ≡ *nat*⌈*RN k l* − *1*⌉
have *k≥12*
using *l4k* ‹*l≥3*› **by** *linarith*
have *exp 1* / (*exp 1* − *2*) < (*12*::*real*)
by (*approximation 5*)
also have *RN12*: … ≤ *RN k l*

```
      by (meson RN-3plus' ‹l≥3› ‹k≥12› le-trans numeral-le-real-of-nat-iff)
    finally have exp 1 / (exp 1 − 2) < RN k l .
    moreover have n < RN k l
      using RN12 by (simp add: n-def)
    moreover have 2 < exp (1::real)
      by (approximation 5)
    ultimately have nRNe: n/2 > RN k l / exp 1
      by (simp add: n-def field-split-simps)

    have (k+l choose l) / exp (−3 + δ∗k) < RN k l
     by (smt (verit) divide-inverse exp-minus mult-minus-left mult-of-nat-commute
non)
    then have (k+l choose l) < (RN k l / exp 2) ∗ exp (δ∗k − 1)
      by (simp add: divide-simps exp-add exp-diff flip: exp-add)
    also have ... ≤ (n/2) ∗ exp (δ∗k − 2)
      using nRNe by (simp add: divide-simps exp-diff)
    finally have n2exp-gt': (n/2) ∗ exp (δ∗k) > (k+l choose l) ∗ exp 2
     by (metis exp-diff exp-gt-zero linorder-not-le pos-divide-le-eq times-divide-eq-right)
    then have n2exp-gt: (n/2) ∗ exp (δ∗k) > (k+l choose l)
      by (smt (verit, best) mult-le-cancel-left1 of-nat-0-le-iff one-le-exp-iff)
    then have nexp-gt: n ∗ exp (δ∗k) > (k+l choose l)
      using less-le-trans linorder-not-le by force

    define V where V ≡ {..<n}
    define E where E ≡ all-edges V
    interpret Book-Basis V E
   proof qed (auto simp: V-def E-def comp-sgraph.wellformed comp-sgraph.two-edges)
    have [simp]: nV = n
      by (simp add: V-def)
    then obtain Red Blue
      where Red-E: Red ⊆ E and Blue-def: Blue = E−Red
        and no-Red-K: ¬ (∃K. size-clique k K Red)
        and no-Blue-K: ¬ (∃K. size-clique l K Blue)
      by (metis ‹n < RN k l› less-RN-Red-Blue)
     have Blue-E: Blue ⊆ E and disjnt-Red-Blue: disjnt Red Blue and Blue-eq:
Blue = all-edges V − Red
      using complete by (auto simp: Blue-def disjnt-iff E-def)
    define is-good-clique where
      is-good-clique ≡ λi K. clique K Blue ∧ K ⊆ V
                          ∧ card (V ∩ (⋂w∈K. Neighbours Blue w))
                          ≥ i ∗ U-lower-bound-ratio (card K) − card K
    have is-good-card: card K < l if is-good-clique i K for i K
      using no-Blue-K that unfolding is-good-clique-def
      by (metis nat-neq-iff size-clique-def size-clique-smaller)
    define max-m where max-m ≡ Suc (nat ⌊l − k/9⌋)
    define GC where GC ≡ {C. is-good-clique n C ∧ card C ≤ max-m}
    have maxm-bounds: l − k/9 ≤ max-m max-m ≤ l+1 − k/9 max-m > 0
      using k9l unfolding max-m-def by linarith+
    then have GC ≠ {}
```

by (*auto simp*: *GC-def is-good-clique-def U-lower-bound-ratio-def E-def V-def intro*: *exI* [**where** *x*={}])
have *GC* ⊆ *Pow V*
by (*auto simp*: *is-good-clique-def GC-def*)
then have *finite GC*
by (*simp add*: *finV finite-subset*)
then obtain *W* **where** *W* ∈ *GC* **and** *MaxW*: *Max* (*card* ' *GC*) = *card W*
using ‹*GC* ≠ {}› *obtains-MAX* **by** *blast*
then have *53*: *is-good-clique n W*
using *GC-def* **by** *blast*
then have *W*⊆*V*
by (*auto simp*: *is-good-clique-def*)

define *m* **where** *m* ≡ *card W*
define γ′ **where** γ′ ≡ (*l* − *real m*) / (*k*+*l*−*real m*)

have *max53*: ¬ (*is-good-clique n* (*insert x W*) ∧ *card* (*insert x W*) ≤ *max-m*) **if** *x*∈*V**W* **for** *x*
proof — Setting up the case analysis for γ′
assume *x*: *is-good-clique n* (*insert x W*) ∧ *card* (*insert x W*) ≤ *max-m*
then have *card* (*insert x W*) = *Suc* (*card W*)
using *finV is-good-clique-def finite-subset that* **by** *fastforce*
with *x* ‹*finite GC*› **have** *Max* (*card* ' *GC*) ≥ *Suc* (*card W*)
by (*metis* (*no-types*, *lifting*) *GC-def Max-ge finite-imageI image-iff mem-Collect-eq*)
then show *False*
by (*simp add*: *MaxW*)
qed
then have *clique-cases*: *m* < *max-m* ∧ (∀ *x*∈*V**W*. ¬ *is-good-clique n* (*insert x W*)) ∨ *m* = *max-m*
using *GC-def* ‹*W* ∈ *GC*› ‹*W* ⊆ *V*› *finV finite-subset m-def* **by** *fastforce*

have *Red-Blue-RN*: ∃ *K* ⊆ *X*. *size-clique m K Red* ∨ *size-clique n K Blue*
if *card X* ≥ *RN m n X*⊆*V* **for** *m n* **and** *X*
using *partn-lst-imp-is-clique-RN* [*OF is-Ramsey-number-RN* [*of m n*]] *finV that*
unfolding *is-clique-RN-def size-clique-def clique-indep-def Blue-eq*
by (*metis clique-iff-indep finite-subset subset-trans*)
define *U* **where** *U* ≡ *V* ∩ (⋂ *w*∈*W*. *Neighbours Blue w*)
have *RN k l* > *0*
by (*metis RN-eq-0-iff gr0I* ‹*k*>*0*› ‹*l*>*0*›)
with ‹*n* < *RN k l*› **have** *n-less*: *n* < (*k*+*l choose l*)
by (*metis add.commute RN-commute RN-le-choose le-trans linorder-not-less*)

have γ′ > *0*
using *is-good-card* [*OF 53*] **by** (*simp add*: γ′*-def m-def*)
have *finite W*
using ‹*W* ⊆ *V*› *finV finite-subset* **by** (*auto simp*: *V-def*)
have *U* ⊆ *V*
by (*force simp*: *U-def*)

```
  then have VUU: V ∩ U = U
    by blast
  have disjnt U W
    using Blue-E not-own-Neighbour unfolding E-def V-def U-def disjnt-iff by
blast
  have m<l
    using 53 is-good-card m-def by blast
  have γ' ≤ 1
    using ‹m<l› by (simp add: γ'-def divide-simps)

  have cardU: n ∗ U-lower-bound-ratio m ≤ m + card U
    using 53 VUU unfolding is-good-clique-def m-def U-def by force
  have clique-W: size-clique m W Blue
    using 53 is-good-clique-def m-def size-clique-def V-def by blast
  have prod-gt0: U-lower-bound-ratio m > 0
    unfolding U-lower-bound-ratio-def using ‹m<l› by (intro prod-pos) auto
 have kl-choose: real(k+l choose l) = (k+l−m choose (l−m)) / U-lower-bound-ratio
m
   unfolding U-lower-bound-ratio-def using kl-choose ‹0 < k› ‹m < l› by blast

  — in both cases below, we find a blue clique of size l − m
  have extend-Blue-clique: ∃K'. size-clique l K' Blue
    if K ⊆ U size-clique (l−m) K Blue for K
  proof −
    have K: card K = l−m clique K Blue
      using that by (auto simp: size-clique-def)
    define K' where K' ≡ K ∪ W
    have card K' = l
      unfolding K'-def
    proof (subst card-Un-disjnt)
      show finite K finite W
        using finV ‹K ⊆ U› ‹U⊆V› finite-subset ‹finite W› that by meson+
      show disjnt K W
        using ‹disjnt U W› ‹K ⊆ U› disjnt-subset1 by blast
      show card K + card W = l
        using K ‹m < l› m-def by auto
    qed
    moreover have clique K' Blue
      using ‹clique K Blue› clique-W ‹K ⊆ U›
      unfolding K'-def size-clique-def U-def
      by (force simp: in-Neighbours-iff insert-commute intro: Ramsey.clique-Un)
    ultimately show ?thesis
      unfolding K'-def size-clique-def using ‹K ⊆ U› ‹U ⊆ V› ‹W ⊆ V› by
auto
  qed

  have γ' ≤ γ
    using ‹m<l› by (simp add: γ-def γ'-def field-simps)
```

consider *m* < *max-m* | *m* = *max-m*
  using *clique-cases* **by** *blast*
then consider *m* < *max-m* $\gamma' \geq 1/10$ | $1/10 - 1/k \leq \gamma' \wedge \gamma' \leq 1/10$
proof *cases*
  case *1*
  then have $\gamma' \geq 1/10$
    using ‹γ>1/10› ‹k>0› *maxm-bounds* **by** (*auto simp*: γ-*def* γ'-*def*)
  with *1 that* **show** *thesis* **by** *blast*
next
  case *2*
  then have γ'-*le110*: $\gamma' \leq 1/10$
    using ‹γ>1/10› ‹k>0› *maxm-bounds* **by** (*auto simp*: γ-*def* γ'-*def*)
  have $1/10 - 1/k \leq \gamma'$
  proof −
    have §: $l-m \geq k/9 - 1$
      using ‹γ>1/10› ‹k>0› *2* **by** (*simp add*: *max-m-def* γ-*def*) *linarith*
    have $1/10 - 1/k \leq 1 - k / (10*k/9 - 1)$
      using γ'-*le110* ‹m<l› ‹k>0› **by** (*simp add*: γ'-*def field-simps*)
    also have ... $\leq 1 - k / (k + l - m)$
      using ‹l≤k› ‹m<l› § **by** (*simp add*: *divide-left-mono*)
    also have ... $= \gamma'$
      using ‹l>0› ‹l≤k› ‹m<l› ‹k>0› **by** (*simp add*: γ'-*def divide-simps*)
    finally show $1/10 - 1$ / *real* $k \leq \gamma'$ **.**
  qed
  with γ'-*le110 that* **show** *thesis*
    by *linarith*
qed
note γ'-*cases* = *this*
have *110*: $1/10 - 1/k \leq \gamma'$
  using γ'-*cases* **by** (*smt* (*verit*, *best*) *divide-nonneg-nonneg of-nat-0-le-iff*)
have $(real\ k)^2 - 10 * real\ k \leq (l-m) * (10 + 9*k)$
  using *110* ‹m<l› ‹k>0›
  by (*simp add*: γ'-*def field-split-simps power2-eq-square*)
with *big* ‹k≥l› **have** $k/10 \leq l-m$
unfolding *Big101b-def Big-Closer-10-1-def* **by** (*smt* (*verit*, *best*) *mult-right-mono of-nat-0-le-iff of-nat-mult*)
then have *k10-lm*: *nat* $\lfloor k/10 \rfloor \leq l - m$
  by *linarith*
have *lm-ge-25*: *nat* $\lfloor 2/5 * l \rfloor \leq l - m$
  using *False l4k k10-lm* **by** *linarith*

— As with 9: a huge effort just to show that U is nontrivial. Proof actually shows its cardinality exceeds a small multiple of l (7/5).

have $l + Suc\ l - q \leq (k+q\ choose\ q) / exp(\delta*k)$
  if *nat*$\lfloor k/10 \rfloor \leq q$ $q \leq l$ **for** *q*
  using *that*
proof (*induction q rule*: *nat-induct-at-least*)
  case *base*
  have †: $0 < 10 + 10 * real\text{-}of\text{-}int\ \lfloor k/10 \rfloor / k$

using ‹k>0› **by** (smt (verit) divide-nonneg-nonneg of-nat-0-le-iff of-nat-int-floor)
have ln9: ln (10::real) ≥ 2
by (approximation 5)
have l + real (Suc l − nat⌊k/10⌋) ≤ 2 + k/2
using l4k **by** linarith
also have ... ≤ exp(of-int⌊k/10⌋ ∗ 2 − k/200)
using big **by** (simp add: Big101a-def Big-Closer-10-1-def ‹l ≤ k›)
also have ... ≤ exp(⌊k/10⌋ ∗ ln(10) − k/200)
by (intro exp-mono diff-mono mult-left-mono ln9) auto
also have ... ≤ exp(⌊k/10⌋ ∗ ln(10)) ∗ exp (−real k/200)
by (simp add: mult-exp-exp)
also have ... ≤ exp(⌊k/10⌋ ∗ ln(10 + (10 ∗ nat⌊k/10⌋) / k)) ∗ exp (−real k/200)
using † **by** (intro mult-mono exp-mono) auto
also have ... ≤ (10 + (10 ∗ nat⌊k/10⌋) / k) ^ nat⌊k/10⌋ ∗ exp (−real k/200)
using † **by** (auto simp: powr-def simp flip: powr-realpow)
also have ... ≤ ((k + nat⌊k/10⌋) / (k/10)) ^ nat⌊k/10⌋ ∗ exp (−real k/200)
using ‹k>0› **by** (simp add: mult.commute add-divide-distrib)
also have ... ≤ ((k + nat⌊k/10⌋) / nat⌊k/10⌋) ^ nat⌊k/10⌋ ∗ exp (−real k/200)
proof (intro mult-mono power-mono divide-left-mono)
show nat⌊k/10⌋ ≤ k/10
by linarith
qed (use ‹k≥36› **in** auto)
also have ... ≤ (k + nat⌊k/10⌋ gchoose nat⌊k/10⌋) ∗ exp (−real k/200)
by (meson exp-gt-zero gbinomial-ge-n-over-k-pow-k le-add2 mult-le-cancel-right-pos of-nat-mono)
also have ... ≤ (k + nat⌊k/10⌋ choose nat⌊k/10⌋) ∗ exp (−real k/200)
by (simp add: binomial-gbinomial)
also have ... ≤ (k + nat⌊k/10⌋ choose nat⌊k/10⌋) / exp (δ ∗ k)
using γ ‹0 < k› **by** (simp add: algebra-simps δ-def exp-minus′ frac-le)
finally show ?case **by** linarith
next
case (Suc q)
then show ?case
apply simp
by (smt (verit) divide-right-mono exp-ge-zero of-nat-0-le-iff)
qed
from ‹m<l› this [of l−m]
have 1 + l + real m ≤ (k+l−m choose (l−m)) / exp δ ^ k
by (simp add: exp-of-nat2-mult k10-lm)
also have ... ≤ (k+l−m choose (l−m)) / exp (δ ∗ k)
by (simp add: exp-of-nat2-mult)
also have ... < U-lower-bound-ratio m ∗ (real n)
proof −
have §: (k+l choose l) / exp (δ ∗ k) < n
by (simp add: less-eq-real-def nexp-gt pos-divide-less-eq)

show *?thesis*
using *mult-strict-left-mono* [*OF* §, *of U-lower-bound-ratio m*] *kl-choose prod-gt0*
by (*auto simp*: *field-simps*)
qed
finally have *U-MINUS-M*: *1+l* < *real n* ∗ *U-lower-bound-ratio m* − *m*
by *argo*
then have *cardU-gt*: *card U* > *l* + *1 card U* > *1*
using *cardU* **by** *linarith+*

show *False*
using *γ′-cases*
proof *cases*
case *1*
— Restricting attention to U
define *EU* **where** *EU* ≡ *E* ∩ *Pow U*
define *RedU* **where** *RedU* ≡ *Red* ∩ *Pow U*
define *BlueU* **where** *BlueU* ≡ *Blue* ∩ *Pow U*
have *RedU-eq*: *RedU* = *EU* \ *BlueU*
using *BlueU-def Blue-def EU-def RedU-def Red-E* **by** *fastforce*
obtain [*iff*]: *finite RedU finite BlueU RedU* ⊆ *EU*
using *BlueU-def EU-def RedU-def E-def V-def Red-E Blue-E fin-edges finite-subset* **by** *blast*
then have *card-EU*: *card EU* = *card RedU* + *card BlueU*
by (*simp add*: *BlueU-def Blue-def Diff-Int-distrib2 EU-def RedU-def card-Diff-subset card-mono*)
then have *card-RedU-le*: *card RedU* ≤ *card EU*
by *linarith*
interpret *UBB*: *Book-Basis U E* ∩ *Pow U p0-min*
proof
fix *e* **assume** *e* ∈ *E* ∩ *Pow U*
with *two-edges* **show** *e* ⊆ *U card e* = *2* **by** *auto*
next
show *finite U*
using ‹*U* ⊆ *V*› **by** (*simp add*: *V-def finite-subset*)
have *x* ∈ *E* **if** *x* ∈ *all-edges U* **for** *x*
using ‹*U* ⊆ *V*› *all-edges-mono that complete E-def* **by** *blast*
then show *E* ∩ *Pow U* = *all-edges U*
using *comp-sgraph.wellformed* ‹*U* ⊆ *V*› **by** (*auto intro*: *e-in-all-edges-ss*)
qed *auto*

have *BlueU-eq*: *BlueU* = *EU* \ *RedU*
using *Blue-eq complete* **by** (*fastforce simp*: *BlueU-def RedU-def EU-def V-def E-def*)
have [*simp*]: *UBB.graph-size* = *card EU*
using *EU-def* **by** *blast*
have *card EU* > *0*
using ‹*card U* > *1*› *UBB.complete* **by** (*simp add*: *EU-def UBB.finV card-all-edges*)

```
have False if UBB.graph-density BlueU > γ′
proof −   — by maximality, etc.; only possible in case 1
  have Nx: Neighbours BlueU x ∩ (U \ {x}) = Neighbours BlueU x for x
    using that by (auto simp: BlueU-eq EU-def Neighbours-def)
  have BlueU ⊆ E ∩ Pow U
    using BlueU-eq EU-def by blast
  with UBB.exists-density-edge-density [of 1 BlueU]
 obtain x where x∈U and x: UBB.graph-density BlueU ≤ UBB.gen-density
BlueU {x} (U\{x})
    by (metis UBB.complete ‹1 < UBB.gorder› card-1-singletonE insertI1
zero-less-one subsetD)
  with that have γ′ ≤ UBB.gen-density BlueU (U\{x}) {x}
    using UBB.gen-density-commute by auto
  then have *: γ′ ∗ (card U − 1) ≤ card (Neighbours BlueU x)
    using ‹BlueU ⊆ E ∩ Pow U› ‹card U > 1› ‹x ∈ U›
     by (simp add: UBB.gen-density-def UBB.edge-card-eq-sum-Neighbours
UBB.finV divide-simps Nx)

  have x: x ∈ V\W
    using ‹x ∈ U› ‹U ⊆ V› ‹disjnt U W› by (auto simp: U-def disjnt-iff)
  moreover
  have is-good-clique n (insert x W)
    unfolding is-good-clique-def
  proof (intro conjI)
    show clique (insert x W) Blue
    proof (intro clique-insert)
      show clique W Blue
        using 53 is-good-clique-def by blast
      show all-edges-betw-un {x} W ⊆ Blue
      using ‹x∈U› by (auto simp: U-def all-edges-betw-un-def insert-commute
in-Neighbours-iff)
    qed (use ‹W ⊆ V› ‹x ∈ V\W› in auto)
  next
    show insert x W ⊆ V
      using ‹W ⊆ V› ‹x ∈ V\W› by auto
  next
    have NB-Int-U: Neighbours Blue x ∩ U = Neighbours BlueU x
      using ‹x ∈ U› by (auto simp: BlueU-def U-def Neighbours-def)
   have ulb-ins: U-lower-bound-ratio (card (insert x W)) = U-lower-bound-ratio
m ∗ γ′
    using ‹x ∈ V\W› ‹finite W› by (simp add: m-def U-lower-bound-ratio-def
γ′-def)
   have n ∗ U-lower-bound-ratio (card (insert x W)) = n ∗ U-lower-bound-ratio
m ∗ γ′
      by (simp add: ulb-ins)
    also have . . . ≤ real (m + card U) ∗ γ′
      using mult-right-mono [OF cardU, of γ′] ‹0 < γ′› by argo
    also have . . . ≤ m + card U ∗ γ′
```

```
          using mult-left-mono [OF ‹γ'≤1›, of m] by (simp add: algebra-simps)
        also have ... ≤ Suc m + γ' * (UBB.gorder − Suc 0)
          using * ‹x ∈ V\W› ‹finite W› ‹1 < UBB.gorder› ‹γ'≤1›
          by (simp add: U-lower-bound-ratio-def algebra-simps)
        also have ... ≤ Suc m + card (V ∩ ⋂ (Neighbours Blue ‘ insert x W))
          using * NB-Int-U finV by (simp add: U-def Int-ac)
        also have ... = real (card (insert x W) + card (V ∩ ⋂ (Neighbours Blue
‘ insert x W)))
          using x ‹finite W› VUU by (auto simp: m-def U-def)
        finally show n * U-lower-bound-ratio (card(insert x W)) − card(insert x
W)
              ≤ card (V ∩ ⋂ (Neighbours Blue ‘ insert x W))
          by simp
      qed
      ultimately show False
        using 1 clique-cases by blast
    qed
    then have *: UBB.graph-density BlueU ≤ γ' by force
    have no-RedU-K: ¬ (∃K. UBB.size-clique k K RedU)
      unfolding UBB.size-clique-def RedU-def
    by (metis Int-subset-iff VUU all-edges-subset-iff-clique no-Red-K size-clique-def)
     have (∃K. UBB.size-clique k K RedU) ∨ (∃K. UBB.size-clique (l−m) K
BlueU)
    proof (rule ccontr)
      assume neg: ¬ ((∃K. UBB.size-clique k K RedU) ∨ (∃K. UBB.size-clique
(l−m) K BlueU))
      interpret UBB-NC: No-Cliques U E ∩ Pow U p0-min RedU BlueU l−m k
      proof
        show BlueU = E ∩ Pow U \ RedU
          using BlueU-eq EU-def by fastforce
      qed (use neg EU-def ‹RedU ⊆ EU› no-RedU-K ‹l≤k› in auto)
      show False
      proof (intro UBB-NC.Closer-10-2)
        have δ ≤ 1/200
          using γ by (simp add: δ-def field-simps)
        then have exp (δ * real k) ≤ exp (real k/200)
          using ‹0 < k› by auto
        then have expexp: exp (δ*k) * exp (− real k/200) ≤ 1
       by (metis divide-minus-left exp-ge-zero exp-minus-inverse mult-right-mono)
          have exp (− real k/200) * (k + (l−m) choose (l−m)) = exp (− real
k/200) * U-lower-bound-ratio m * (k+l choose l)
          using ‹m < l› kl-choose by force
     also have ... < (n/2) * exp (δ*k) * exp (− real k/200) * U-lower-bound-ratio
m
          using n2exp-gt prod-gt0 by auto
        also have ... ≤ (n/2) * U-lower-bound-ratio m
            using mult-left-mono [OF expexp, of (n/2) * U-lower-bound-ratio m]
prod-gt0 by (simp add: mult-ac)
        also have ... ≤ n * U-lower-bound-ratio m − m  — formerly stuck here,
```

due to the "minus m"
using *U-MINUS-M* ‹m < l› **by** *auto*
finally have *exp* (− *real k*/200) ∗ (k + (l−m) *choose* (l−m)) ≤ *UBB.nV*
using *cardU* **by** *linarith*
then show *exp* (− *real k* / 200) ∗ (k + (l−m) *choose* (l−m)) ≤ *UBB.nV*
using ‹m < l› **by** (*simp add*: γ′-*def*)
next
have 1 − γ′ ≤ *UBB.graph-density RedU*
using ∗ *card-EU* ‹*card EU* > 0›
by (*simp add*: *UBB.graph-density-def BlueU-eq field-split-simps split*: *if-split-asm*)
then show 1 − *real* (l−m) / (*real* k + *real* (l−m)) ≤ *UBB.graph-density RedU*
unfolding γ′-*def* **using** ‹m<l› **by** (*smt* (*verit*, *ccfv-threshold*) *less-imp-le-nat of-nat-add of-nat-diff*)
next
show *p0-min* ≤ 1 − *real* (l−m) / (*real* k + *real* (l−m))
using *p0-min-101* ‹γ′≤γ› ‹m < l› γ
by (*smt* (*verit*, *del-insts*) *of-nat-add* γ′-*def less-imp-le-nat of-nat-diff*)
next
have *Big-10-2I*: ⋀l′ μ. ⟦*nat* ⌊2/5 ∗ l⌋ ≤ l′; 1/10 ≤ μ; μ ≤ 1 / 5⟧ ⟹ *Big-Closer-10-2* μ l′
using *big* **by** (*meson Big101d-def Big-Closer-10-1-def order.refl*)
have m ≤ *real* l ∗ (1 − (10/11)∗γ)
using ‹m<l› ‹γ>1/10› ‹γ′≥1/10› γ
apply (*simp add*: γ-*def* γ′-*def field-simps*)
by (*smt* (*verit*, *ccfv-SIG*) *mult.commute mult-left-mono distrib-left*)
then have *real* l − *real* m ≥ (10/11) ∗ γ ∗ l
by (*simp add*: *algebra-simps*)
moreover
have 1/10 ≤ γ′ ∧ γ′ ≤ 1/5
using *mult-mono* [*OF* γ γ] ‹γ′≥1/10› ‹γ′ ≤ γ› γ **by** (*auto simp*: *power2-eq-square*)
ultimately
have *Big-Closer-10-2* γ′ (l−m)
using *lm-ge-25* **by** (*intro Big-10-2I*) *auto*
then show *Big-Closer-10-2* ((l−m) / (*real* k + *real* (l−m))) (l−m)
by (*simp add*: γ′-*def* ‹m < l› *add-diff-eq less-or-eq-imp-le*)
next
show l−m ≤ k
using ‹l ≤ k› **by** *auto*
show (l−m) / (*real* k + *real* (l−m)) ≤ 1/5
using γ γ-*def* ‹m < l› **by** *fastforce*
show 1/10 ≤ (l−m) / (*real* k + *real* (l−m))
using γ′-*def* ‹1/10 ≤ γ′› ‹m < l› **by** *auto*
qed
qed
with *no-RedU-K UBB.size-clique-def* **obtain** K **where** K ⊆ U *UBB.size-clique* (l−m) K *BlueU*

by *meson*
then show *False*
using *no-Blue-K extend-Blue-clique VUU*
unfolding *UBB.size-clique-def size-clique-def BlueU-def*
by (*metis Int-subset-iff all-edges-subset-iff-clique*)
next
case *2*
have *RN k* (*l−m*) ≤ *exp* (− ((*l−m*) / (*k* + *real* (*l−m*)) / *20*) ∗ *k* + *1*) ∗ (*k* + (*l−m*) *choose* (*l−m*))
proof (*intro Far-9-1 strip*)
show *real* (*l−m*) / (*real k* + *real* (*l−m*)) ≤ *1/10*
using *γ′-def 2* ‹*m* < *l*› **by** *auto*
next — here is where we need the specified definition of *γ0*
show *Big-Far-9-1* (*real* (*l−m*) / (*k* + *real* (*l−m*))) (*l−m*)
proof (*intro Big91-I* [*OF lm-ge-25*])
have *0.07* ≤ (*1*::*real*)/*10* − *1/36*
by (*approximation 5*)
also have ... ≤ *1/10* − *1/k*
using ‹*k*≥*36*› **by** (*intro diff-mono divide-right-mono*) *auto*
finally have *7*: *γ′* ≥ *0.07* **using** *110* **by** *linarith*
with ‹*m*<*l*› **show** *γ0* ≤ *real* (*l−m*) / (*real k* + *real* (*l−m*))
by (*simp add*: *γ0-def min-le-iff-disj γ′-def algebra-simps*)
next
show *real* (*l−m*) / (*real k* + *real* (*l−m*)) ≤ *1/10*
using *2* ‹*m*<*l*› **by** (*simp add*: *γ′-def*)
qed
next
show *p0-min* ≤ *1* − *1/10* ∗ (*1* + *1* / *15*)
using *p0-min-101* **by** *auto*
qed
also have ... ≤ *real n* ∗ *U-lower-bound-ratio m* − *m*
proof −
have *γ* ∗ *real k* ≤ *k/5*
using *γ* ‹*0* < *k*› **by** *auto*
also have ... ≤ *γ′* ∗ (*real k* ∗ *2*) + *2*
using *mult-left-mono* [*OF 110*, *of k*∗*2*] ‹*k*>*0*› **by** (*simp add*: *algebra-simps*)
finally have *γ* ∗ *real k* ≤ *γ′* ∗ (*real k* ∗ *2*) + *2* **.**
then have *expexp*: *exp* (*δ* ∗ *real k*) ∗ *exp* (−*γ′*∗*k* / *20* − *1*) ≤ *1*
by (*simp add*: *δ-def flip*: *exp-add*)
have *exp* (−*γ′*∗*k/20* + *1*) ∗ (*k* + (*l−m*) *choose* (*l−m*)) = *exp* (−*γ′*∗*k/20*+*1*) ∗ *U-lower-bound-ratio m* ∗ (*k*+*l choose l*)
using ‹*m* < *l*› *kl-choose* **by** *force*
also have ... < (*n/2*) ∗ *exp* (*δ*∗*k*) ∗ *exp* (−*γ′*∗*k/20* − *1*) ∗ *U-lower-bound-ratio m*
using *n2exp-gt′ prod-gt0* **by** (*simp add*: *exp2 exp-diff exp-minus′ mult-ac pos-less-divide-eq*)
also have ... ≤ (*n/2*) ∗ *U-lower-bound-ratio m*
using *expexp order-le-less prod-gt0* **by** *fastforce*
also have ... ≤ *n* ∗ *U-lower-bound-ratio m* − *m*

```
          using U-MINUS-M ‹m < l› by fastforce
        finally show ?thesis
          using ‹m < l› by (simp add: γ'-def) argo
      qed
      also have ... ≤ card U
        using cardU by auto
      finally have RN k (l−m) ≤ card U by linarith
      then show False
        using Red-Blue-RN ‹U ⊆ V› extend-Blue-clique no-Blue-K no-Red-K by
blast
    qed
  qed
qed

definition ok-fun-10-1 ≡ λγ k. if Big-Closer-10-1 (min γ 0.07) (nat⌈((γ / (1−γ))
∗ k)⌉) then 3 else (γ/40 ∗ k)

lemma ok-fun-10-1:
  assumes 0 < γ γ < 1
  shows ok-fun-10-1 γ ∈ o(real)
proof −
  define γ0 where γ0 ≡ min γ 0.07
  have γ0 > 0
    using assms by (simp add: γ0-def)
  then have ∀∞l. Big-Closer-10-1 γ0 l
    by (simp add: Big-Closer-10-1)
  then obtain l where ⋀l'. l' ≥ l ⟹ Big-Closer-10-1 γ0 l'
    using eventually-sequentially by auto
  moreover
  have nat⌈((γ / (1−γ)) ∗ k)⌉ ≥ l if real k ≥ l/γ − l for k
    using that assms
    by (auto simp: field-simps intro!: le-natceiling-iff)
  ultimately have ∀∞k. Big-Closer-10-1 (min γ 0.07) (nat⌈((γ / (1−γ)) ∗ k)⌉)
    by (smt (verit) γ0-def eventually-sequentially nat-ceiling-le-eq)
  then have ∀∞k. ok-fun-10-1 γ k = 3
    by (simp add: ok-fun-10-1-def eventually-mono)
  then show ?thesis
    by (simp add: const-smallo-real landau-o.small.in-cong)
qed

theorem Closer-10-1-unconditional:
  fixes l k::nat
  fixes δ γ::real
  defines γ ≡ real l / (real k + real l)
  defines δ ≡ γ/40
  assumes γ: 0 < γ γ ≤ 1/5
  assumes p0-min-101: p0-min ≤ 1 − 1/5
  shows RN k l ≤ exp (−δ∗k + ok-fun-10-1 γ k) ∗ (k+l choose l)
proof −
```

```
  define γ0 where γ0 ≡ min γ 0.07
  show ?thesis
  proof (cases Big-Closer-10-1 γ0 l)
    case True
    show ?thesis
      using Closer-10-1 [OF True [unfolded γ0-def γ-def]] assms
      by (simp add: ok-fun-10-1-def γ-def δ-def RN-le-choose')
  next
    case False
    have (nat ⌈γ ∗ k / (1−γ)⌉) ≤ l
      by (simp add: γ-def divide-simps)
    with False Big-Closer-10-1-upward
    have ¬ Big-Closer-10-1 γ0 (nat ⌈γ ∗ k / (1−γ)⌉)
      by blast
    then show ?thesis
      by (simp add: ok-fun-10-1-def δ-def γ0-def RN-le-choose')
  qed
qed

end

end
```

10 From diagonal to off-diagonal

```
theory From-Diagonal
  imports Closer-To-Diagonal

begin
```

10.1 Lemma 11.2

```
definition ok-fun-11-2a ≡ λk. ⌈real k powr (3/4)⌉ ∗ log 2 k

definition ok-fun-11-2b ≡ λμ k. k powr (39/40) ∗ (log 2 μ + 3 ∗ log 2 k)

definition ok-fun-11-2c ≡ λμ k. − k ∗ log 2 (1 − (2 / (1−μ)) ∗ k powr (−1/40))

definition ok-fun-11-2 ≡ λμ k. 2 − ok-fun-71 μ k + ok-fun-11-2a k
    + max (ok-fun-11-2b μ k) (ok-fun-11-2c μ k)

lemma ok-fun-11-2a: ok-fun-11-2a ∈ o(real)
  unfolding ok-fun-11-2a-def
  by real-asymp
```

possibly, the functions that depend upon μ need a more refined analysis to cover a closed interval of possible values. But possibly not, as the text implies $\mu = (2::'a) / (5::'a)$.

```
lemma ok-fun-11-2b: ok-fun-11-2b μ ∈ o(real)
```

unfolding *ok-fun-11-2b-def* **by** *real-asymp*

lemma *ok-fun-11-2c*: *ok-fun-11-2c* $\mu \in o(real)$
unfolding *ok-fun-11-2c-def*
by *real-asymp*

lemma *ok-fun-11-2*:
assumes $0<\mu$ $\mu<1$
shows *ok-fun-11-2* $\mu \in o(real)$
unfolding *ok-fun-11-2-def*
by (*simp add*: *assms const-smallo-real maxmin-in-smallo ok-fun-11-2a ok-fun-11-2b ok-fun-11-2c ok-fun-71 sum-in-smallo*)

definition *Big-From-11-2* $\equiv$
$\lambda\mu$ *k*. *Big-ZZ-8-6* μ *k* $\wedge$ *Big-X-7-1* μ *k* $\wedge$ *Big-Y-6-2* μ *k* $\wedge$ *Big-Red-5-3* μ *k* $\wedge$ *Big-Blue-4-1* μ *k*
$\wedge$ $1 \le \mu\hat{}2 * real\ k \wedge 2\ /\ (1-\mu) * real\ k\ powr\ (-1/40) < 1 \wedge 1/k < 1/2 - 3 * eps\ k$

lemma *Big-From-11-2*:
assumes $0<\mu0$ $\mu0 \le \mu1$ $\mu1<1$
shows $\forall^\infty k.\ \forall \mu.\ \mu \in \{\mu0..\mu1\} \longrightarrow$ *Big-From-11-2* μ *k*
proof −
have *A*: $\forall^\infty k.\ \forall \mu.\ \mu0 \le \mu \wedge \mu \le \mu1 \longrightarrow 1 \le \mu^2 * k$
proof (*intro eventually-all-geI0*)
show $*$: $\forall^\infty x.\ 1 \le \mu0^2 * real\ x$
using ‹$0<\mu0$› **by** *real-asymp*
next
fix *k* μ
assume $1 \le \mu0^2 * real\ k$ **and** $\mu0 \le \mu$ $\mu \le \mu1$
with ‹$0<\mu0$› **show** $1 \le \mu^2 * k$
by (*smt* (*verit*, *ccfv-SIG*) *mult-le-cancel-right of-nat-less-0-iff power-mono*)
qed
have *B*: $\forall^\infty k.\ \forall \mu.\ \mu0 \le \mu \wedge \mu \le \mu1 \longrightarrow 2\ /\ (1-\mu) * k\ powr\ (-1/40) < 1$
proof (*intro eventually-all-geI1*)
show $\forall^\infty k.\ 2\ /\ (1-\mu1) * k\ powr\ (-1/40) < 1$
by *real-asymp*
qed (*use assms* **in** *auto*)
have *C*: $\forall^\infty k.\ 1/k < 1/2 - 3 * eps\ k$
unfolding *eps-def* **by** *real-asymp*
show *?thesis*
unfolding *Big-From-11-2-def*
using *assms Big-ZZ-8-6 Big-X-7-1 Big-Y-6-2 Big-Red-5-3 Big-Blue-4-1 A B C*
by (*simp add*: *eventually-conj-iff all-imp-conj-distrib*)
qed

Simply to prevent issues about the positioning of the function *real*

abbreviation *ratio* $\equiv \lambda\mu\ s\ t.\ \mu * (real\ s + real\ t)\ /\ real\ s$

the text refers to the actual Ramsey number but I don't see how that could work. Theorem 11.1 will define n to be one less than the Ramsey number, hence we add that one back here.

```
lemma (in Book) From-11-2:
  assumes l=k
  assumes big: Big-From-11-2 μ k
  defines R ≡ Step-class {red-step} and S ≡ Step-class {dboost-step}
  defines t ≡ card R and s ≡ card S
  defines nV' ≡ Suc nV
  assumes 0: card X0 ≥ nV div 2 and p0 ≥ 1/2
  shows log 2 nV' ≤ k * log 2 (1/μ) + t * log 2 (1 / (1−μ)) + s * log 2 (ratio
μ s t) + ok-fun-11-2 μ k
proof −
  have big71: Big-X-7-1 μ k and big62: Big-Y-6-2 μ k and big86: Big-ZZ-8-6 μ
k and big53: Big-Red-5-3 μ k
    and big41: Big-Blue-4-1 μ k and bigμ: 1 ≤ μ^2 * real k
    and big-le1: 2 / (1−μ) * real k powr (−1/40) < 1
    using big by (auto simp: Big-From-11-2-def)
  have bigμ1: 1 ≤ μ * real k
    using bigμ μ01
     by (smt (verit, best) mult-less-cancel-right2 mult-right-mono of-nat-less-0-iff
power2-eq-square)
  then have log2μk: log 2 μ + log 2 k ≥ 0
    using kn0 μ01 add-log-eq-powr by auto
  have bigμ2: 1 ≤ μ * (real k)^2
   unfolding power2-eq-square by (smt (verit, ccfv-SIG) bigμ1 μ01 mult-less-cancel-left1
mult-mono')
  define g where g ≡ λk. ⌈real k powr (3/4)⌉ * log 2 k
  have g: g ∈ o(real)
    unfolding g-def by real-asymp
  have bb-gt0: bigbeta > 0
    using big53 bigbeta-gt0 ‹l=k› by blast
  have t < k
    by (simp add: R-def t-def red-step-limit)
  have s < k
    unfolding S-def s-def
    using bblue-dboost-step-limit big41 ‹l=k› by fastforce

  have k34: k powr (3/4) ≤ k powr 1
    using kn0 by (intro powr-mono) auto

  define g712 where g712 ≡ λk. 2 − ok-fun-71 μ k + g k
  have nV' ≥ 2
    using gorder-ge2 nV'-def by linarith
  have nV' ≤ 4 * card X0
    using 0 card-XY0 by (auto simp: nV'-def odd-iff-mod-2-eq-one)
  with μ01 have 2 powr (ok-fun-71 μ k − 2) * μ^k * (1−μ) ^ t * (bigbeta / μ)
^ s * nV'
      ≤ 2 powr ok-fun-71 μ k * μ^k * (1−μ) ^ t * (bigbeta / μ) ^ s * card X0
```

using *μ01* **by** (*simp add*: *powr-diff mult.assoc bigbeta-ge0 mult-left-mono*)
also have ... ≤ *card* (*Xseq halted-point*)
using *X-7-1 assms big71* **by** *blast*
also have ... ≤ *2 powr* (*g k*)
proof −
have *1*/*k* < *p0* − *3* ∗ *ε*
using *big* ‹*p0* ≥ *1*/*2*› **by** (*auto simp*: *Big-From-11-2-def*)
also have ... ≤ *pseq halted-point*
using *Y-6-2-halted big62 assms* **by** *blast*
finally have *pseq halted-point* > *1*/*k* **.**
moreover have *termination-condition* (*Xseq halted-point*) (*Yseq halted-point*)
using *halted-point-halted step-terminating-iff* **by** *blast*
ultimately have *card* (*Xseq halted-point*) ≤ *RN k* (*nat* ⌈*real k powr* (*3*/*4*)⌉)
using ‹*l*=*k*› *pseq-def termination-condition-def* **by** *auto*
then show *?thesis*
unfolding *g-def* **by** (*smt* (*verit*) *RN34-le-2powr-ok kn0 of-nat-le-iff*)
qed
finally have *58*: *2 powr* (*g k*) ≥ *2 powr* (*ok-fun-71 μ k* − *2*) ∗ *μ*^*k* ∗ (*1*−*μ*) ^ *t* ∗ (*bigbeta* / *μ*) ^ *s* ∗ *nV*′ **.**
then have *59*: *nV*′ ≤ *2 powr* (*g712 k*) ∗ (*1*/*μ*) ^ *k* ∗ (*1* / (*1*−*μ*)) ^ *t* ∗ (*μ* / *bigbeta*) ^ *s*
using *μ01 bb-gt0* **by** (*simp add*: *g712-def powr-diff powr-add mult.commute divide-simps*) *argo*

define *a* **where** *a* ≡ *2* / (*1*−*μ*)
have *ok-less1*: *a* ∗ *real k powr* (−*1*/*40*) < *1*
unfolding *a-def* **using** *big-le1* **by** *blast*
consider *s* < *k powr* (*39*/*40*) | *s* ≥ *k powr* (*39*/*40*) *bigbeta* ≥ (*1* − *a* ∗ *k powr* (−*1*/*40*)) ∗ (*s* / (*s* + *t*))
using *ZZ-8-6 big86 a-def* ‹*l*=*k*› **by** (*force simp*: *s-def t-def 𝒮-def ℛ-def*)
then show *?thesis*
proof *cases*
case *1*
define *h* **where** *h* ≡ *λc k*. *real k powr* (*39*/*40*) ∗ (*log 2 μ* + *real c* ∗ *log 2* (*real k*))
have *h*: *h c* ∈ *o*(*real*) **for** *c*
unfolding *h-def* **by** *real-asymp*

have *le-h*: |*s* ∗ *log 2* (*ratio μ s t*)| ≤ *h 1 k*
proof (*cases s*>*0*)
case *True*
with ‹*s*>*0*› **have** *μeq*: *ratio μ s t* = *μ* ∗ (*1* + *t*/*s*)
by (*auto simp*: *distrib-left add-divide-distrib*)
show *?thesis*
proof (*cases log 2* (*ratio μ s t*) ≤ *0*)
case *True*
have *s* ∗ (− *log 2* (*μ* ∗ (*1* + *t*/*s*))) ≤ *real k powr* (*39*/*40*) ∗ (*log 2 μ* + *log 2* (*real k*))
proof (*intro mult-mono*)

```
        show s ≤ k powr (39 / 40)
          using 1 by linarith
      next
        have inverse (μ * (1 + t/s)) ≤ inverse μ
          using μ01 inverse-le-1-iff by fastforce
        also have ... ≤ μ * k
              using bigμ μ01 by (metis neq-iff mult.assoc mult-le-cancel-left-pos
power2-eq-square right-inverse)
        finally have inverse (μ * (1 + t/s)) ≤ μ * k .
        moreover have 0 < μ * (1 + real t / real s)
          using μ01 ‹0 < s› by (simp add: zero-less-mult-iff add-num-frac)
        ultimately have − log 2 (μ * (1 + real t / real s)) ≤ log 2 (μ * k)
         using μ01 kn0 by (simp add: zero-less-mult-iff flip: log-inverse log-mult)
        then show − log 2 (μ * (1 + real t / real s)) ≤ log 2 μ + log 2 (real k)
          using ‹μ>0› kn0 log-mult by fastforce
      qed (use True μeq in auto)
      with ‹s>0› bigμ1 True show ?thesis
        by (simp add: μeq h-def mult-le-0-iff)
    next
      case False
      have lek: 1 + t/s ≤ k
      proof −
        have real t ≤ real t * real s
          using True mult-le-cancel-left1 by fastforce
        then have 1 + t/s ≤ 1 + t
          by (simp add: True pos-divide-le-eq)
        also have ... ≤ k
          using ‹t < k› by linarith
        finally show ?thesis .
      qed
      have |s * log 2 (ratio μ s t)| ≤ k powr (39/40) * log 2 (ratio μ s t)
        using False 1 by auto
      also have ... = k powr (39/40) * (log 2 (μ * (1 + t/s)))
        by (simp add: μeq)
      also have ... = k powr (39/40) * (log 2 μ + log 2 (1 + t/s))
      using μ01 by (smt (verit, best) divide-nonneg-nonneg log-mult of-nat-0-le-iff)

      also have ... ≤ k powr (39/40) * (log 2 μ + log 2 k)
       by (smt (verit, best) 1 Transcendental.log-mono divide-nonneg-nonneg lek
            mult-le-cancel-left-pos of-nat-0-le-iff)
      also have ... ≤ h 1 k
        unfolding h-def using kn0 by force
      finally show ?thesis .
    qed
  qed (use log2μk h-def in auto)

  have β: bigbeta ≥ 1 / (real k)^2
    using big53 bigbeta-ge-square ‹l=k› by blast
  then have (μ / bigbeta) ^ s ≤ (μ * (real k)^2) ^ s
```

using *bb-gt0 kn0 μ01* **by** (*intro power-mono*) (*auto simp*: *divide-simps mult.commute*)
also have ... $\leq (\mu * (real\ k)^2)$ *powr* (*k powr* (*39/40*))
using *μ01 bigμ2 1* **by** (*smt* (*verit*) *powr-less-mono powr-one-eq-one powr-realpow*)
also have ... = *2 powr* (*log 2* $((\mu * (real\ k)^2)$ *powr* (*k powr* (*39/40*))))
by (*smt* (*verit*, *best*) *bigμ2 powr-gt-zero powr-log-cancel*)
also have ... = *2 powr h 2 k*
using *μ01 bigμ2 kn0* **by** (*simp add*: *log-powr log-nat-power log-mult h-def*)
finally have †: (*μ* / *bigbeta*) ^ *s* ≤ *2 powr h 2 k* **.**
have ‡: *nV′* ≤ *2 powr* (*g712 k*) ∗ (*1/μ*) ^ *k* ∗ (*1* / (*1−μ*)) ^ *t* ∗ *2 powr h 2 k*
using *59 mult-left-mono* [*OF* †, *of 2 powr* (*g712 k*) ∗ (*1/μ*) ^ *k* ∗ (*1* / (*1−μ*)) ^ *t*]
by (*smt* (*verit*) *μ01 pos-prod-le powr-nonneg-iff zero-less-divide-iff zero-less-power*)
have ∗: *log 2 nV′* ≤ *k* ∗ *log 2* (*1/μ*) + *t* ∗ *log 2* (*1* / (*1−μ*)) + (*g712 k* + *h 2 k*)
using *μ01* ‹*nV′* ≥ *2*› **by** (*simp add*: *log-mult log-nat-power order.trans* [*OF Transcendental.log-mono* [*OF* - - ‡]])

show *?thesis*
proof −
have *le-ok-fun*: *g712 k* + *h 3 k* ≤ *ok-fun-11-2 μ k*
by (*simp add*: *g712-def h-def ok-fun-11-2-def g-def ok-fun-11-2a-def ok-fun-11-2b-def*)
have *h3*: *h 3 k* = *h 1 k* + *h 2 k* − *real k powr* (*39/40*) ∗ *log 2 μ*
by (*simp add*: *h-def algebra-simps*)
have *0* ≤ *h 1 k* + *s* ∗ *log 2* ((*μ* ∗ *real s* + *μ* ∗ *real t*) / *s*)
by (*smt* (*verit*, *del-insts*) *of-nat-add distrib-left le-h*)
moreover have *log 2 μ* < *0*
using *μ01* **by** *simp*
ultimately have *g712 k* + *h 2 k* ≤ *s* ∗ *log 2* (*ratio μ s t*) + *ok-fun-11-2 μ k*
by (*smt* (*verit*, *best*) *kn0 distrib-left h3 le-ok-fun nat-neq-iff of-nat-eq-0-iff pos-prod-lt powr-gt-zero*)
then show *log 2 nV′* ≤ *k* ∗ *log 2* (*1/μ*) + *t* ∗ *log 2* (*1* / (*1−μ*)) + *s* ∗ *log 2* (*ratio μ s t*) + *ok-fun-11-2 μ k*
using ∗ **by** *linarith*
qed
next
case *2*
then have *s* > *0*
using *kn0 powr-gt-zero* **by** *fastforce*
define *h* **where** *h* ≡ λ*k*. *real k* ∗ *log 2* (*1* − *a* ∗ *k powr* (−*1/40*))
have *s* ∗ *log 2* (*μ* / *bigbeta*) = *s* ∗ *log 2 μ* − *s* ∗ *log 2* (*bigbeta*)
using *μ01 bb-gt0 2* **by** (*simp add*: *log-divide algebra-simps*)
also have ... ≤ *s* ∗ *log 2 μ* − *s* ∗ *log 2* ((*1* − *a* ∗ *k powr* (−*1/40*)) ∗ (*s* / (*s* + *t*)))
using *2* ‹*s*>*0*› *ok-less1* **by** (*intro diff-mono order-refl mult-left-mono Transcendental.log-mono*) *auto*
also have ... = *s* ∗ *log 2 μ* − *s* ∗ (*log 2* (*1* − *a* ∗ *k powr* (−*1/40*)) + *log 2* (*s* / (*s* + *t*)))
using ‹*0* < *s*› *a-def add-log-eq-powr big-le1* **by** *auto*

```
    also have ... = s * log 2 (ratio μ s t) − s * log 2 (1 − a * k powr (−1/40))
     using ‹0 < μ› ‹0 < s› minus-log-eq-powr by (auto simp flip: right-diff-distrib')
    also have ... < s * log 2 (ratio μ s t) − h k
    proof −
      have log 2 (1 − a * real k powr (−1/40)) < 0
        using μ01 kn0 a-def ok-less1 by auto
      with ‹s<k› show ?thesis
        by (simp add: h-def)
    qed
    finally have †: s * log 2 (μ / bigbeta) < s * log 2 (ratio μ s t) − h k .
    show ?thesis
    proof −
      have le-ok-fun: g712 k − h k ≤ ok-fun-11-2 μ k
         by (simp add: g712-def h-def ok-fun-11-2-def g-def ok-fun-11-2a-def a-def
ok-fun-11-2c-def)
      have log 2 nV' ≤ s * log 2 (μ / bigbeta) + k * log 2 (1/μ) + t * log 2 (1 /
(1−μ)) + (g712 k)
      proof (intro order.trans [OF Transcendental.log-mono [OF - - 59]])
        show log 2 (2 powr g712 k * (1/μ) ^ k * (1 / (1−μ)) ^ t * (μ / bigbeta)
^ s)
            ≤ s * log 2 (μ / bigbeta) + k * log 2 (1/μ) + t * log 2 (1 / (1−μ)) +
g712 k
          using bb-gt0 μ01 by (simp add: log-mult log-nat-power)
      qed (use ‹nV' ≥ 2› in auto)
      with † le-ok-fun show log 2 nV' ≤ k * log 2 (1/μ) + t * log 2 (1 / (1−μ))
+ s * log 2 (ratio μ s t) + ok-fun-11-2 μ k
        by simp
    qed
  qed
qed
```

10.2 Lemma 11.3

same remark as in Lemma 11.2 about the use of the Ramsey number in the conclusion

```
lemma (in Book) From-11-3:
  assumes l=k
  assumes big: Big-Y-6-1 μ k
  defines ℛ ≡ Step-class {red-step} and 𝒮 ≡ Step-class {dboost-step}
  defines t ≡ card ℛ and s ≡ card 𝒮
  defines nV' ≡ Suc nV
  assumes 0: card Y0 ≥ nV div 2 and p0 ≥ 1/2
  shows log 2 nV' ≤ log 2 (RN k (k−t)) + s + t + 2 − ok-fun-61 k
proof −
  define RS where RS ≡ Step-class {red-step,dboost-step}
  have RS = ℛ ∪ 𝒮
    using Step-class-insert ℛ-def 𝒮-def RS-def by blast
  moreover obtain finite ℛ finite 𝒮
    by (simp add: ℛ-def 𝒮-def)
```

moreover have *disjnt* $\mathcal{R}$ $\mathcal{S}$
using $\mathcal{R}$*-def* $\mathcal{S}$*-def disjnt-Step-class* **by** *auto*
ultimately have *card-RS*: *card RS* = *t*+*s*
by (*simp add*: *t-def s-def card-Un-disjnt*)
have *4*: *nV'/4* ≤ *card Y0*
using *0 card-XY0* **by** (*auto simp*: *nV'-def odd-iff-mod-2-eq-one*)
have *ge0*: *0* ≤ *2 powr ok-fun-61 k* ∗ *p0* ^ *card RS*
using *p0-01* **by** *fastforce*
have *nV'* ≥ *2*
using *gorder-ge2 nV'-def* **by** *linarith*
have *2 powr* (− *real s* − *real t* + *ok-fun-61 k* − *2*) ∗ *nV'* = *2 powr* (*ok-fun-61 k* − *2*) ∗ (*1/2*) ^ *card RS* ∗ *nV'*
by (*simp add*: *powr-add powr-diff powr-minus power-add powr-realpow divide-simps card-RS*)
also have ... ≤ *2 powr* (*ok-fun-61 k* − *2*) ∗ *p0* ^ *card RS* ∗ *nV'*
using *power-mono* [*OF* ‹*p0* ≥ *1/2*›] ‹*nV'* ≥ *2*› **by** *auto*
also have ... ≤ *2 powr* (*ok-fun-61 k*) ∗ *p0* ^ *card RS* ∗ (*nV'/4*)
by (*simp add*: *divide-simps powr-diff split*: *if-split-asm*)
also have ... ≤ *2 powr* (*ok-fun-61 k*) ∗ *p0* ^ *card RS* ∗ *card Y0*
using *mult-left-mono* [*OF 4 ge0*] **by** *simp*
also have ... ≤ *card* (*Yseq halted-point*)
using *Y-6-1 big* ‹*l*=*k*› **by** (*auto simp*: *RS-def divide-simps split*: *if-split-asm*)
finally have *2 powr* (− *real s* − *real t* + *ok-fun-61 k* − *2*) ∗ *nV'* ≤ *card* (*Yseq halted-point*) **.**
moreover
{ assume *card* (*Yseq halted-point*) ≥ *RN k* (*k*−*t*)
then obtain *K* **where** *K*: *K* ⊆ *Yseq halted-point* **and** *size-clique* (*k*−*t*) *K Red* ∨ *size-clique k K Blue*
by (*metis RN-commute Red-Blue-RN Yseq-subset-V*)
then have *KRed*: *size-clique* (*k*−*t*) *K Red*
using ‹*l*=*k*› *no-Blue-clique* **by** *blast*
have *card* (*K* ∪ *Aseq halted-point*) = *k*
proof (*subst card-Un-disjnt*)
show *finite K finite* (*Aseq halted-point*)
using *K finite-Aseq finite-Yseq infinite-super* **by** *blast*+
show *disjnt K* (*Aseq halted-point*)
using *valid-state-seq*[*of halted-point*] *K disjnt-subset1*
by (*auto simp*: *valid-state-def disjoint-state-def*)
have *card* (*Aseq halted-point*) = *t*
using *red-step-eq-Aseq* $\mathcal{R}$*-def t-def* **by** *presburger*
then show *card K* + *card* (*Aseq halted-point*) = *k*
using *Aseq-less-k*[*OF*] *nat-less-le KRed size-clique-def* **by** *force*
qed
moreover have *clique* (*K* ∪ *Aseq halted-point*) *Red*
proof −
obtain *K* ⊆ *V Aseq halted-point* ⊆ *V*
by (*meson Aseq-subset-V KRed size-clique-def*)
moreover have *clique K Red*
using *KRed size-clique-def* **by** *blast*

moreover have *clique* (*Aseq halted-point*) *Red*
by (*meson A-Red-clique valid-state-seq*)
moreover have *all-edges-betw-un* (*Aseq halted-point*) (*Yseq halted-point*) ⊆ *Red*
using *valid-state-seq*[*of halted-point*] *K*
by (*auto simp*: *valid-state-def RB-state-def all-edges-betw-un-Un2*)
then have *all-edges-betw-un K* (*Aseq halted-point*) ⊆ *Red*
using *K all-edges-betw-un-mono2 all-edges-betw-un-commute* **by** *blast*
ultimately show *?thesis*
by (*simp add*: *local.clique-Un*)
qed
ultimately have *size-clique k* (*K* ∪ *Aseq halted-point*) *Red*
using *KRed Aseq-subset-V* **by** (*auto simp*: *size-clique-def*)
then have *False*
using *no-Red-clique* **by** *blast*
}
ultimately have ∗: *2 powr* (− *real s* − *real t* + *ok-fun-61 k* − *2*) ∗ *nV'* < *RN k* (*k*−*t*)
by *fastforce*
have − *real s* − *real t* + *ok-fun-61 k* − *2* + *log 2 nV'* = *log 2* (*2 powr* (− *real s* − *real t* + *ok-fun-61 k* − *2*) ∗ *nV'*)
using *add-log-eq-powr* ‹*nV'* ≥ *2*› **by** *auto*
also have . . . ≤ *log 2* (*RN k* (*k*−*t*))
using ∗ *Transcendental.log-mono* ‹*nV'* ≥ *2*› *less-eq-real-def* **by** *auto*
finally show *log 2 nV'* ≤ *log 2* (*RN k* (*k* − *t*)) + *real s* + *real t* + *2* − *ok-fun-61 k*
by *linarith*
qed

10.3 Theorem 11.1

definition *FF* :: *nat* ⇒ *real* ⇒ *real* ⇒ *real* **where**
FF ≡ λ*k x y*. *log 2* (*RN k* (*nat*⌊*real k* − *x* ∗ *real k*⌋)) / *real k* + *x* + *y*

definition *GG* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
GG ≡ λμ *x y*. *log 2* (*1*/μ) + *x* ∗ *log 2* (*1*/(*1*−μ)) + *y* ∗ *log 2* (μ ∗ (*x*+*y*) / *y*)

definition *FF-bound* :: *nat* ⇒ *real* ⇒ *real* **where**
FF-bound ≡ λ*k u*. *FF k 0 u* + *1*

lemma *log2-RN-ge0*: *0* ≤ *log 2* (*RN k k*) / *k*
proof (*cases k*=*0*)
case *False*
then have *RN k k* ≥ *1*
by (*simp add*: *RN-eq-0-iff leI*)
then show *?thesis*
by *simp*
qed *auto*

```
lemma le-FF-bound:
  assumes x: x ∈ {0..1} and y ∈ {0..u}
  shows FF k x y ≤ FF-bound k u
proof (cases ⌊k − x∗k⌋ = 0)
  case True — to handle the singularity
  with assms log2-RN-ge0[of k] show ?thesis
    by (simp add: True FF-def FF-bound-def log-def)
next
  case False
  with gr0I have k>0 by fastforce
  with False assms have ∗: 0 < ⌊k − x∗k⌋
    using linorder-neqE-linordered-idom by fastforce
  have le-k: k − x∗k ≤ k
    using x by auto
  then have le-k: nat ⌊k − x∗k⌋ ≤ k
    by linarith
  have log 2 (RN k (nat ⌊k − x∗k⌋)) / k ≤ log 2 (RN k k) / k
  proof (intro divide-right-mono Transcendental.log-mono)
    show 0 < real (RN k (nat ⌊k − x∗k⌋))
      by (metis RN-eq-0-iff ‹k>0› gr-zeroI ∗ of-nat-0-less-iff zero-less-nat-eq)
  qed (auto simp: RN-mono le-k)
  then show ?thesis
    using assms False le-SucE by (fastforce simp: FF-def FF-bound-def)
qed

lemma FF2: y' ≤ y ⟹ FF k x y' ≤ FF k x y
  by (simp add: FF-def)

lemma FF-GG-bound:
  assumes μ: 0 < μ μ < 1 and x: x ∈ {0..1} and y: y ∈ {0..μ ∗ x / (1−μ) +
η}
  shows min (FF k x y) (GG μ x y) + η ≤ FF-bound k (μ / (1−μ) + η) + η
proof −
  have FF-ub: FF k x y ≤ FF-bound k (μ / (1−μ) + η)
  proof (rule order.trans)
    show FF k x y ≤ FF-bound k y
      using x y by (simp add: le-FF-bound)
  next
    have y ≤ μ / (1−μ) + η
      using x y μ by simp (smt (verit, best) frac-le mult-left-le)
    then show FF-bound k y ≤ FF-bound k (μ / (1−μ) + η)
      by (simp add: FF-bound-def FF-def)
  qed
  show ?thesis
    using FF-ub by auto
qed

context P0-min
```

```
begin

definition ok-fun-11-1 ≡ λμ k. max (ok-fun-11-2 μ k) (2 − ok-fun-61 k)

lemma ok-fun-11-1:
  assumes 0<μ μ<1
  shows ok-fun-11-1 μ ∈ o(real)
  unfolding ok-fun-11-1-def
  by (simp add: assms const-smallo-real maxmin-in-smallo ok-fun-11-2 ok-fun-61
sum-in-smallo)

lemma eventually-ok111-le-η:
  assumes η > 0 and μ: 0<μ μ<1
  shows ∀∞k. ok-fun-11-1 μ k / k ≤ η
proof −
  have (λk. ok-fun-11-1 μ k / k) ∈ o(λk. 1)
    using eventually-mono ok-fun-11-1 [OF μ] by (fastforce simp: smallo-def
divide-simps)
  with assms have ∀∞k. |ok-fun-11-1 μ k| / k ≤ η
    by (auto simp: smallo-def)
  then show ?thesis
    by (metis (mono-tags, lifting) eventually-mono abs-divide abs-le-D1 abs-of-nat)
qed

lemma eventually-powr-le-η:
  assumes η > 0
  shows ∀∞k. (2 / (1−μ)) ∗ k powr (−1/20) ≤ η
  using assms by real-asymp

definition Big-From-11-1 ≡
  λη μ k. Big-From-11-2 μ k ∧ Big-ZZ-8-5 μ k ∧ Big-Y-6-1 μ k ∧ ok-fun-11-1 μ
k / k ≤ η/2
      ∧ (2 / (1−μ)) ∗ k powr (−1/20) ≤ η/2
      ∧ Big-Closer-10-1 (1/101) (nat⌈k/100⌉) ∧ 3 / (k ∗ ln 2) ≤ η/2 ∧ k≥3
```

In sections 9 and 10 (and by implication all proceeding sections), we needed to consider a closed interval of possible values of μ. Let's hope, maybe not here. The fact below can only be proved with the strict inequality $0 < \eta$, which is why it is also strict in the theorems depending on this property.

```
lemma Big-From-11-1:
  assumes η > 0 0<μ μ<1
  shows ∀∞k. Big-From-11-1 η μ k
proof −
  have ∀∞l. Big-Closer-10-1 (1/101) l
    by (rule Big-Closer-10-1) auto
  then have a: ∀∞k. Big-Closer-10-1 (1/101) (nat⌈k/100⌉)
    unfolding eventually-sequentially
    by (meson le-divide-eq-numeral1(1) le-natceiling-iff nat-ceiling-le-eq)
  have b: ∀∞k. 3 / (k ∗ ln 2) ≤ η/2
```

using ‹η>0› **by** *real-asymp*
show *?thesis*
unfolding *Big-From-11-1-def*
using *assms a b Big-From-11-2*[*of* μ μ] *Big-ZZ-8-5*[*of* μ μ] *Big-Y-6-1*[*of* μ μ]
using *eventually-ok111-le-*η[*of* $\eta/2$] *eventually-powr-le-*η [*of* $\eta/2$]
by (*auto simp*: *eventually-conj-iff all-imp-conj-distrib eventually-sequentially*)
qed

The actual proof of theorem 11.1 is now combined with the development of section 12, since the concepts seem to be inescapably mixed up.

end

end

11 The Proof of Theorem 1.1

theory *The-Proof*
imports *From-Diagonal*

begin

11.1 The bounding functions

definition *H* ≡ $\lambda p.\ -p * log\ 2\ p - (1-p) * log\ 2\ (1-p)$

definition *dH* **where** *dH* ≡ $\lambda x{::}real.\ -ln(x)/ln(2) + ln(1 - x)/ln(2)$

lemma *dH* [*derivative-intros*]:
assumes $0<x$ $x<1$
shows (*H has-real-derivative dH x*) (*at x*)
unfolding *H-def dH-def log-def*
by (*rule derivative-eq-intros* | *use assms* **in** *force*)+

lemma *H0* [*simp*]: *H 0 = 0* **and** *H1* [*simp*]: *H 1 = 0*
by (*auto simp*: *H-def*)

lemma *H-reflect*: *H* (*1*−*p*) = *H p*
by (*simp add*: *H-def*)

lemma *H-ge0*:
assumes $0 \le p$ $p \le 1$
shows $0 \le H\ p$
unfolding *H-def*
by (*smt* (*verit*, *best*) *assms mult-minus-left mult-le-0-iff zero-less-log-cancel-iff*)

Going up, from 0 to 1/2

lemma *H-half-mono*:
assumes $0 \le p'$ $p' \le p$ $p \le 1/2$
shows $H\ p' \le H\ p$

```
proof (cases p′=0)
  case True
  then have H p′ = 0
    by (auto simp: H-def)
  then show ?thesis
    by (smt (verit) H-ge0 True assms(2) assms(3) divide-le-eq-1-pos)
next
  case False
  with assms have p′>0 by simp
  have dH(1/2) = 0
    by (simp add: dH-def)
  moreover
  have dH x ≥ 0 if 0<x x≤1/2 for x
    using that by (simp add: dH-def divide-right-mono)
  ultimately show ?thesis
   by (smt (verit) dH DERIV-nonneg-imp-nondecreasing ‹p′>0› assms le-divide-eq-1-pos)
qed
```

Going down, from 1/2 to 1

```
lemma H-half-mono′:
  assumes 1/2 ≤ p′ p′≤p p ≤ 1
  shows H p′ ≥ H p
  using H-half-mono [of 1−p 1−p′] H-reflect assms by auto

lemma H-half: H(1/2) = 1
  by (simp add: H-def log-divide)

lemma H-le1:
  assumes 0 ≤ p p ≤ 1
  shows H p ≤ 1
  by (smt (verit, best) H0 H1 H-ge0 H-half-mono H-half-mono′ H-half assms)
```

Many thanks to Fedor Petrov on mathoverflow

```
lemma H-12-1:
  fixes a b::nat
  assumes a ≥ b
  shows log 2 (a choose b) ≤ a ∗ H(b/a)
proof (cases a=b ∨ b=0)
  case True
  with assms show ?thesis
    by (auto simp: H-def)
next
  let ?p = b/a
  case False
  then have p01: 0 < ?p ?p < 1
    using assms by auto
  then have (a choose b) ∗ ?p ^ b ∗ (1−?p) ^ (a−b) ≤ (?p + (1−?p)) ^ a
    by (subst binomial-ring) (force intro!: member-le-sum assms)
  also have . . . = 1
```

by *simp*
finally have §: (*a choose b*) ∗ *?p* ^ *b* ∗ (*1*−*?p*) ^ (*a*−*b*) ≤ *1* **.**
have *log 2* (*a choose b*) + *b* ∗ *log 2 ?p* + (*a*−*b*) ∗ *log 2* (*1*−*?p*) ≤ *0*
using *Transcendental.log-mono* [*OF* - - §] *False assms*
by (*force simp add*: *p01 log-mult log-nat-power*)
then show *?thesis*
using *p01 False assms* **unfolding** *H-def* **by** (*simp add*: *divide-simps*)
qed

definition *gg* ≡ *GG* (*2*/*5*)

lemma *gg-eq*: *gg x y* = *log 2* (*5*/*2*) + *x* ∗ *log 2* (*5*/*3*) + *y* ∗ *log 2* ((*2* ∗ (*x*+*y*)) / (*5*∗*y*))
by (*simp add*: *gg-def GG-def*)

definition *f1* ≡ λ*x y*. *x* + *y* + (*2*−*x*) ∗ *H*(*1*/(*2*−*x*))

definition *f2* ≡ λ*x y*. *f1 x y* − (*1* / (*40* ∗ *ln 2*)) ∗ ((*1*−*x*) / (*2*−*x*))

definition *ff* ≡ λ*x y*. *if x* < *3*/*4 then f1 x y else f2 x y*

Incorporating Bhavik's idea, which gives us a lower bound for γ of $1/101$

definition *ffGG* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
ffGG ≡ λμ *x y*. *max 1.9* (*min* (*ff x y*) (*GG* μ *x y*))

The proofs involving *Sup* are needlessly difficult because ultimately the sets involved are finite, eliminating the need to demonstrate boundedness. Simpler might be to use the extended reals.

lemma *f1-le*:
assumes *x*≤*1*
shows *f1 x y* ≤ *y*+*2*
unfolding *f1-def*
using *H-le1* [*of 1*/(*2*−*x*)] *assms*
by (*smt* (*verit*) *divide-le-eq-1-pos divide-nonneg-nonneg mult-left-le*)

lemma *ff-le4*:
assumes *x*≤*1 y*≤*1*
shows *ff x y* ≤ *4*
proof −
have *ff x y* ≤ *f1 x y*
using *assms* **by** (*simp add*: *ff-def f2-def*)
also have . . . ≤ *4*
using *assms* **by** (*smt* (*verit*) *f1-le*)
finally show *?thesis* **.**
qed

lemma *ff-GG-bound*:
assumes *x*≤*1 y*≤*1*
shows *ffGG* μ *x y* ≤ *4*

using *ff-le4* [*OF assms*] **by** (*auto simp*: *ffGG-def*)

lemma *bdd-above-ff-GG*:
assumes $x \leq 1$ $u \leq 1$
shows *bdd-above* ((λy. *ffGG* μ *x y* + η) ‘ {*0..u*})
using *ff-GG-bound assms*
by (*intro bdd-above.I2* [**where** $M = 4+\eta$]) *force*

lemma *bdd-above-SUP-ff-GG*:
assumes $0 \leq u$ $u \leq 1$
shows *bdd-above* ((λx. $\bigsqcup y \in \{0..u\}$. *ffGG* μ *x y* + η) ‘ {*0..1*})
using *bdd-above-ff-GG assms*
by (*intro bdd-aboveI* [**where** $M = 4 + \eta$]) (*auto simp*: *cSup-le-iff ff-GG-bound Pi-iff*)

Claim (62). A singularity if $x = 1$. Okay if we put $\ln(0) = 0$

lemma *FF-le-f1*:
fixes *k*::*nat* **and** *x y*::*real*
assumes *x*: $0 \leq x$ $x \leq 1$ **and** *y*: $0 \leq y$ $y \leq 1$
shows *FF k x y* $\leq$ *f1 x y*
proof (*cases nat*$\lfloor k - x*k \rfloor = 0$)
case *True*
with *x* **show** *?thesis*
by (*simp add*: *FF-def f1-def H-ge0 log-def*)
next
case *False*
let *?kl* = $k + k - nat \lceil x*k \rceil$
have *kk-less-1*: *k* / *?kl* < *1*
using *x False* **by** (*simp add*: *field-split-simps*, *linarith*)
have *le*: *nat*$\lfloor k - x*k \rfloor \leq k - nat\lceil x*k \rceil$
using *floor-ceiling-diff-le x*
by (*meson mult-left-le-one-le mult-nonneg-nonneg of-nat-0-le-iff*)
have $k>0$
using *False zero-less-iff-neq-zero* **by** *fastforce*
have *RN-gt0*: *RN k* (*nat*$\lfloor k - x*k \rfloor$) > *0*
by (*metis False RN-eq-0-iff* ‹$k>0$› *gr0I*)
then have §: *RN k* (*nat*$\lfloor k - x*k \rfloor$) $\leq$ *k* + *nat*$\lfloor k - x*k \rfloor$ *choose k*
using *RN-le-choose* **by** *force*
also have ... $\leq$ $k + k - nat\lceil x*k \rceil$ *choose k*
using *False Nat.le-diff-conv2 binomial-right-mono le* **by** *fastforce*
finally have *RN k* (*nat* $\lfloor real\ k - x*k \rfloor$) $\leq$ *?kl choose k* **.**
with *RN-gt0* **have** *FF k x y* $\leq$ *log 2* (*?kl choose k*) / *k* + *x* + *y*
by (*simp add*: *FF-def divide-right-mono nat-less-real-le*)
also have ... $\leq$ (*?kl* ∗ *H*(*k*/*?kl*)) / *k* + *x* + *y*
proof −
have $k \leq k + k - nat\lceil x*k \rceil$
using *False* **by** *linarith*
then show *?thesis*
by (*simp add*: *H-12-1 divide-right-mono*)

```
  qed
  also have ... ≤ f1 x y
  proof -
    have 1: ?kl / k ≤ 2−x
        using x by (simp add: field-split-simps)
    have 2: H (k / ?kl) ≤ H (1 / (2−x))
    proof (intro H-half-mono')
      show 1 / (2−x) ≤ k / ?kl
        using x False by (simp add: field-split-simps, linarith)
    qed (use x kk-less-1 in auto)
    have ?kl / k ∗ H (k / ?kl) ≤ (2−x) ∗ H (1 / (2−x))
      using x mult-mono [OF 1 2 - H-ge0] kk-less-1 by fastforce
    then show ?thesis
      by (simp add: f1-def)
  qed
  finally show ?thesis .
qed
```

Bhavik's *eleven-one-large-end*

```
lemma f1-le-19:
  fixes k::nat and x y::real
  assumes x: 0.99 ≤ x x ≤ 1 and y: 0 ≤ y y ≤ 3/4
  shows f1 x y ≤ 1.9
proof -
  have A: 2−x ≤ 1.01
    using x by simp
  have H (1 / (2−x)) ≤ H (1 / (2−0.99))
    using x by (intro H-half-mono') (auto simp: divide-simps)
  also have ... ≤ 0.081
    unfolding H-def by (approximation 15)
  finally have B: H (1 / (2−x)) ≤ 0.081 .
  have (2−x) ∗ H (1 / (2−x)) ≤ 1.01 ∗ 0.081
    using mult-mono [OF A B] x
    by (smt (verit) A H-ge0 divide-le-eq-1-pos divide-nonneg-nonneg)
  with assms show ?thesis by (auto simp: f1-def)
qed
```

Claim (63) in weakened form; we get rid of the extra bit later

```
lemma (in P0-min) FF-le-f2:
  fixes k::nat and x y::real
  assumes x: 3/4 ≤ x x ≤ 1 and y: 0 ≤ y y ≤ 1
  and l: real l = k − x∗k
  assumes p0-min-101: p0-min ≤ 1 − 1/5
  defines γ ≡ real l / (real k + real l)
  defines γ0 ≡ min γ (0.07)
  assumes γ > 0
  shows FF k x y ≤ f2 x y + ok-fun-10-1 γ k / (k ∗ ln 2)
proof -
  have l>0
```

```
    using ‹γ>0› γ-def less-irrefl by fastforce
  have x>0
    using x by linarith
  with l have k≥l
    by (smt (verit, del-insts) of-nat-0-le-iff of-nat-le-iff pos-prod-lt)
  with ‹0 < l› have k>0 by force
  have RN-gt0: RN k l > 0
    by (metis RN-eq-0-iff ‹0 < k› ‹0 < l› gr0I)
  define δ where δ ≡ γ/40
  have A: l / real(k+l) = (1−x)/(2−x)
    using x ‹k>0› by (simp add: l field-simps)
  have B: real(k+l) / k = 2−x
    using ‹0 < k› l by (auto simp: divide-simps left-diff-distrib)
  have γ: γ ≤ 1/5
    using x A by (simp add: γ-def)
  have 1 − 1 / (2−x) = (1−x) / (2−x)
    using x by (simp add: divide-simps)
  then have Heq: H (1 / (2−x)) = H ((1−x) / (2−x))
    by (metis H-reflect)
  have RN k l ≤ exp (−δ∗k + ok-fun-10-1 γ k) ∗ (k+l choose l)
    unfolding δ-def γ-def
  proof (rule Closer-10-1-unconditional)
    show 0 < l / (real k + real l) l / (real k + real l) ≤ 1/5
      using γ ‹γ > 0› by (auto simp: γ-def)
    have min (l / (k + real l)) 0.07 > 0
      using ‹l>0› by force
  qed (use p0-min-101 in auto)
   with RN-gt0 have FF k x y ≤ log 2 (exp (−δ∗k + ok-fun-10-1 γ k) ∗ (k+l
choose l)) / k + x + y
    unfolding FF-def
    by (intro add-mono divide-right-mono Transcendental.log-mono; simp flip: l)
  also have ... = (log 2 (exp (−δ∗k + ok-fun-10-1 γ k)) + log 2 (k+l choose l))
/ k + x + y
    by (simp add: log-mult)
  also have ... ≤ ((−δ∗k + ok-fun-10-1 γ k) / ln 2 + (k+l) ∗ H(l/(k+l))) / k
+ x + y
    using H-12-1
    by (smt (verit, ccfv-SIG) log-exp divide-right-mono le-add2 of-nat-0-le-iff)
  also have ... = (−δ∗k + ok-fun-10-1 γ k) / k / ln 2 + (k+l) / k ∗ H(l/(k+l))
+ x + y
    by argo
  also have ... = −δ / ln 2 + ok-fun-10-1 γ k / (k ∗ ln 2) + (2−x) ∗ H((1−x)/(2−x))
+ x + y
  proof −
    have (−δ∗k + ok-fun-10-1 γ k) / k / ln 2 = −δ / ln 2 + ok-fun-10-1 γ k /
(k ∗ ln 2)
      using ‹0 < k› by (simp add: divide-simps)
    with A B show ?thesis
      by presburger
```

```
  qed
  also have ... = - (log 2 (exp 1) / 40) * (1-x) / (2-x) + ok-fun-10-1 γ k /
(k * ln 2) + (2-x) * H((1-x)/(2-x)) + x + y
    using A by (force simp: δ-def γ-def field-simps)
  also have ... ≤ f2 x y + ok-fun-10-1 γ k / (real k * ln 2)
    by (simp add: Heq f1-def f2-def mult-ac)
  finally show ?thesis .
qed
```

The body of the proof has been extracted to allow the symmetry argument. And 1/12 is 3/4-2/3, the latter number corresponding to $\mu = (2::'a) / (5::'a)$

```
lemma (in Book-Basis) From-11-1-Body:
  fixes V :: 'a set
  assumes μ: 0 < μ μ ≤ 2/5 and η: 0 < η η ≤ 1/12
    and ge-RN: Suc nV ≥ RN k k
    and Red: graph-density Red ≥ 1/2
    and p0-min12: p0-min ≤ 1/2
    and Red-E: Red ⊆ E and Blue-def: Blue = E\Red
    and no-Red-K: ¬ (∃K. size-clique k K Red)
    and no-Blue-K: ¬ (∃K. size-clique k K Blue)
    and big: Big-From-11-1 η μ k
  shows log 2 (RN k k) / k ≤ (SUP x ∈ {0..1}. SUP y ∈ {0..3/4}. ffGG μ x y
+ η)
proof -
  have 12: 3/4 - 2/3 = (1/12::real)
    by simp
  define η' where η' ≡ η/2
  have η': 0 < η' η' ≤ 1/12
    using η by (auto simp: η'-def)
 have k>0 and big101: Big-Closer-10-1 (1/101) (nat⌈k/100⌉) and ok-fun-10-1-le:
3 / (k * ln 2) ≤ η'
    using big by (auto simp: Big-From-11-1-def η'-def)
  interpret No-Cliques where l=k
    using assms unfolding No-Cliques-def No-Cliques-axioms-def
    using Book-Basis-axioms P0-min-axioms by blast
   obtain X0 Y0 where card-X0: card X0 ≥ nV/2 and card-Y0: card Y0 =
gorder div 2
    and X0 = V \ Y0 Y0⊆V
    and p0-half: 1/2 ≤ gen-density Red X0 Y0
    and Book V E p0-min Red Blue k k μ X0 Y0
  proof (rule to-Book)
    show p0-min ≤ graph-density Red
      using p0-min12 Red by linarith
    show 0 < μ μ < 1
      using μ by auto
  qed (use infinite-UNIV p0-min Blue-def Red μ in auto)
  then interpret Book V E p0-min Red Blue k k μ X0 Y0
    by meson
```

```
define ℛ where ℛ ≡ Step-class {red-step}
define 𝒮 where 𝒮 ≡ Step-class {dboost-step}
define t where t ≡ card ℛ
define s where s ≡ card 𝒮
define x where x ≡ t/k
define y where y ≡ s/k
have sts: (s + real t) / s = (x+y) / y
  using ‹k>0› by (simp add: x-def y-def divide-simps)
have t<k
  by (simp add: ℛ-def μ t-def red-step-limit)
then obtain x01: 0≤x x<1
  by (auto simp: x-def)

have big41: Big-Blue-4-1 μ k and big61: Big-Y-6-1 μ k
  and big85: Big-ZZ-8-5 μ k and big11-2: Big-From-11-2 μ k
  and ok111-le: ok-fun-11-1 μ k / k ≤ η'
  and powr-le: (2 / (1−μ)) ∗ k powr (−1/20) ≤ η' and k>0
 using big by (auto simp: Big-From-11-1-def Big-Y-6-1-def Big-Y-6-2-def η'-def)
then have big53: Big-Red-5-3 μ k
  by (meson Big-From-11-2-def)
have μ < 1
  using μ by auto

have s<k
  unfolding s-def 𝒮-def
  by (meson μ le-less-trans bblue-dboost-step-limit big41 le-add2)
then obtain y01: 0≤y y<1
  by (auto simp: y-def)
```

Now that x and y are fixed, here's the body of the outer supremum

```
define w where w ≡ (⨆y∈{0..3/4}. ffGG μ x y + η)
show ?thesis
proof (intro cSup-upper2 imageI)
  show w ∈ (λx. ⨆y∈{0..3/4}. ffGG μ x y + η) ‘ {0..1}
    using x01 by (force simp: w-def intro!: image-eqI [where x=x])
next
  have μ23: μ / (1−μ) ≤ 2/3
    using μ by (simp add: divide-simps)
  have beta-le: bigbeta ≤ μ
    using ‹μ<1› μ big53 bigbeta-le by blast
  have s ≤ (bigbeta / (1 − bigbeta)) ∗ t + (2 / (1−μ)) ∗ k powr (19/20)
    using ZZ-8-5 [OF big85] μ by (auto simp: ℛ-def 𝒮-def s-def t-def)
  also have ... ≤ (μ / (1−μ)) ∗ t + (2 / (1−μ)) ∗ k powr (19/20)
   by (smt (verit, ccfv-SIG) ‹μ<1› μ beta-le frac-le mult-right-mono of-nat-0-le-iff)
  also have ... ≤ (μ / (1−μ)) ∗ t + (2 / (1−μ)) ∗ (k powr (−1/20) ∗ k powr
1)
    unfolding powr-add [symmetric] by simp
  also have ... ≤ (2/3) ∗ t + (2 / (1−μ)) ∗ (k powr (−1/20)) ∗ k
    using mult-right-mono [OF μ23, of t] by (simp add: mult-ac)
```

also have $\ldots \le (3/4 - \eta') * k + (2 / (1-\mu)) * (k\ powr\ (-1/20)) * k$
proof –
have $(2/3) * t \le (2/3) * k$
using ‹*t < k*› **by** *simp*
then show *?thesis*
using *12* η' **by** (*smt* (*verit*) *mult-right-mono of-nat-0-le-iff*)
qed
finally have $s \le (3/4 - \eta') * k + (2 / (1-\mu)) * k\ powr\ (-1/20) * k$
by *simp*
with *mult-right-mono* [*OF powr-le*, *of k*]
have †: $s \le 3/4 * k$
by (*simp add*: *mult.commute right-diff-distrib′*)
then have $y \le 3/4$
by (*metis* † ‹*0 < k*› *of-nat-0-less-iff pos-divide-le-eq y-def*)

have *k-minus-t*: *nat* $\lfloor$*real k* − *real t*$\rfloor$ = *k*−*t*
by *linarith*
have *nV div 2* ≤ *card Y0*
by (*simp add*: *card-Y0*)
then have §: *log 2* (*Suc nV*) ≤ *log 2* (*RN k* (*k*−*t*)) + *s* + *t* + *2* − *ok-fun-61 k*
using *From-11-3* [*OF - big61*] *p0-half* μ **by** (*auto simp*: $\mathcal{R}$*-def* $\mathcal{S}$*-def p0-def s-def t-def*)

define *l* **where** *l* ≡ *k*−*t*
define γ **where** γ ≡ *real l* / (*real k* + *real l*)
have $\gamma < 1$
using ‹*t < k*› **by** (*simp add*: γ*-def*)
have *nV div 2* ≤ *card X0*
using *card-X0* **by** *linarith*
then have *112*: *log 2* (*Suc nV*) ≤ *k* ∗ *log 2* (*1*/μ) + *t* ∗ *log 2* (*1* / (*1*−μ)) + *s* ∗ *log 2* (*ratio* μ *s t*)
\+ *ok-fun-11-2* μ *k*
using *From-11-2* [*OF - big11-2*] *p0-half* μ
unfolding *s-def t-def p0-def* $\mathcal{R}$*-def* $\mathcal{S}$*-def* **by** *force*
have *log 2* (*Suc nV*) / *k* ≤ *log 2* (*1*/μ) + *x* ∗ *log 2* (*1* / (*1*−μ)) + *y* ∗ *log 2* (*ratio* μ *s t*)
\+ *ok-fun-11-2* μ *k* / *k*
using ‹*k*>*0*› *divide-right-mono* [*OF 112*, *of k*]
by (*simp add*: *add-divide-distrib x-def y-def*)
also have … = *GG* μ *x y* + *ok-fun-11-2* μ *k* / *k*
by (*metis GG-def sts times-divide-eq-right*)
also have … ≤ *GG* μ *x y* + *ok-fun-11-1* μ *k* / *k*
by (*simp add*: *ok-fun-11-1-def divide-right-mono*)
finally have *le-GG*: *log 2* (*Suc nV*) / *k* ≤ *GG* μ *x y* + *ok-fun-11-1* μ *k* / *k* **.**

have *log 2* (*Suc nV*) / *k* ≤ *log 2* (*RN k* (*k*−*t*)) / *k* + *x* + *y* + (*2* − *ok-fun-61 k*) / *k*
using ‹*k*>*0*› *divide-right-mono* [*OF* §, *of k*] *add-divide-distrib x-def y-def*

by (*smt* (*verit*) *add-uminus-conv-diff of-nat-0-le-iff*)
also have ... = *FF k x y* + (*2* − *ok-fun-61 k*) / *k*
by (*simp add*: *FF-def x-def k-minus-t*)
finally have *DD*: *log 2* (*Suc nV*) / *k* ≤ *FF k x y* + (*2* − *ok-fun-61 k*) / *k* **.**

have *RN k k* > *0*
by (*metis RN-eq-0-iff* ‹*k*>*0*› *gr0I*)
moreover have *log 2* (*Suc nV*) / *k* ≤ *ffGG μ x y* + *η*
proof (*cases x* < *0.99*) — a further case split that gives a lower bound for gamma
case *True*
have ‡: *Big-Closer-10-1* (*min γ 0.07*) (*nat* ⌈*γ* ∗ *real k* / (*1* − *γ*)⌉)
proof (*intro Big-Closer-10-1-upward* [*OF big101*])
show *1*/*101* ≤ *min γ 0.07*
using ‹*k*>*0*› ‹*t*<*k*› *True* **by** (*simp add*: *γ-def l-def x-def divide-simps*)
with ‹*γ* < *1*› *less-eq-real-def* **have** *k*/*100* ≤ *γ* ∗ *k* / (*1* − *γ*)
by (*fastforce simp*: *field-simps*)
then show *nat* ⌈*k*/*100*⌉ ≤ *nat* ⌈*γ* ∗ *k* / (*1* − *γ*)⌉
using *ceiling-mono nat-mono* **by** *blast*
qed
have *122*: *FF k x y* ≤ *ff x y* + *η*′
proof −
have *FF k x y* ≤ *f1 x y*
using *x01 y01*
by (*intro FF-le-f1*) *auto*
moreover
have *FF k x y* ≤ *f2 x y* + *ok-fun-10-1 γ k* / (*k* ∗ *ln 2*) **if** *x* ≥ *3*/*4*
unfolding *γ-def*
proof (*intro FF-le-f2 that*)
have *γ* = (*1*−*x*) / (*2*−*x*)
using ‹*0* < *k*› ‹*t* < *k*› **by** (*simp add*: *l-def γ-def x-def divide-simps*)
then have *γ* ≤ *1*/*5*
using *that* ‹*x*<*1*› **by** *simp*
show *real l* = *real k* − *x* ∗ *real k*
using ‹*t* < *k*› **by** (*simp add*: *l-def x-def*)
show *0* < *l* / (*k* + *real l*)
using ‹*t* < *k*› *l-def* **by** *auto*
qed (*use x01 y01 p0-min12* **in** *auto*)
moreover have *ok-fun-10-1 γ k* / (*k* ∗ *ln 2*) ≤ *η*′
using ‡ *ok-fun-10-1-le* **by** (*simp add*: *ok-fun-10-1-def*)
ultimately show *?thesis*
using *η*′ **by** (*auto simp*: *ff-def*)
qed
have *log 2* (*Suc nV*) / *k* ≤ *ff x y* + *η*′ + (*2* − *ok-fun-61 k*) / *k*
using *122 DD* **by** *linarith*
also have ... ≤ *ff x y* + *η*′ + *ok-fun-11-1 μ k* / *k*
by (*simp add*: *ok-fun-11-1-def divide-right-mono*)
finally have *le-ff*: *log 2* (*Suc nV*) / *k* ≤ *ff x y* + *η*′ + *ok-fun-11-1 μ k* / *k* **.**
then show *?thesis*

using *η ok111-le le-ff le-GG* **unfolding** *η′-def ffGG-def* **by** *linarith*
next
case *False* — in this case, we can use the existing bound involving *f1*
have *log 2 (Suc nV) / k ≤ FF k x y + (2 − ok-fun-61 k) / k*
by (*metis DD*)
also have ... *≤ f1 x y + (2 − ok-fun-61 k) / k*
using *x01 y01 FF-le-f1 [of x y]* **by** *simp*
also have ... *≤ 1.9 + (2 − ok-fun-61 k) / k*
using *x01 y01* **by** (*smt (verit) False ‹y ≤ 3/4› f1-le-19*)
also have ... *≤ ffGG μ x y + η*
by (*smt (verit) P0-min.intro P0-min.ok-fun-11-1-def η′(1) η′-def divide-right-mono ffGG-def field-sum-of-halves of-nat-0-le-iff ok111-le p0-min(1) p0-min(2)*)
finally show *?thesis* **.**
qed
ultimately have *log 2 (RN k k) / k ≤ ffGG μ x y + η*
using *ge-RN ‹k>0›*
by (*smt (verit, best) Transcendental.log-mono divide-right-mono of-nat-0-less-iff of-nat-mono*)
also have ... *≤ w*
unfolding *w-def*
proof (*intro cSup-upper2*)
have *y ∈ {0..3/4}*
using *divide-right-mono [OF †, of k] ‹k>0›* **by** (*simp add: x-def y-def*)
then show *ffGG μ x y + η ∈ (λy. ffGG μ x y + η) ‘ {0..3/4}*
by *blast*
next
show *bdd-above ((λy. ffGG μ x y + η) ‘ {0..3/4})*
by (*simp add: bdd-above-ff-GG less-imp-le x01*)
qed *auto*
finally show *log 2 (real (RN k k)) / k ≤ w* **.**
next
show *bdd-above ((λx. ⨆y∈{0..3/4}. ffGG μ x y + η) ‘ {0..1})*
by (*auto intro: bdd-above-SUP-ff-GG*)
qed
qed

theorem (**in** *P0-min*) *From-11-1*:
assumes *μ: 0 < μ μ ≤ 2/5* **and** *0 < η η ≤ 1/12*
and *p0-min12: p0-min ≤ 1/2* **and** *big: Big-From-11-1 η μ k*
shows *log 2 (RN k k) / k ≤ (SUP x ∈ {0..1}. SUP y ∈ {0..3/4}. ffGG μ x y + η)*
proof −
have *k≥3*
using *big* **by** (*auto simp: Big-From-11-1-def*)
define *n* **where** *n ≡ RN k k − 1*
define *V* **where** *V ≡ {..<n}*
define *E* **where** *E ≡ all-edges V*
interpret *Book-Basis V E*
proof qed (*auto simp: V-def E-def comp-sgraph.wellformed comp-sgraph.two-edges*)

have *RN k k* ≥ *3*
using ‹*k*≥*3*› *RN-3plus le-trans* **by** *blast*
then have *n* < *RN k k*
by (*simp add*: *n-def*)
moreover have [*simp*]: *nV* = *n*
by (*simp add*: *V-def*)
ultimately obtain *Red Blue*
where *Red-E*: *Red* ⊆ *E* **and** *Blue-def*: *Blue* = *E**Red*
and *no-Red-K*: ¬ (∃ *K*. *size-clique k K Red*)
and *no-Blue-K*: ¬ (∃ *K*. *size-clique k K Blue*)
by (*metis* ‹*n* < *RN k k*› *less-RN-Red-Blue*)
have *Blue-E*: *Blue* ⊆ *E* **and** *disjnt-Red-Blue*: *disjnt Red Blue* **and** *Blue-eq*: *Blue* = *all-edges V* \ *Red*
using *complete* **by** (*auto simp*: *Blue-def disjnt-iff E-def*)
have *nV* > *1*
using ‹*RN k k* ≥ *3*› ‹*nV*=*n*› *n-def* **by** *linarith*
with *graph-size* **have** *graph-size* > *0*
by *simp*
then have *graph-density E* = *1*
by (*simp add*: *graph-density-def*)
then have *graph-density Red* + *graph-density Blue* = *1*
using *graph-density-Un* [*OF disjnt-Red-Blue*] **by** (*simp add*: *Blue-def Red-E Un-absorb1*)
then consider (*Red*) *graph-density Red* ≥ *1/2* | (*Blue*) *graph-density Blue* ≥ *1/2*
by *force*
then show *?thesis*
proof *cases*
case *Red*
show *?thesis*
proof (*intro From-11-1-Body*)
next
show *RN k k* ≤ *Suc nV*
by (*simp add*: *n-def*)
show ∄*K*. *size-clique k K Red*
using *no-Red-K* **by** *blast*
show ∄*K*. *size-clique k K Blue*
using *no-Blue-K* **by** *blast*
qed (*use Red Red-E Blue-def assms* **in** *auto*)
next
case *Blue*
show *?thesis*
proof (*intro From-11-1-Body*)
show *RN k k* ≤ *Suc nV*
by (*simp add*: *n-def*)
show *Blue* ⊆ *E*
by (*simp add*: *Blue-E*)
show *Red* = *E* \ *Blue*

```
      by (simp add: Blue-def Red-E double-diff)
    show ∄K. size-clique k K Red
      using no-Red-K by blast
    show ∄K. size-clique k K Blue
      using no-Blue-K by blast
  qed (use Blue Red-E Blue-def assms in auto)
 qed
qed
```

11.2 The monster calculation from appendix A

11.2.1 Observation A.1

```
lemma gg-increasing:
  assumes x ≤ x' 0 ≤ x 0 ≤ y
  shows gg x y ≤ gg x' y
proof (cases y=0)
  case False
  with assms show ?thesis
    unfolding gg-eq by (intro add-mono mult-left-mono divide-right-mono Tran-
scendental.log-mono) auto
qed (auto simp: gg-eq assms)
```

Thanks to Manuel Eberl

```
lemma continuous-on-x-ln: continuous-on {0..} (λx::real. x * ln x)
proof –
  have continuous (at x within {0..}) (λx. x * ln x)
    if x ≥ 0 for x :: real
  proof (cases x = 0)
    case True
    have continuous (at-right 0) (λx::real. x * ln x)
      unfolding continuous-within by real-asymp
    thus ?thesis
      using True by (simp add: at-within-Ici-at-right)
  qed (auto intro!: continuous-intros)
  thus ?thesis
    by (simp add: continuous-on-eq-continuous-within)
qed

lemma continuous-on-f1: continuous-on {..1} (λx. f1 x y)
proof –
  have §: (λx::real. (1 − 1/(2−x)) * ln (1 − 1/(2−x))) = (λx. x * ln x) o (λx.
1 − 1/(2−x))
    by (simp add: o-def)
 have cont-xln: continuous-on {..1} (λx::real. (1 − 1/(2−x)) * ln (1 − 1/(2−x)))
    unfolding §
  proof (rule continuous-intros)
    show continuous-on {..1::real} (λx. 1 − 1/(2−x))
      by (intro continuous-intros) auto
  next
```

    show *continuous-on* ((λx::real. 1 − 1/(2−x)) ‘ {..1}) (λx. x ∗ ln x)
      by (*rule continuous-on-subset* [*OF continuous-on-x-ln*]) *auto*
  qed
  show *?thesis*
    apply (*simp add*: *f1-def H-def log-def*)
    by (*intro continuous-on-subset* [*OF cont-xln*] *continuous-intros*) *auto*
qed

definition *df1* **where** *df1* ≡ λx. *log 2* (2 ∗ ((1−x) / (2−x)))

lemma *Df1* [*derivative-intros*]:
  assumes x<1
  shows ((λx. *f1 x y*) *has-real-derivative df1 x*) (*at x*)
proof −
  have (2 − x ∗ 2) = 2 ∗ (1−x)
    by *simp*
  then have [*simp*]: *log 2* (2 − x ∗ 2) = *log 2* (1−x) + 1
    using *log-mult* [*of 2 1−x 2*] *assms* **by** (*smt* (*verit*, *best*) *log-eq-one*)
  show *?thesis*
    using *assms*
    unfolding *f1-def H-def df1-def*
    apply −
    apply (*rule derivative-eq-intros* | *simp*)+
    apply (*simp add*: *log-divide divide-simps*)
    apply (*simp add*: *algebra-simps*)
    done
qed

definition *delta* **where** *delta* ≡ λu::real. 1 / (*ln 2* ∗ 40 ∗ (2 − u)^2)

lemma *Df2*:
  assumes 1/2≤x x<1
  shows ((λx. *f2 x y*) *has-real-derivative df1 x* + *delta x*) (*at x*)
  using *assms* **unfolding** *f2-def delta-def*
  apply −
  apply (*rule derivative-eq-intros Df1* | *simp*)+
  apply (*simp add*: *divide-simps power2-eq-square*)
  done

lemma *antimono-on-ff*:
  assumes 0 ≤ y y < 1
  shows *antimono-on* {1/2..1} (λx. *ff x y*)
proof −
  have §: 1 − 1 / (2−x) = (1−x) / (2−x) **if** x<2 **for** x::*real*
    using *that* **by** (*simp add*: *divide-simps*)
  have *f1*: *f1 x′ y* ≤ *f1 x y*
    if x ∈ {1/2..1} x′ ∈ {1/2..1} x ≤ x′ x′ ≤ 1 **for** x x′::*real*
  proof (*rule DERIV-nonpos-imp-decreasing-open* [*OF* ‹x ≤ x′›, **where** f = λx. *f1 x y*])

```
    fix u :: real
    assume x < u u < x'
    with that show ∃D. ((λx. f1 x y) has-real-derivative D) (at u) ∧ D ≤ 0
      by − (rule exI conjI Df1 [unfolded df1-def] | simp)+
  next
    show continuous-on {x..x'} (λx. f1 x y)
      using that by (intro continuous-on-subset [OF continuous-on-f1]) auto
  qed
  have f1f2: f2 x' y ≤ f1 x y
    if x ∈ {1/2..1} x' ∈ {1/2..1} x ≤ x' x < 3/4 ¬ x' < 3/4 for x x'::real
    using that
    apply (simp add: f2-def)
    by (smt (verit, best) divide-nonneg-nonneg f1 ln-le-zero-iff pos-prod-lt that)

  have f2: f2 x' y ≤ f2 x y
    if A: x ∈ {1/2..1} x' ∈ {1/2..1} x ≤ x' and B: ¬ x < 3/4 for x x'::real
  proof (rule DERIV-nonpos-imp-decreasing-open [OF ‹x ≤ x'› , where f = λx.
f2 x y])
    fix u :: real
    assume u: x < u u < x'
    have ((λx. f2 x y) has-real-derivative df1 u + delta u) (at u)
      using u that by (intro Df2) auto
    moreover have df1 u + delta u ≤ 0
    proof −
      have df1 (1/2) ≤ −1/2
        unfolding df1-def by (approximation 20)
      moreover have df1 u ≤ df1 (1/2)
        using u that unfolding df1-def
        by (intro Transcendental.log-mono) (auto simp: divide-simps)
      moreover have delta 1 ≤ 0.04
        unfolding delta-def by (approximation 4)
      moreover have delta u ≤ delta 1
        using u that by (auto simp: delta-def divide-simps)
      ultimately show ?thesis
        by auto
    qed
    ultimately show ∃D. ((λx. f2 x y) has-real-derivative D) (at u) ∧ D ≤ 0
      by blast
  next
    show continuous-on {x..x'} (λx. f2 x y)
      unfolding f2-def
     using that by (intro continuous-on-subset [OF continuous-on-f1] continuous-intros)
auto
  qed
  show ?thesis
    using f1 f1f2 f2 by (simp add: monotone-on-def ff-def)
qed
```

11.2.2 Claims A.2–A.4

Called simply x in the paper, but are you kidding me?

```
definition x-of ≡ λy::real. 3*y/5 + 0.5454

lemma x-of: x-of ∈ {0..3/4} → {1/2..1}
  by (simp add: x-of-def)

definition y-of ≡ λx::real. 5 * x/3 − 0.909

lemma y-of-x-of [simp]: y-of (x-of y) = y
  by (simp add: x-of-def y-of-def add-divide-distrib)

lemma x-of-y-of [simp]: x-of (y-of x) = x
  by (simp add: x-of-def y-of-def divide-simps)

lemma Df1-y [derivative-intros]:
  assumes x<1
  shows ((λx. f1 x (y-of x)) has-real-derivative 5/3 + df1 x) (at x)
proof −
  have (2 − x * 2) = 2 * (1−x)
    by simp
  then have [simp]: log 2 (2 − x * 2) = log 2 (1−x) + 1
    using log-mult [of 2 1−x 2] assms by (smt (verit, best) log-eq-one)
  show ?thesis
    using assms
    unfolding f1-def y-of-def H-def df1-def
    apply −
    apply (rule derivative-eq-intros refl | simp)+
    apply (simp add: log-divide divide-simps)
    apply (simp add: algebra-simps)
    done
qed

lemma Df2-y [derivative-intros]:
  assumes 1/2≤x x<1
  shows ((λx. f2 x (y-of x)) has-real-derivative 5/3 + df1 x + delta x) (at x)
  using assms unfolding f2-def delta-def
  apply −
  apply (rule derivative-eq-intros Df1 | simp)+
  apply (simp add: divide-simps power2-eq-square)
  done

definition Dg-x ≡ λy. 3 * log 2 (5/3) / 5 + log 2 ((2727 + y * 8000) / (y *
12500))
                − 2727 / (ln 2 * (2727 + y * 8000))

lemma Dg-x [derivative-intros]:
  assumes y ∈ {0<..<3/4}
```

```
  shows ((λy. gg (x-of y) y) has-real-derivative Dg-x y) (at y)
  using assms
  unfolding x-of-def gg-def GG-def Dg-x-def
  apply −
  apply (rule derivative-eq-intros refl | simp)+
  apply (simp add: field-simps)
  done
```

Claim A2 is difficult because it comes *real close*: max value = 1.999281, when y = 0.4339. There is no simple closed form for the maximum point (where the derivative goes to 0).

Due to the singularity at zero, we need to cover the zero case analytically, but at least interval arithmetic covers the maximum point

```
lemma A2:
  assumes y ∈ {0..3/4}
  shows gg (x-of y) y ≤ 2 − 1/2^11
proof −
  have ?thesis if y ∈ {0..1/10}
  proof −
    have gg (x-of y) y ≤ gg (x-of (1/10)) (1/10)
    proof (rule DERIV-nonneg-imp-increasing-open [of y 1/10])
      fix y' :: real
      assume y': y < y' y' < 1/10
      then have y'>0
        using that by auto
      show ∃D. ((λu. gg (x-of u) u) has-real-derivative D) (at y') ∧ 0 ≤ D
      proof (intro exI conjI)
        show ((λu. gg (x-of u) u) has-real-derivative Dg-x y') (at y')
          using y' that by (intro derivative-eq-intros) auto
      next
        define Num where Num ≡ 3 ∗ log 2 (5/3) / 5 ∗ (ln 2 ∗ (2727 + y' ∗
8000)) + log 2 ((2727 + y' ∗ 8000) / (y' ∗ 12500)) ∗ (ln 2 ∗ (2727 + y' ∗ 8000))
− 2727
        have A: 835.81 ≤ 3 ∗ log 2 (5/3) / 5 ∗ ln 2 ∗ 2727
          by (approximation 25)
        have B: 2451.9 ≤ 3 ∗ log 2 (5/3) / 5 ∗ ln 2 ∗ 8000
          by (approximation 25)
        have C: Dg-x y' = Num / (ln 2 ∗ (2727 + y' ∗ 8000))
       using ‹y'>0› by (simp add: Dg-x-def Num-def add-divide-distrib diff-divide-distrib)
        have 0 ≤ −1891.19 + log 2 (2727 / 1250) ∗ (ln 2 ∗ (2727))
          by (approximation 6)
        also have ... ≤ −1891.19 + 2451.9 ∗ y' + log 2 ((2727 + y' ∗ 8000) /
(y' ∗ 12500)) ∗ (ln 2 ∗ (2727 + y' ∗ 8000))
          using y' ‹0 < y'›
         by (intro add-mono mult-mono Transcendental.log-mono frac-le order.refl)
auto
        also have ... = 835.81 + 2451.9 ∗ y' + log 2 ((2727 + y' ∗ 8000) / (y'
∗ 12500)) ∗ (ln 2 ∗ (2727 + y' ∗ 8000))
            − 2727
```

```
        by simp
      also have ... ≤ Num
        using A mult-right-mono [OF B, of y'] ‹y'>0›
        unfolding Num-def ring-distribs
        by (intro add-mono diff-mono order.refl) (auto simp: mult-ac)
      finally have Num ≥ 0 .
      with C show 0 ≤ Dg-x y'
        using ‹0 < y'› by auto
    qed
  next
    let ?f = λx. x * log 2 ((16*x/5 + 2727/2500) / (5*x))
    have †: continuous-on {0..} ?f
    proof −
      have continuous (at x within {0..}) ?f
        if x ≥ 0 for x :: real
      proof (cases x = 0)
        case True
        have continuous (at-right 0) ?f
          unfolding continuous-within by real-asymp
        thus ?thesis
          using True by (simp add: at-within-Ici-at-right)
      qed (use that in ‹auto intro!: continuous-intros›)
      thus ?thesis
        by (simp add: continuous-on-eq-continuous-within)
    qed
    show continuous-on {y..1/10} (λy. gg (x-of y) y)
      unfolding gg-eq x-of-def using that
      by (force intro: continuous-on-subset [OF †] continuous-intros)
  qed (use that in auto)
  also have ... ≤ 2 − 1/2^11
    unfolding gg-eq x-of-def by (approximation 10)
  finally show ?thesis .
qed
moreover
have ?thesis if y ∈ {1/10 .. 3/4}
  using that unfolding gg-eq x-of-def
  by (approximation 24 splitting: y = 12)   — many thanks to Fabian Immler
ultimately show ?thesis
  by (meson assms atLeastAtMost-iff linear)
qed

lemma A3:
  assumes y ∈ {0..0.341}
  shows f1 (x-of y) y ≤ 2 − 1/2^11
proof −
  define D where D ≡ λx. 5/3 + df1 x
  define I where I ≡ {0.5454 .. 3/4::real}
  define x where x ≡ x-of y
  then have yeq: y = y-of x
```

```
    by (metis y-of-x-of)
  have x ∈ {x-of 0 .. x-of 0.341}
    using assms by (simp add: x-def x-of-def)
  then have x: x ∈ I
    by (simp add: x-of-def I-def)
  have D: ((λx. f1 x (y-of x)) has-real-derivative D x) (at x) if x ∈ I for x
    using that Df1-y by (force simp: D-def I-def)
  have Dgt0: D x ≥ 0 if x ∈ I for x
    using that unfolding D-def df1-def I-def by (approximation 10)
  have f1 x y = f1 x (y-of x)
    by (simp add: yeq)
  also have ... ≤ f1 (3/4) (y-of (3/4))
    using x Dgt0
    by (force simp: I-def intro!: D DERIV-nonneg-imp-nondecreasing [where f =
λx. f1 x (y-of x)])
  also have ... < 1.994
    by (simp add: f1-def H-def y-of-def) (approximation 50)
  also have ... < 2 − 1/2^11
    by (approximation 50)
  finally show ?thesis
    using x-def by auto
qed
```

This one also comes close: max value = 1.999271, when y = 0.4526. The specified upper bound is 1.99951

```
lemma A4:
  assumes y ∈ {0.341..3/4}
  shows f2 (x-of y) y ≤ 2 − 1/2^11
  unfolding f2-def f1-def x-of-def H-def
  using assms by (approximation 18 splitting: y = 13)


context P0-min
begin
```

The truly horrible Lemma 12.3

```
lemma 123:
  assumes δ ≤ 1 / 2^11
  shows (SUP x ∈ {0..1}. SUP y ∈ {0..3/4}. ffGG (2/5) x y) ≤ 2−δ
proof −
  have min (ff x y) (gg x y) ≤ 2 − 1/2^11 if x ∈ {0..1} y ∈ {0..3/4} for x y
  proof (cases x ≤ x-of y)
    case True
    with that have gg x y ≤ gg (x-of y) y
      by (intro gg-increasing) auto
    with A2 that show ?thesis
      by fastforce
  next
    case False
```

```
    with that have ff x y ≤ ff (x-of y) y
      by (intro monotone-onD [OF antimono-on-ff]) (auto simp: x-of-def)
    also have ... ≤ 2 − 1/2^11
    proof (cases x-of y < 3/4)
      case True
      with that have f1 (x-of y) y ≤ 2 − 1/2^11
        by (intro A3) (auto simp: x-of-def)
      then show ?thesis
        using True ff-def by presburger
    next
      case False
      with that have f2 (x-of y) y ≤ 2 − 1/2^11
        by (intro A4) (auto simp: x-of-def)
      then show ?thesis
        using False ff-def by presburger
    qed
    finally show ?thesis
      by linarith
  qed
  moreover have 2 − 1/2^11 ≤ 2−δ
    using assms by auto
  ultimately show ?thesis
    by (fastforce simp: ffGG-def gg-def intro!: cSUP-least)
qed

end
```

11.3 Concluding the proof

we subtract a tiny bit, as we seem to need this gap

```
definition delta'::real where delta' ≡ 1 / 2^11 − 1 / 2^18

lemma Aux-1-1:
  assumes p0-min12: p0-min ≤ 1/2
  shows ∀∞k. log 2 (RN k k) / k ≤ 2 − delta'
proof −
  define p0-min::real where p0-min ≡ 1/2
  interpret P0-min p0-min
  proof qed (auto simp: p0-min-def)
  define δ::real where δ ≡ 1 / 2^11
  define η::real where η ≡ 1 / 2^18
  have η: 0 < η η ≤ 1/12
    by (auto simp: η-def)
  define μ::real where μ ≡ 2/5
  have ∀∞k. Big-From-11-1 η μ k
    unfolding μ-def using η by (intro Big-From-11-1) auto
  moreover have log 2 (real (RN k k)) / k ≤ 2−δ + η if Big-From-11-1 η μ k
for k
  proof −
```

```
    have ∗: (⨆y∈{0..3/4}. ffGG μ x y + η) = (⨆y∈{0..3/4}. ffGG μ x y) + η
      if x≤1 for x
      using bdd-above-ff-GG [OF that, of 3/4 μ 0]
      by (simp add: add.commute [of - η] Sup-add-eq)
    have log 2 (RN k k) / k ≤ (SUP x ∈ {0..1}. SUP y ∈ {0..3/4}. ffGG μ x y
+ η)
      using that p0-min12 η μ-def
      by (intro From-11-1) (auto simp: p0-min-def)
    also have ... ≤ (SUP x ∈ {0..1}. (SUP y ∈ {0..3/4}. ffGG μ x y) + η)
    proof (intro cSUP-subset-mono bdd-above.I2 [where M = 4+η])
      fix x :: real
      assume x: x ∈ {0..1}
      have (⨆y∈{0..3/4}. ffGG μ x y + η) ≤ 4 + η
        using bdd-above-ff-GG ff-GG-bound x by (simp add: cSup-le-iff)
      with ∗ x show (⨆y∈{0..3/4}. ffGG μ x y) + η ≤ 4 + η
        by simp
    qed (use ∗ in auto)
    also have ... = (SUP x ∈ {0..1}. SUP y ∈ {0..3/4}. ffGG μ x y) + η
      using bdd-above-SUP-ff-GG [of 3/4  μ 0]
      by (simp add: add.commute [of - η] Sup-add-eq)
    also have ... ≤ 2−δ + η
      using 123 [of 1 / 2^11]
      unfolding δ-def ffGG-def by (auto simp: δ-def ffGG-def μ-def)
    finally show ?thesis .
  qed
  ultimately have ∀∞k. log 2 (RN k k) / k ≤ 2−δ + η
    by (metis (lifting) eventually-mono)
  then show ?thesis
    by (simp add: δ-def η-def delta′-def)
qed
```

Main theorem 1.1: the exponent is approximately 3.9987

```
theorem Main-1-1:
  obtains ε::real where ε>0 ∀∞k. RN k k ≤ (4−ε)^k
proof
  let ?ε = 0.00134::real
  have ∀∞k. k>0 ∧ log 2 (RN k k) / k ≤ 2 − delta′
    unfolding eventually-conj-iff using Aux-1-1 eventually-gt-at-top by blast
  then have ∀∞k. RN k k ≤ (2 powr (2−delta′)) ^ k
  proof (eventually-elim)
    case (elim k)
    then have log 2 (RN k k) ≤ (2−delta′) ∗ k
      by (meson of-nat-0-less-iff pos-divide-le-eq)
    then have RN k k ≤ 2 powr ((2−delta′) ∗ k)
      by (smt (verit, best) Transcendental.log-le-iff powr-ge-zero)
    then show RN k k ≤ (2 powr (2−delta′)) ^ k
      by (simp add: mult.commute powr-power)
  qed
  moreover have 2 powr (2−delta′) ≤ 4 − ?ε
```

unfolding *delta'-def* **by** (*approximation 25*)
ultimately show $\forall^\infty k$. *real* (*RN k k*) $\leq$ (*4* − *?ε*) ^ *k*
by (*smt* (*verit*) *power-mono powr-ge-zero eventually-mono*)
qed *auto*

end

References

[1] M. Campos, S. Griffiths, R. Morris, and J. Sahasrabudhe. An exponential improvement for diagonal Ramsey, 2023. arXiv, 2303.09521.