Distributed Distinct Elements

Emin Karayel

March 17, 2025

Abstract

This entry formalizes a randomized cardinality estimation data structure with asymptotically optimal space usage. It is inspired by the streaming algorithm presented by Błasiok [3] in 2018. His work closed the gap between the best-known lower bound and upper bound after a long line of research started by Flajolet and Martin [4] in 1984 and was to first to apply expander graphs (in addition to hash families) to the problem. The formalized algorithm has two improvements compared to the algorithm by Błasiok. It supports operation in parallel mode, and it relies on a simpler pseudo-random construction avoiding the use of code based extractors.

Contents

1 Introduction

The algorithm is described as functional data strucutures, given a seed which needs to be choosen uniformly from a initial segment of the natural numbers and globally, there are three functions:

- single - given the seed and an element from the universe computes a sketch for that singleton set
- merge - computes a sketch based on two input sketches and returns a sketch representing the union set
- estimate - computes an estimate for the cardinality of the set represented by a sketch

The main point is that a sketch requires $\mathcal{O}(\delta^{-2} \ln(\varepsilon^{-1}) + \ln n)$ space where n is the universe size, δ is the desired relative accuracy and ε is the desired failure probability. Note that it is easy to see that an exact solution would necessarily require $\mathcal{O}(n)$ bits.

The algorithm is split into two parts an inner algorithm, described in Section 6, which itself is already a full cardinality estimation algorithm, however its space usage is below optimal. The outer algorithm is introduced in Section 10, which runs mutiple copies of the inner algorithm with carefully chosen inner parameters.

As mentioned in the abstract the algorithm is inspired by the solution to the streaming version of the problem by Błasiok [3] in 2020. His work builds on a long line of reasarch starting in 1985 [4, 1, 2, 7, 11, 5].

In an earlier AFP entry [9] I have formalized an earlier cardinality estimation algorithm based on the work by Bar-Yossef et al. [2] in 2002. Since then I have addressed the existence of finite fields for higher prime powers and expander graphs [8, 10]. Building on these results, the formalization of this more advanced solution presented here became possible.

The solution described here improves on the algorithms described by Błasiok in two ways (without comprising its optimal space usage). It can be used in a parallel mode of operation. Moreover the pseudo-random construction used is simpler than the solution described by Błasiok — who uses an extractor based on Parvaresh-Vardy codes [6] to sample random walks in an expander graph, which are then sub-sampled and then the walks are used to sample seeds for hash functions. In the solution presented here neither the sub-sampling step nor the extractor is needed, instead a two-stage expander construction is used, this means that the nodes of the first expander correspond to the walks in a second expander graph. The latters nodes correspond to seeds of hash functions (as in Błasiok's solution).

The modification needed to support a parallel mode of operation is a change in the failure strategy of the solution presented in Kane et al., which is the event when the data in the sketch reequires too much space. The main issue is that in the parallel case the number of states the algorithm might reach is not bounded by the universe size and thus an estimate they make for the probability of the failure event does not transfer to the parallel case. To solve that the algorithm in this work is more conservative. Instead of failing out-right it instead increases a cutoff threshold. For which it is then possible to show an upper estimate independent of the number of reached states.

2 Preliminary Results

This section contains various short preliminary results used in the sections below.

theory *Distributed-Distinct-Elements-Preliminary*
imports
Frequency-Moments.Frequency-Moments-Preliminary-Results
Universal-Hash-Families.Universal-Hash-Families-More-Product-PMF
Median-Method.Median
Expander-Graphs.Extra-Congruence-Method
Expander-Graphs.Constructive-Chernoff-Bound
Frequency-Moments.Landau-Ext
Stirling-Formula.Stirling-Formula
begin

unbundle *intro-cong-syntax*

lemma *pmf-rev-mono*:
assumes $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow x \notin Q \Longrightarrow x \notin P$
shows *measure p P* $\leq$ *measure p Q*
using *assms* **by** (*intro pmf-mono*) *blast*

lemma *pmf-exp-mono*:
fixes *f g* :: *'a* $\Rightarrow$ *real*
assumes *integrable* (*measure-pmf p*) *f integrable* (*measure-pmf p*) *g*
assumes $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow$ *f x* $\leq$ *g x*
shows *integral*^{L} (*measure-pmf p*) *f* $\leq$ *integral*^{L} (*measure-pmf p*) *g*
using *assms* **by** (*intro integral-mono-AE AE-pmfI*) *auto*

lemma *pmf-markov*:
assumes *integrable* (*measure-pmf p*) *f c* $>$ *0*
assumes $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow$ *f x* $\geq$ *0*
shows *measure p* $\{\omega.\ f\ \omega \geq c\} \leq (\int \omega.\ f\ \omega\ \partial p)/\ c$ (**is** *?L* $\leq$ *?R*)
proof −
have *a*:*AE* ω *in* (*measure-pmf p*). *0* $\leq$ *f* ω
by (*intro AE-pmfI assms*(*3*))
have *b*:{} $\in$ *measure-pmf.events p*
unfolding *assms*(*1*) **by** *simp*

have *?L* = $\mathcal{P}(\omega$ *in* (*measure-pmf p*). *f* $\omega \geq c)$
using *assms*(*1*) **by** *simp*
also have ... $\leq$ *?R*
by (*intro integral-Markov-inequality-measure*[*OF - b*] *assms a*)
finally show *?thesis* **by** *simp*
qed

lemma *pair-pmf-prob-left*:
measure-pmf.prob (*pair-pmf A B*) $\{\omega.\ P\ (fst\ \omega)\}$ = *measure-pmf.prob A* $\{\omega.\ P\ \omega\}$ (**is** *?L* = *?R*)
proof −
have *?L* = *measure-pmf.prob* (*map-pmf fst* (*pair-pmf A B*)) $\{\omega.\ P\ \omega\}$
by (*subst measure-map-pmf*) *simp*
also have ... = *?R*
by (*subst map-fst-pair-pmf*) *simp*
finally show *?thesis* **by** *simp*
qed

lemma *pmf-exp-of-fin-function*:
assumes *finite A g* ‘ *set-pmf p* $\subseteq$ *A*
shows $(\int \omega.\ f\ (g\ \omega)\ \partial p) = (\sum y \in A.\ f\ y * measure\ p\ \{\omega.\ g\ \omega = y\})$
(**is** *?L* = *?R*)
proof −
have *?L* = *integral*^{L} (*map-pmf g p*) *f*
using *integral-map-pmf assms* **by** *simp*
also have ... = $(\sum a \in A.\ f\ a * pmf\ (map\text{-}pmf\ g\ p)\ a)$
using *assms*
by (*intro integral-measure-pmf-real*) *auto*
also have ... = $(\sum y \in A.\ f\ y *$ *measure p* $(g -‘ \{y\}))$
unfolding *assms*(*1*) **by** (*intro-cong* [σ_2 (∗)] *more*:*sum.cong pmf-map*)
also have ... = *?R*
by (*intro sum.cong*) (*auto simp add*: *vimage-def*)
finally show *?thesis* **by** *simp*
qed

Cardinality rules for distinct/ordered pairs of a set without the finiteness constraint - to use in simplification:

lemma *card-distinct-pairs*:
  card $\{x \in B \times B.\ fst\ x \neq snd\ x\} = card\ B\hat{}2 - card\ B$ (**is** *card ?L = ?R*)
proof (*cases finite B*)
  case *True*
  include *intro-cong-syntax*
  have *card ?L = card* $(B \times B - (\lambda x.\ (x,x))\ `\ B)$
    by (*intro arg-cong*[**where** *f=card*]) *auto*
  also have *... = card* $(B \times B) - card\ ((\lambda x.\ (x,x))\ `\ B)$
    by (*intro card-Diff-subset finite-imageI True image-subsetI*) *auto*
  also have *... = ?R*
    using *True* **by** (*intro-cong* $[\sigma_2\ (-)]$ *more*: *card-image*)
      (*auto simp add:power2-eq-square inj-on-def*)
  finally show *?thesis* **by** *simp*
next
  case *False*
  then obtain *p* **where** *p-in*: $p \in B$ **by** *fastforce*
  have *False* **if** *finite ?L*
  proof −
    have $(\lambda x.\ (p,x))\ `\ (B - \{p\}) \subseteq ?L$
      using *p-in* **by** (*intro image-subsetI*) *auto*
    hence *finite* $((\lambda x.\ (p,x))\ `\ (B - \{p\}))$
      using *finite-subset that* **by** *auto*
    hence *finite* $(B - \{p\})$
      by (*rule finite-imageD*) (*simp add:inj-on-def*)
    hence *finite B*
      by *simp*
    thus *False* **using** *False* **by** *simp*
  qed
  hence *infinite ?L* **by** *auto*
  hence *card ?L = 0* **by** *simp*
  also have *... = ?R*
    using *False* **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *card-ordered-pairs'*:
  fixes *M* :: $('a$ ::*linorder*$)$ *set*
  shows *card* $\{(x,y) \in M \times M.\ x < y\} = card\ M * (card\ M - 1)\ /\ 2$
proof (*cases finite M*)
  case *True*
  show *?thesis* **using** *card-ordered-pairs*[*OF True*] **by** *linarith*
next
  case *False*
  then obtain *p* **where** *p-in*: $p \in M$ **by** *fastforce*
  let *?f*= $(\lambda x.\ if\ x < p\ then\ (x,p)\ else\ (p,x))$
  have *False* **if** *finite* $\{(x,y) \in M \times M.\ x < y\}$ (**is** *finite ?X*)
  proof −
    have *?f* ‘ $(M-\{p\}) \subseteq ?X$
      using *p-in* **by** (*intro image-subsetI*) *auto*
    hence *finite* (*?f* ‘ $(M-\{p\})$) **using** *that finite-subset* **by** *auto*
    moreover have *inj-on ?f* $(M-\{p\})$
      by (*intro inj-onI*) (*metis Pair-inject*)
    ultimately have *finite* $(M - \{p\})$
      using *finite-imageD* **by** *blast*
    hence *finite M*

```
    using finite-insert[where a=p and A=M−{p}] by simp
   thus False using False by simp
  qed
  hence infinite ?X by auto
  then show ?thesis using False by simp
qed
```

The following are versions of the mean value theorem, where the interval endpoints may be reversed.

```
lemma MVT-symmetric:
  assumes ⋀x. ⟦min a b ≤ x; x ≤ max a b⟧ ⟹ DERIV f x :> f' x
  shows ∃ z::real. min a b ≤ z ∧ z ≤ max a b ∧ (f b − f a = (b − a) * f' z)
proof −
  consider (a) a < b | (b) a = b | (c) a > b
    by argo
  then show ?thesis
  proof (cases)
    case a
    then obtain z :: real where r: a < z z < b f b − f a = (b − a) * f' z
      using assms MVT2[where a=a and b=b and f=f and f'=f'] by auto
    have a ≤ z z ≤ b using r(1,2) by auto
    thus ?thesis using a r(3) by auto
  next
    case b
    then show ?thesis by auto
  next
    case c
    then obtain z :: real where r: b < z z < a f a − f b = (a − b) * f' z
      using assms MVT2[where a=b and b=a and f=f and f'=f'] by auto
    have f b − f a = (b−a) * f' z using r by argo
    moreover have b ≤ z z ≤ a using r(1,2) by auto
    ultimately show ?thesis using c by auto
  qed
qed

lemma MVT-interval:
  fixes I :: real set
  assumes interval I a ∈ I b ∈ I
  assumes ⋀x. x ∈ I ⟹ DERIV f x :> f' x
  shows ∃ z. z ∈ I ∧ (f b − f a = (b − a) * f' z)
proof −
  have a:min a b ∈ I
    using assms(2,3) by (cases a < b) auto
  have b:max a b ∈ I
    using assms(2,3) by (cases a < b) auto
  have c:x ∈ {min a b..max a b} ⟹ x ∈ I for x
    using interval-def assms(1) a b by auto
  have ⟦min a b ≤ x; x ≤ max a b⟧ ⟹ DERIV f x :> f' x for x
    using c assms(4) by auto
  then obtain z where z:z ≥ min a b z ≤ max a b f b − f a = (b−a) * f' z
    using MVT-symmetric by blast
  have z ∈ I
    using c z(1,2) by auto
  thus ?thesis using z(3) by auto
qed
```

Ln is monotone on the positive numbers and thus commutes with min and max:

```
lemma ln-min-swap:
```

$x > (0::real) \Longrightarrow (y > 0) \Longrightarrow ln\ (min\ x\ y) = min\ (ln\ x)\ (ln\ y)$
using *ln-less-cancel-iff* **by** *fastforce*

lemma *ln-max-swap*:
$x > (0::real) \Longrightarrow (y > 0) \Longrightarrow ln\ (max\ x\ y) = max\ (ln\ x)\ (ln\ y)$
using *ln-le-cancel-iff* **by** *fastforce*

Loose lower bounds for the factorial fuction:.

lemma *fact-lower-bound*:
$sqrt(2*pi*n)*(n/exp(1))\hat{}n \leq fact\ n$ (**is** $?L \leq ?R$)
proof (*cases* $n > 0$)
case *True*
have $ln\ ?L = ln\ (2*pi*n)/2 + n * ln\ n - n$
using *True* **by** (*simp add*: *ln-mult ln-sqrt ln-realpow ln-div algebra-simps*)
also have ... $\leq ln\ ?R$
by (*intro Stirling-Formula.ln-fact-bounds True*)
finally show *?thesis*
using *iffD1*[*OF ln-le-cancel-iff*] *True* **by** *simp*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *fact-lower-bound-1*:
assumes $n > 0$
shows $(n/exp\ 1)\hat{}n \leq fact\ n$ (**is** $?L \leq ?R$)
proof −
have $2 * pi \geq 1$ **using** *pi-ge-two* **by** *auto*
moreover have $n \geq 1$ **using** *assms* **by** *simp*
ultimately have $2 * pi * n \geq 1*1$
by (*intro mult-mono*) *auto*
hence *a*:$2 * pi * n \geq 1$ **by** *simp*

have $?L = 1 * ?L$ **by** *simp*
also have ... $\leq sqrt(2 * pi * n) * ?L$
using *a* **by** (*intro mult-right-mono*) *auto*
also have ... $\leq ?R$
using *fact-lower-bound* **by** *simp*
finally show *?thesis* **by** *simp*
qed

Rules to handle O-notation with multiple variables, where some filters may be towards zero:

lemma *real-inv-at-right-0-inf*:
$\forall_F\ x\ in\ at\text{-}right\ (0::real).\ c \leq 1\ /\ x$
proof −
have $c \leq 1\ /\ \ x$ **if** *b*: $x \in \{0<..<1\ /\ (max\ c\ 1)\}$ **for** x
proof −
have $c * x \leq (max\ c\ 1) * x$
using *b* **by** (*intro mult-right-mono*, *linarith*, *auto*)
also have ... $\leq (max\ c\ 1)\ \ * (1\ /\ (max\ c\ 1))$
using *b* **by** (*intro mult-left-mono*) *auto*
also have ... ≤ 1
by (*simp add*:*of-rat-divide*)
finally have $c * x \leq 1$ **by** *simp*
moreover have $0 < x$
using *b* **by** *simp*
ultimately show *?thesis* **by** (*subst pos-le-divide-eq*, *auto*)

```
  qed
  thus ?thesis
    by (intro eventually-at-rightI[where b=1/(max c 1)], simp-all)
qed

lemma bigo-prod-1:
  assumes (λx. f x) ∈ O[F](λx. g x) G ≠ bot
  shows (λx. f (fst x)) ∈ O[F ×_F G](λx. g (fst x))
proof −
  obtain c where a: ∀_F x in F. norm (f x) ≤ c * norm (g x) and c-gt-0: c > 0
    using assms unfolding bigo-def by auto

  have ∃ c>0. ∀_F x in F ×_F G. norm (f (fst x)) ≤ c * norm (g (fst x))
    by (intro exI[where x=c] conjI c-gt-0  eventually-prod1' a assms(2))
  thus ?thesis
    unfolding bigo-def by simp
qed

lemma bigo-prod-2:
  assumes (λx. f x) ∈ O[G](λx. g x) F ≠ bot
  shows (λx. f (snd x)) ∈ O[F ×_F G](λx. g (snd x))
proof −
  obtain c where a: ∀_F x in G. norm (f x) ≤ c * norm (g x) and c-gt-0: c > 0
    using assms unfolding bigo-def by auto

  have ∃ c>0. ∀_F x in F ×_F G. norm (f (snd x)) ≤ c * norm (g (snd x))
    by (intro exI[where x=c] conjI c-gt-0  eventually-prod2' a assms(2))
  thus ?thesis
    unfolding bigo-def by simp
qed

lemma eventually-inv:
  fixes P :: real ⇒ bool
  assumes eventually (λx. P (1/x)) at-top
  shows eventually (λx. P x) (at-right 0)
proof −
  obtain N where c:n ≥ N ⟹ P (1/n) for n
    using assms unfolding eventually-at-top-linorder by auto

  define q where q = max 1 N
  have d: 0 < 1 / q q > 0
    unfolding q-def by auto

  have P x if x ∈ {0<..<1 / q} for x
  proof −
    define n where n = 1/x
    have x-eq: x = 1 / n
      unfolding n-def using that by simp

    have N ≤ q unfolding q-def by simp
    also have ... ≤ n
      unfolding n-def using that d by (simp add:divide-simps ac-simps)
    finally have N ≤ n by simp
    thus ?thesis
      unfolding x-eq by (intro c)
  qed

  thus ?thesis
```

by (*intro eventually-at-rightI*[**where** *b*=*1*/*q*] *d*)
qed

lemma *bigo-inv*:
fixes *f g* :: *real* ⇒ *real*
assumes (λ*x*. *f* (*1*/*x*)) ∈ *O*(λ*x*. *g* (*1*/*x*))
shows *f* ∈ *O*[*at-right 0*](*g*)
using *assms eventually-inv* **unfolding** *bigo-def* **by** *auto*

unbundle *no intro-cong-syntax*

3 Blind

Blind section added to preserve section numbers

end

4 Balls and Bins

The balls and bins model describes the probability space of throwing r balls into b bins. This section derives the expected number of bins hit by at least one ball, as well as the variance in the case that each ball is thrown independently. Further, using an approximation argument it is then possible to derive bounds for the same measures in the case when the balls are being thrown only k-wise independently. The proofs follow the reasoning described in [7, §A.1] but improve on the constants, as well as constraints.

theory *Distributed-Distinct-Elements-Balls-and-Bins*
imports
Distributed-Distinct-Elements-Preliminary
Discrete-Summation.Factorials
HOL−Combinatorics.Stirling
HOL−Computational-Algebra.Polynomial
HOL−Decision-Procs.Approximation
begin

hide-fact *Henstock-Kurzweil-Integration.integral-sum*
hide-fact *Henstock-Kurzweil-Integration.integral-mult-right*
hide-fact *Henstock-Kurzweil-Integration.integral-nonneg*
hide-fact *Henstock-Kurzweil-Integration.integral-cong*
unbundle *intro-cong-syntax*

lemma *sum-power-distrib*:
fixes *f* :: *'a* ⇒ *real*
assumes *finite R*
shows (∑ *i*∈*R*. *f i*) ^ *s* = (∑ *xs* | *set xs* ⊆ *R* ∧ *length xs* = *s*. (∏ *x* ← *xs*. *f x*))
proof (*induction s*)
case *0*
have {*xs*. *xs* = [] ∧ *set xs* ⊆ *R*} = {[]}
by (*auto simp add*:*set-eq-iff*)
then show *?case* **by** *simp*
next
case (*Suc s*)
have *a*:
(⋃ *i*∈*R*. (#) *i* ‘ {*xs*. *set xs* ⊆ *R* ∧ *length xs* = *s*}) = {*xs*. *set xs* ⊆ *R* ∧ *length xs* = *Suc s*}
by (*subst lists-length-Suc-eq*) *auto*
have *sum f R* ^ *Suc s* = (*sum f R*) ∗ (*sum f R*) ^ *s*
by *simp*

also have ... = $(sum\ f\ R) * (\sum xs \mid set\ xs \subseteq R \wedge length\ xs = s.\ (\prod x \leftarrow xs.\ f\ x))$
using *Suc* **by** *simp*
also have ... = $(\sum i \in R.\ (\sum xs \mid set\ xs \subseteq R \wedge length\ xs = s.\ (\prod x \leftarrow i\#xs.\ f\ x)))$
by (*subst sum-product*) *simp*
also have ... =
$(\sum i \in R.\ (\sum xs \in (\lambda xs.\ i\#xs)\ `\ \{xs.\ set\ xs \subseteq R \wedge length\ xs = s\}.\ (\prod x \leftarrow xs.\ f\ x)))$
by (*subst sum.reindex*) (*auto*)
also have ... = $(\sum xs \in (\bigcup i \in R.\ (\#)\ i\ `\ \{xs.\ set\ xs \subseteq R \wedge length\ xs = s\}).\ (\prod x \leftarrow xs.\ f\ x))$
by (*intro sum.UNION-disjoint*[*symmetric*] *assms ballI finite-imageI finite-lists-length-eq*)
auto
also have ... = $(\sum xs \mid set\ xs \subseteq R \wedge length\ xs = Suc\ s.\ (\prod x \leftarrow xs.\ f\ x))$
by (*intro sum.cong a*) *auto*
finally show *?case* **by** *simp*
qed

lemma *sum-telescope-eq*:
fixes $f :: nat \Rightarrow 'a :: \{comm\text{-}ring\text{-}1\}$
shows $(\sum k \in \{Suc\ m..n\}.\ f\ k - f\ (k - 1)) = of\text{-}bool(m \leq n) * (f\ n - f\ m)$
by (*cases* $m \leq n$, *subst sum-telescope''*, *auto*)

An improved version of *diff-power-eq-sum*.

lemma *power-diff-sum*:
fixes $a\ b :: 'a :: \{comm\text{-}ring\text{-}1,power\}$
shows $a\hat{}k - b\hat{}k = (a-b) * (\sum i = 0..<k.\ a\ \hat{}\ i * b\ \hat{}\ (k - 1 - i))$
proof (*cases k*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc nat*)
then show *?thesis*
unfolding *Suc diff-power-eq-sum*
using *atLeast0LessThan diff-Suc-1* **by** *presburger*
qed

lemma *power-diff-est*:
assumes $(a :: real) \geq b$
assumes $b \geq 0$
shows $a\hat{}k - b\hat{}k \leq (a-b) * k * a\hat{}(k-1)$
proof −
have $a\hat{}k - b\hat{}k = (a-b) * (\sum i = 0..<k.\ a\ \hat{}\ i * b\ \hat{}\ (k - 1 - i))$
by (*rule power-diff-sum*)
also have ... $\leq (a-b) * (\sum i = 0..<k.\ a\hat{}i * a\hat{}(k-1-i))$
using *assms* **by** (*intro mult-left-mono sum-mono mult-right-mono power-mono*, *auto*)
also have ... $= (a-b) * (k * a\hat{}(k-1))$
by (*simp add:power-add*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *power-diff-est-2*:
assumes $(a :: real) \geq b$
assumes $b \geq 0$
shows $a\hat{}k - b\hat{}k \geq (a-b) * k * b\hat{}(k-1)$
proof −
have $(a-b) * k * b\hat{}(k-1) = (a-b) * (\sum i=0..<k.\ b\hat{}i * b\hat{}(k-1-i))$
by (*simp add:power-add*[*symmetric*])
also have ... $\leq\ (a-b) * (\sum i=0..<k.\ a\hat{}i * b\hat{}(k-1-i))$
using *assms*
by (*intro mult-left-mono sum-mono mult-right-mono power-mono*) *auto*

```
  also have ... = a^k - b^k
    by (rule power-diff-sum[symmetric])
  finally show ?thesis by simp
qed

lemma of-bool-prod:
  assumes finite R
  shows (∏ j ∈ R. of-bool(f j)) = (of-bool(∀ j ∈ R. f j) :: real)
  using assms by (induction R rule:finite-induct) auto
```

Additional results about falling factorials:

```
lemma ffact-nonneg:
  fixes x :: real
  assumes k - 1 ≤ x
  shows ffact k x ≥ 0
  using assms unfolding prod-ffact[symmetric]
  by (intro prod-nonneg ballI) simp

lemma ffact-pos:
  fixes x :: real
  assumes k - 1 < x
  shows ffact k x > 0
  using assms unfolding prod-ffact[symmetric]
  by (intro prod-pos ballI) simp

lemma ffact-mono:
  fixes x y :: real
  assumes k-1 ≤ x x ≤ y
  shows ffact k x ≤ ffact k y
  using assms
  unfolding prod-ffact[symmetric]
  by (intro prod-mono) auto

lemma ffact-of-nat-nonneg:
  fixes x :: 'a :: {comm-ring-1, linordered-nonzero-semiring}
  assumes x ∈ ℕ
  shows ffact k x ≥ 0
proof -
  obtain y where y-def: x = of-nat y
    using assms(1) Nats-cases by auto
  have (0::'a) ≤ of-nat (ffact k y)
    by simp
  also have ... = ffact k x
    by (simp add:of-nat-ffact y-def)
  finally show ?thesis by simp
qed

lemma ffact-suc-diff:
  fixes x :: ('a :: comm-ring-1)
  shows ffact k x - ffact k (x-1) = of-nat k * ffact (k-1) (x-1) (is ?L = ?R)
proof (cases k)
  case 0
  then show ?thesis by simp
next
  case (Suc n)
  hence ?L = ffact (Suc n) x - ffact (Suc n) (x-1) by simp
  also have ... =  x * ffact n (x-1) - ((x-1)-of-nat n) * ffact n (x-1)
    by (subst (1) ffact-Suc, simp add: ffact-Suc-rev)
```

```
  also have ... = of-nat (Suc n)  * ffact n (x−1)
    by (simp add:algebra-simps)
  also have ... = of-nat k * ffact (k−1) (x−1) using Suc by simp
  finally show ?thesis by simp
qed

lemma ffact-bound:
  ffact k (n::nat) ≤ n^k
proof −
  have ffact k n = (∏ i=0..<k. (n−i))
    unfolding prod-ffact-nat[symmetric]
    by simp
  also have ... ≤ (∏ i=0..<k. n)
    by (intro prod-mono) auto
  also have ... = n^k
    by simp
  finally show ?thesis by simp
qed

lemma fact-moment-binomial:
  fixes n :: nat and α :: real
  assumes α ∈ {0..1}
  defines p ≡ binomial-pmf n α
  shows (∫ ω. ffact s (real ω) ∂p) = ffact s (real n) * α^s (is ?L = ?R)
proof (cases s ≤ n)
  case True
  have ?L =  (∑ k≤n. (real (n choose k) * α ^ k * (1 − α) ^ (n − k)) * real (ffact s k))
   unfolding p-def using assms by (subst expectation-binomial-pmf') (auto simp add:of-nat-ffact)
  also have ... = (∑ k ∈ {0+s..(n−s)+s}. (real (n choose k) * α ^ k * (1 − α) ^ (n − k)) *
ffact s k)
    using True ffact-nat-triv by (intro sum.mono-neutral-cong-right) auto
  also have ... = (∑ k=0..n−s. α^s * real (n choose (k+s)) * α^k * (1−α)^(n−(k+s)) *ffact s
(k+s))
    by (subst sum.atLeastAtMost-shift-bounds, simp add:algebra-simps power-add)
  also have ... = α^s * (∑ k≤n−s. real (n choose (k+s))*ffact s (k+s)*α^k*(1−α)^((n−s)−k))
    using atMost-atLeast0 by (simp add: sum-distrib-left algebra-simps cong:sum.cong)
 also have ... = α^s * (∑ k≤n−s. real (n choose (k+s))*fact (k+s) / fact k * α^k*(1−α)^((n−s)−k))
    using real-of-nat-div[OF fact-dvd[OF le-add1]]
    by (subst fact-div-fact-ffact-nat[symmetric], auto)
  also have ... = α^s * (∑ k≤n−s.
    (fact n / fact (n−s)) * fact (n−s) / (fact ((n−s)−k) * fact k) * α^k*(1−α)^((n−s)−k))
    using True by (intro arg-cong2[where f=(*)] sum.cong)
     (auto simp add: binomial-fact algebra-simps)
  also have ... = α^s * (fact n / fact (n − s)) *
    (∑ k≤n−s. fact (n−s) / (fact ((n−s)−k) * fact k) * α^k*(1−α)^((n−s)−k))
    by (simp add:sum-distrib-left algebra-simps)
 also have ... = α^s * (fact n / fact (n − s)) * (∑ k≤n−s. ((n−s) choose k) * α^k*(1−α)^((n−s)−k))
    using True by (intro-cong [σ2(*)] more: sum.cong) (auto simp add: binomial-fact)
  also have ... = α^s * real (fact n div fact (n − s)) * (α+(1−α))^(n−s)
    using True real-of-nat-div[OF fact-dvd] by (subst binomial-ring, simp)
  also have ... = α^s * real (ffact s n)
    by (subst fact-div-fact-ffact-nat[OF True], simp)
  also have ... = ?R
    by (subst of-nat-ffact, simp)
  finally show ?thesis by simp
next
  case False
  have ?L =  (∑ k≤n. (real (n choose k) * α ^ k * (1 − α) ^ (n − k)) * real (ffact s k))
```

```
  unfolding p-def using assms by (subst expectation-binomial-pmf') (auto simp add:of-nat-ffact)
 also have ... = (∑ k≤n. (real (n choose k) * α ^ k * (1 - α) ^ (n - k)) * real 0)
   using False
   by (intro-cong [σ2(*),σ1 of-nat] more: sum.cong ffact-nat-triv) auto
 also have ... = 0 by simp
 also have ... = real (ffact s n) * α^s
   using False by (subst ffact-nat-triv, auto)
 also have ... = ?R
   by (subst of-nat-ffact, simp)
 finally show ?thesis by simp
qed
```

The following describes polynomials of a given maximal degree as a subset of the functions, similar to the subsets $\mathbb{Z}$ or $\mathbb{Q}$ as subsets of larger number classes.

```
definition Polynomials (‹ℙ›)
 where Polynomials k = {f. ∃p. f = poly p ∧ degree p ≤ k}

lemma Polynomials-mono:
 assumes s ≤ t
 shows ℙ s ⊆ ℙ t
 using assms unfolding Polynomials-def by auto

lemma Polynomials-addI:
 assumes f ∈ ℙ k g ∈ ℙ k
 shows (λω. f ω + g ω) ∈ ℙ k
proof −
 obtain pf pg where fg-def: f = poly pf degree pf ≤ k g = poly pg degree pg ≤ k
   using assms unfolding Polynomials-def by blast
 hence degree (pf + pg) ≤ k (λx. f x + g x) = poly (pf + pg)
   using degree-add-le by auto
 thus ?thesis unfolding Polynomials-def by auto
qed

lemma Polynomials-diffI:
 fixes f g :: 'a :: comm-ring ⇒ 'a
 assumes f ∈ ℙ k g ∈ ℙ k
 shows (λx. f x − g x) ∈ ℙ k
proof −
 obtain pf pg where fg-def: f = poly pf degree pf ≤ k g = poly pg degree pg ≤ k
   using assms unfolding Polynomials-def by blast
 hence degree (pf − pg) ≤ k (λx. f x − g x) = poly (pf − pg)
   using degree-diff-le by auto
 thus ?thesis unfolding Polynomials-def by auto
qed

lemma Polynomials-idI:
 (λx. x) ∈ (ℙ 1 :: ('a::comm-ring-1 ⇒ 'a) set)
proof −
 have (λx. x) = poly [: 0,(1::'a) :]
   by (intro ext, auto)
 also have ... ∈ ℙ 1
   unfolding Polynomials-def by auto
 finally show ?thesis by simp
qed

lemma Polynomials-constI:
 (λx. c) ∈ ℙ k
proof −
```

```
  have (λx. c) = poly [: c :]
    by (intro ext, simp)
  also have ... ∈ ℙ k
    unfolding Polynomials-def by auto
  finally show ?thesis by simp
qed

lemma Polynomials-multI:
  fixes f g :: 'a :: {comm-ring} ⇒ 'a
  assumes f ∈ ℙ s g ∈ ℙ t
  shows (λx. f x * g x) ∈ ℙ (s+t)
proof −
  obtain pf pg where xy-def: f = poly pf degree pf ≤ s g = poly pg degree pg ≤ t
    using assms unfolding Polynomials-def by blast

  have degree (pf * pg) ≤ degree pf + degree pg
    by (intro degree-mult-le)
  also have ... ≤ s + t
    using xy-def by (intro add-mono) auto
  finally have degree (pf * pg) ≤ s+t by simp
  moreover have (λx. f x * g x) = poly (pf * pg)
    using xy-def by auto
  ultimately show ?thesis unfolding Polynomials-def by auto
qed

lemma Polynomials-composeI:
  fixes f g :: 'a :: {comm-semiring-0, semiring-no-zero-divisors} ⇒ 'a
  assumes f ∈ ℙ s g ∈ ℙ t
  shows (λx. f (g x)) ∈ ℙ (s*t)
proof −
  obtain pf pg where xy-def: f = poly pf degree pf ≤ s g = poly pg degree pg ≤ t
    using assms unfolding Polynomials-def by blast
  have degree (pf ∘p pg) = degree pf * degree pg
    by (intro degree-pcompose)
  also have ... ≤ s * t
    using xy-def by (intro mult-mono) auto
  finally have degree (pf ∘p pg) ≤ s * t
    by simp
  moreover have (λx. f (g x)) = poly (pf ∘p pg)
    unfolding xy-def
    by (intro ext poly-pcompose[symmetric])
  ultimately show ?thesis unfolding Polynomials-def by auto
qed

lemma Polynomials-const-left-multI:
  fixes c :: 'a :: {comm-ring}
  assumes f ∈ ℙ k
  shows (λx. c * f x) ∈ ℙ k
proof −
  have (λx. c * f x) ∈ ℙ (0+k)
    by (intro Polynomials-multI Polynomials-constI assms)
  thus ?thesis by simp
qed

lemma Polynomials-const-right-multI:
  fixes c :: 'a :: {comm-ring}
  assumes f ∈ ℙ k
  shows (λx. f x * c) ∈ ℙ k
```

proof −
  have $(\lambda x.\ f\ x * c) \in \mathbb{P}\ (k+0)$
    by (*intro Polynomials-multI Polynomials-constI assms*)
  thus *?thesis* **by** *simp*
qed

lemma *Polynomials-const-divI*:
  fixes $c :: 'a :: \{field\}$
  assumes $f \in \mathbb{P}\ k$
  shows $(\lambda x.\ f\ x\ /\ c) \in \mathbb{P}\ k$
proof −
  have $(\lambda x.\ f\ x * (1/c)) \in \mathbb{P}\ (k+0)$
    by (*intro Polynomials-multI Polynomials-constI assms*)
  thus *?thesis* **by** *simp*
qed

lemma *Polynomials-ffact*: $(\lambda x.\ ffact\ s\ (x - y)) \in (\mathbb{P}\ s :: ('a :: comm\text{-}ring\text{-}1 \Rightarrow 'a)\ set)$
proof (*induction s arbitrary*: *y*)
  case *0*
  then show *?case*
    using *Polynomials-constI*[**where** *c=1*] **by** *simp*
next
  case (*Suc s*)
  have $(\lambda(x :: 'a).\ ffact\ (Suc\ s)\ (x-y)) = (\lambda x.\ (x-y) * ffact\ s\ (x - (y+1)))$
    by (*simp add*: *ffact-Suc algebra-simps*)
  also have ... $\in \mathbb{P}\ (1+s)$
    by (*intro Polynomials-multI Suc Polynomials-diffI Polynomials-idI Polynomials-constI*)
  finally show *?case* **by** *simp*
qed

lemmas *Polynomials-intros* =
  Polynomials-const-divI
  Polynomials-composeI
  Polynomials-const-left-multI
  Polynomials-const-right-multI
  Polynomials-multI
  Polynomials-addI
  Polynomials-diffI
  Polynomials-idI
  Polynomials-constI
  Polynomials-ffact

definition $C_2 :: real$ **where** $C_2 = 7.5$
definition $C_3 :: real$ **where** $C_3 = 16$

A locale fixing the sets of balls and bins

locale *balls-and-bins-abs* =
  fixes $R :: 'a\ set$ **and** $B :: 'b\ set$
  assumes *fin-B*: *finite B* **and** *B-ne*: $B \neq \{\}$
  assumes *fin-R*: *finite R*
begin

Independent balls and bins space:

definition Ω
  where Ω = *prod-pmf R* (λ-. *pmf-of-set B*)

lemma *set-pmf-Ω*: *set-pmf* $\Omega = R \rightarrow_E B$
  unfolding Ω*-def set-prod-pmf*[*OF fin-R*]

by (*simp add*:*comp-def set-pmf-of-set*[*OF B-ne fin-B*])

lemma *card-B-gt-0*: *card B* > *0*
using *B-ne fin-B* **by** *auto*

lemma *card-B-ge-1*: *card B* ≥ *1*
using *card-B-gt-0* **by** *simp*

definition *Z j ω* = *real* (*card* {*i*. *i* ∈ *R* ∧ *ω i* = (*j*::'*b*)})
definition *Y ω* = *real* (*card* (*ω* ' *R*))
definition *μ* = *real* (*card B*) ∗ (*1* − (*1*−*1*/*real* (*card B*))^*card R*)

Factorial moments for the random variable describing the number of times a bin will be hit:

lemma *fact-moment-balls-and-bins*:
assumes *J* ⊆ *B J* ≠ {}
shows (∫ *ω*. *ffact s* (∑ *j* ∈ *J*. *Z j ω*) ∂Ω) =
ffact s (*real* (*card R*)) ∗ (*real* (*card J*) / *real* (*card B*))^*s*
(**is** *?L* = *?R*)
proof −
let *?α* = *real* (*card J*) / *real* (*card B*)
let *?q* = *binomial-pmf* (*card R*) *?α*
let *?Y* = (λ*ω*. *card* {*r* ∈ *R*. *ω r* ∈ *J*})

have *fin-J*: *finite J*
using *finite-subset assms*(*1*) *fin-B* **by** *auto*

have *Z-sum-eq*: (∑ *j* ∈ *J*. *Z j ω*) = *real* (*?Y ω*) **for** *ω*
proof −
have *?Y ω* = *card* (⋃ *j* ∈ *J*. {*r* ∈ *R*. *ω r*= *j*})
by (*intro arg-cong*[**where** *f*=*card*]) *auto*
also have ... = (∑ *i*∈*J*. *card* {*r* ∈ *R*. *ω r* = *i*})
using *fin-R fin-J* **by** (*intro card-UN-disjoint*) *auto*
finally have *?Y ω* = (∑ *j* ∈ *J*. *card* {*r* ∈ *R*. *ω r* = *j*}) **by** *simp*
thus *?thesis*
unfolding *Z-def of-nat-sum*[*symmetric*] **by** *simp*
qed

have *card-J*: *card J* ≤ *card B*
using *assms*(*1*) *fin-B card-mono* **by** *auto*
have *α-range*: *?α* ≥ *0* *?α* ≤ *1*
using *card-J card-B-gt-0* **by** *auto*

have *pmf* (*map-pmf* (λ*ω*. *ω* ∈ *J*) (*pmf-of-set B*)) *x* = *pmf* (*bernoulli-pmf ?α*) *x*
(**is** *?L1*=*?R1*) **for** *x*
proof −
have *?L1* = *real* (*card* (*B* ∩ {*ω*. (*ω* ∈ *J*) = *x*})) / *real* (*card B*)
using *B-ne fin-B*
by (*simp add*:*pmf-map measure-pmf-of-set vimage-def*)
also have ... = (*if x then* (*card J*) *else* (*card* (*B* − *J*))) / *real* (*card B*)
using *Int-absorb1*[*OF assms*(*1*)] **by** (*auto simp add*:*Diff-eq Int-def*)
also have ... = (*if x then* (*card J*) / *card B else* (*real* (*card B*) − *card J*) / *real* (*card B*))
using *card-J fin-J assms*(*1*) **by** (*simp add*: *of-nat-diff card-Diff-subset*)
also have ... = (*if x then ?α else* (*1* − *?α*))
using *card-B-gt-0* **by** (*simp add*:*divide-simps*)
also have ... = *?R1*
using *α-range* **by** *auto*
finally show *?thesis* **by** *simp*

```
  qed
  hence c:map-pmf (λω. ω ∈ J) (pmf-of-set B) = bernoulli-pmf ?α
    by (intro pmf-eqI) simp

  have map-pmf (λω. λr ∈ R. ω r ∈ J) Ω = prod-pmf R (λ-. (map-pmf (λω. ω ∈ J) (pmf-of-set
B)))
    unfolding map-pmf-def Ω-def restrict-def using fin-R
    by (subst Pi-pmf-bind[where d'=undefined]) auto
  also have ... = prod-pmf R (λ-. bernoulli-pmf ?α)
    unfolding c by simp
  finally have b:map-pmf (λω. λr ∈ R. ω r ∈ J) Ω  =  prod-pmf R (λ-. bernoulli-pmf ?α)
    by simp

  have map-pmf ?Y Ω = map-pmf ((λω. card {r ∈ R. ω r}) ∘ (λω. λr∈R. ω r ∈ J)) Ω
    unfolding comp-def
    by (intro map-pmf-cong arg-cong[where f=card]) (auto simp add:comp-def)
  also have ... = (map-pmf (λω. card {r ∈ R. ω r}) ∘ map-pmf (λω. λr ∈ R. ω r ∈ J)) Ω
    by (subst map-pmf-compose[symmetric]) auto
  also have ... = map-pmf (λω. card {r ∈ R. ω r}) (prod-pmf R (λ-. (bernoulli-pmf ?α)))
    unfolding comp-def b by simp
  also have ... = ?q
    using α-range by (intro binomial-pmf-altdef'[symmetric] fin-R) auto
  finally have a:map-pmf ?Y Ω =?q
    by simp

  have ?L = (∫ ω. ffact s (real (?Y ω)) ∂Ω)
    unfolding Z-sum-eq by simp
  also have ... = (∫ ω. ffact s (real ω) ∂(map-pmf ?Y Ω))
    by simp
  also have ... = (∫ ω. ffact s (real ω) ∂?q)
    unfolding a by simp
  also have ... = ?R
    using α-range by (subst fact-moment-binomial, auto)
  finally show ?thesis by simp
qed
```

Expectation and variance for the number of distinct bins that are hit by at least one ball in the fully independent model. The result for the variance is improved by a factor of 4 w.r.t. the paper.

```
lemma
  shows exp-balls-and-bins: measure-pmf.expectation Ω Y = μ (is ?AL = ?AR)
    and var-balls-and-bins: measure-pmf.variance Ω Y ≤ card R * (real (card R) − 1) / card B
      (is ?BL ≤ ?BR)
proof −
  let ?b = real (card B)
  let ?r = card R
  define Z :: 'b ⇒ ('a ⇒ 'b) ⇒ real
    where Z = (λi ω. of-bool(i ∉ ω ‘ R))
  define α where α = (1 − 1 / ?b)^?r
  define β where β = (1 − 2 / ?b)^?r

  have card (B × B ∩ {x. fst x = snd x}) = card ((λx. (x,x)) ‘ B)
    by (intro arg-cong[where f=card]) auto
  also have ... = card B
    by (intro card-image, simp add:inj-on-def)
  finally have d: card (B × B ∩ {x. fst x = snd x}) = card B
    by simp
  hence count-1: real (card (B × B ∩ {x. fst x = snd x})) = card B
```

```
  by simp

have card B + card (B × B ∩ −{x. fst x = snd x}) =
  card (B × B ∩ {x. fst x = snd x}) + card (B × B ∩ −{x. fst x = snd x})
  by (subst d) simp
also have ... = card ((B × B ∩ {x. fst x = snd x}) ∪ (B × B ∩ −{x. fst x = snd x}))
  using finite-subset[OF - finite-cartesian-product[OF fin-B fin-B]]
  by (intro card-Un-disjoint[symmetric]) auto
also have ... = card (B × B)
  by (intro arg-cong[where f=card]) auto
also have ... = card B^2
  unfolding card-cartesian-product by (simp add:power2-eq-square)
finally have card B + card (B × B ∩ −{x. fst x = snd x}) = card B^2 by simp

hence count-2: real (card (B × B ∩ −{x. fst x = snd x})) = real (card B)^2 − card B
  by (simp add:algebra-simps flip: of-nat-add of-nat-power)

hence finite (set-pmf Ω)
  unfolding set-pmf-Ω
  using fin-R fin-B by (auto intro!:finite-PiE)
hence int: integrable (measure-pmf Ω) f
  for f :: ('a ⇒ 'b) ⇒ real
  by (intro integrable-measure-pmf-finite) simp

have a:prob-space.indep-vars (measure-pmf Ω) (λi. discrete) (λx ω. ω x) R
  unfolding Ω-def using indep-vars-Pi-pmf[OF fin-R] by metis
have b: (∫ ω. of-bool (ω ' R ⊆ A) ∂Ω) = (real (card (B ∩ A)) / real (card B))^card R
  (is ?L = ?R) for A
proof −
  have ?L = (∫ ω. (∏ j ∈ R. of-bool(ω j ∈ A)) ∂Ω)
    by (intro Bochner-Integration.integral-cong ext)
      (auto simp add: of-bool-prod[OF fin-R])
  also have ... = (∏ j ∈ R. (∫ ω. of-bool(ω j ∈ A) ∂Ω))
    using fin-R
    by (intro prob-space.indep-vars-lebesgue-integral[OF prob-space-measure-pmf] int
        prob-space.indep-vars-compose2[OF prob-space-measure-pmf a]) auto
  also have ... = (∏ j ∈ R. (∫ ω. of-bool(ω ∈ A) ∂(map-pmf (λω. ω j) Ω)))
    by simp
  also have ... = (∏ j ∈ R. (∫ ω. of-bool(ω ∈ A) ∂(pmf-of-set B)))
    unfolding Ω-def by (subst Pi-pmf-component[OF fin-R]) simp
  also have ... = ((∑ ω∈B. of-bool (ω ∈ A)) / real (card B)) ^ card R
    by (simp add: integral-pmf-of-set[OF B-ne fin-B])
  also have ... = ?R
    unfolding of-bool-def sum.If-cases[OF fin-B] by simp
  finally show ?thesis by simp
qed

have Z-exp: (∫ ω. Z i ω ∂Ω) = α if i ∈ B for i
proof −
  have real (card (B ∩ −{i})) = real (card (B − {i}))
    by (intro-cong [σ1 card,σ1 of-nat]) auto
  also have ... = real (card B − card {i})
    using that by (subst card-Diff-subset) auto
  also have ... = real (card B) − real (card {i})
    using fin-B that by (intro of-nat-diff card-mono) auto
  finally have c: real (card (B ∩ −{i})) = real (card B) − 1
    by simp
```

```
  have (∫ ω. Z i ω ∂Ω) = (∫ ω. of-bool(ω ‘ R ⊆ - {i}) ∂Ω)
    unfolding Z-def by simp
  also have ... = (real (card (B ∩ -{i})) / real (card B))^card R
    by (intro b)
  also have ... = ((real (card B) -1) / real (card B))^card R
    by (subst c) simp
  also have ... = α
    unfolding α-def using card-B-gt-0
    by (simp add:divide-eq-eq diff-divide-distrib)
  finally show ?thesis
    by simp
qed

have Z-prod-exp: (∫ ω. Z i ω * Z j ω ∂Ω) = (if i = j then α else β)
  if i ∈ B j ∈ B for i j
proof -
  have real (card (B ∩ -{i,j})) = real (card (B - {i,j}))
    by (intro-cong [σ1 card,σ1 of-nat]) auto
  also have ... = real (card B - card {i,j})
    using that by (subst card-Diff-subset) auto
  also have ... = real (card B) - real (card {i,j})
    using fin-B that by (intro of-nat-diff card-mono) auto
  finally have c: real (card (B ∩ -{i,j})) = real (card B) - card {i,j}
    by simp

  have (∫ ω. Z i ω * Z j ω ∂Ω) = (∫ ω. of-bool(ω ‘ R ⊆ - {i,j}) ∂Ω)
    unfolding Z-def of-bool-conj[symmetric]
    by (intro integral-cong ext) auto
  also have ... = (real (card (B ∩ -{i,j})) / real (card B))^card R
    by (intro b)
  also have ... = ((real (card B) - card {i,j}) / real (card B))^card R
    by (subst c) simp
  also have ... = (if i = j then α else β)
    unfolding α-def β-def using card-B-gt-0
    by (simp add:divide-eq-eq diff-divide-distrib)
  finally show ?thesis by simp
qed

have Y-eq: Y ω = (∑ i ∈ B. 1 - Z i ω) if ω ∈ set-pmf Ω for ω
proof -
  have set-pmf Ω ⊆ Pi R (λ-. B)
    using set-pmf-Ω by (simp add:PiE-def)
  hence ω ‘ R ⊆ B
    using that by auto
  hence Y ω = card (B ∩ ω ‘ R)
    unfolding Y-def using Int-absorb1 by metis
  also have ... = (∑ i ∈ B. of-bool(i ∈ ω ‘ R))
    unfolding of-bool-def sum.If-cases[OF fin-B] by(simp)
  also have ... = (∑ i ∈ B. 1 - Z i ω)
    unfolding Z-def by (intro sum.cong) (auto simp add:of-bool-def)
  finally show Y ω = (∑ i ∈ B. 1 - Z i ω) by simp
qed

have Y-sq-eq: (Y ω)^2 = (∑ (i,j) ∈ B × B. 1 - Z i ω - Z j ω + Z i ω * Z j ω)
  if ω ∈ set-pmf Ω for ω
  unfolding Y-eq[OF that] power2-eq-square sum-product sum.cartesian-product
  by (intro sum.cong) (auto simp add:algebra-simps)
```

have *measure-pmf.expectation* Ω *Y* = (∫ ω. (∑ *i* ∈ *B*. *1* − *Z i* ω) ∂Ω)
using *Y-eq* **by** (*intro integral-cong-AE AE-pmfI*) *auto*
also have ... = (∑ *i* ∈ *B*. *1* − (∫ ω. *Z i* ω ∂Ω))
using *int* **by** *simp*
also have ... = *?b* ∗ (*1* − α)
using *Z-exp* **by** *simp*
also have ... = *?AR*
unfolding α*-def* μ*-def* **by** *simp*
finally show *?AL* = *?AR* **by** *simp*

have *measure-pmf.variance* Ω *Y* = (∫ ω. *Y* ω^2 ∂Ω) − (∫ ω. *Y* ω ∂Ω)^2
using *int* **by** (*subst measure-pmf.variance-eq*) *auto*
also have ... =
(∫ ω. (∑ *i* ∈ *B* × *B*. *1* − *Z* (*fst i*) ω − *Z* (*snd i*) ω + *Z* (*fst i*) ω ∗ *Z* (*snd i*) ω) ∂Ω) −
(∫ ω. (∑ *i* ∈ *B*. *1* − *Z i* ω) ∂Ω)^2
using *Y-eq Y-sq-eq*
by (*intro-cong* [σ2(−),σ2 *power*] *more*: *integral-cong-AE AE-pmfI*) (*auto simp add*:*case-prod-beta*)
also have ... =
(∑ *i* ∈ *B* × *B*. (∫ ω. (*1* − *Z* (*fst i*) ω − *Z* (*snd i*) ω + *Z* (*fst i*) ω ∗ *Z* (*snd i*) ω) ∂Ω)) −
(∑ *i* ∈ *B*. (∫ ω. (*1* − *Z i* ω) ∂Ω))^2
by (*intro-cong* [σ2 (−), σ2 *power*] *more*: *integral-sum int*)
also have ... =
(∑ *i* ∈ *B* × *B*. (∫ ω. (*1* − *Z* (*fst i*) ω − *Z* (*snd i*) ω + *Z* (*fst i*) ω ∗ *Z* (*snd i*) ω) ∂Ω)) −
(∑ *i* ∈ *B* × *B*. (∫ ω. (*1* − *Z* (*fst i*) ω) ∂Ω) ∗ (∫ ω. (*1* − *Z* (*snd i*) ω) ∂Ω))
unfolding *power2-eq-square sum-product sum.cartesian-product*
by (*simp add*:*case-prod-beta*)
also have ... = (∑ (*i*,*j*) ∈ *B* × *B*. (∫ ω. (*1* − *Z i* ω − *Z j* ω + *Z i* ω ∗ *Z j* ω) ∂Ω) −
(∫ ω. (*1* − *Z i* ω) ∂Ω) ∗ (∫ ω. (*1* − *Z j* ω) ∂Ω))
by (*subst sum-subtractf*[*symmetric*], *simp add*:*case-prod-beta*)
also have ... = (∑ (*i*,*j*) ∈ *B* × *B*. (∫ ω. *Z i* ω ∗ *Z j* ω ∂Ω) −(∫ ω. *Z i* ω ∂Ω) ∗ (∫ ω. *Z j* ω ∂Ω))
using *int* **by** (*intro sum.cong refl*) (*simp add*:*algebra-simps case-prod-beta*)
also have ... = (∑ *i* ∈ *B* × *B*. (*if fst i* = *snd i then* α − α^2 *else* β − α^2))
by (*intro sum.cong refl*)
(*simp add*:*Z-exp Z-prod-exp mem-Times-iff case-prod-beta power2-eq-square*)
also have ... = *?b* ∗ (α − α²) + (*?b*^2 − *card B*) ∗ (β − α²)
using *count-1 count-2 finite-cartesian-product fin-B* **by** (*subst sum.If-cases*) *auto*
also have ... = *?b*^2 ∗ (β − α²) + *?b* ∗ (α − β)
by (*simp add*:*algebra-simps*)
also have ... = *?b* ∗ ((*1*−*1* / *?b*)^*?r* − (*1*−*2* / *?b*)^*?r*) − *?b*^2 ∗ (((*1*−*1* / *?b*)^2)^*?r* − (*1*−*2* / *?b*)^*?r*)
unfolding β*-def* α*-def*
by (*simp add*: *power-mult*[*symmetric*] *algebra-simps*)
also have ... ≤ *card R* ∗ (*real* (*card R*) − *1*)/ *card B* (**is** *?L* ≤ *?R*)
proof (*cases ?b* ≥ *2*)
case *True*
have *?L* ≤
?b ∗ (((*1* − *1* / *?b*) − (*1*−*2* / *?b*)) ∗ *?r* ∗ (*1*−*1* / *?b*)^(*?r* − *1*)) −
?b^2 ∗ ((((*1*−*1* / *?b*)^2) − ((*1* − *2* / *?b*))) ∗ *?r* ∗ ((*1*−*2* / *?b*))^(*?r* − *1*))
using *True*
by (*intro diff-mono mult-left-mono power-diff-est-2 power-diff-est divide-right-mono*)
(*auto simp add*:*power2-eq-square algebra-simps*)
also have ... = *?b* ∗ ((*1* / *?b*) ∗ *?r* ∗ (*1*−*1* / *?b*)^(*?r*−*1*)) − *?b*^2∗((*1* / *?b*^2)∗*?r*∗((*1*−*2* / *?b*))^(*?r*−*1*))
by (*intro arg-cong2*[**where** *f*=(−)] *arg-cong2*[**where** *f*=(∗)] *refl*)
(*auto simp add*:*algebra-simps power2-eq-square*)
also have ... = *?r* ∗ ((*1*−*1* / *?b*)^(*?r* − *1*) − ((*1*−*2* / *?b*))^(*?r* − *1*))
by (*simp add*:*algebra-simps*)
also have ... ≤ *?r* ∗ (((*1*−*1* / *?b*) − (*1*−*2* / *?b*)) ∗ (*?r* − *1*) ∗ (*1*−*1* / *?b*)^(*?r* −*1* −*1*))
using *True* **by** (*intro mult-left-mono power-diff-est*) (*auto simp add*:*algebra-simps*)

```
    also have ... ≤ ?r * ((1 / ?b) * (?r - 1) * 1^(?r - 1-1))
      using True by (intro mult-left-mono mult-mono power-mono) auto
    also have ... =  ?R
      using card-B-gt-0 by auto
    finally show ?L ≤ ?R by simp
  next
    case False
    hence ?b = 1 using card-B-ge-1 by simp
    thus ?L ≤ ?R
      by (cases card R =0) auto
  qed
  finally show measure-pmf.variance Ω Y ≤ card R * (real (card R) - 1)/ card B
    by simp
qed

definition lim-balls-and-bins k p = (
   prob-space.k-wise-indep-vars (measure-pmf p) k (λ-. discrete) (λx ω. ω x) R ∧
  (∀ x. x ∈ R ⟶ map-pmf (λω. ω x) p = pmf-of-set B))

lemma indep:
  assumes lim-balls-and-bins k p
  shows prob-space.k-wise-indep-vars (measure-pmf p) k (λ-. discrete) (λx ω. ω x) R
  using assms lim-balls-and-bins-def by simp

lemma ran:
  assumes lim-balls-and-bins k p x ∈ R
  shows map-pmf (λω. ω x) p = pmf-of-set B
  using assms lim-balls-and-bins-def by simp

lemma Z-integrable:
  fixes f :: real ⇒ real
  assumes lim-balls-and-bins k p
  shows integrable p (λω. f (Z i ω) )
  unfolding Z-def using fin-R card-mono
  by (intro integrable-pmf-iff-bounded[where C=Max (abs ` f ` real ` {..card R})])
   fastforce+

lemma Z-any-integrable-2:
 fixes f :: real ⇒ real
  assumes lim-balls-and-bins k p
  shows integrable p (λω. f (Z i ω + Z j ω))
proof −
  have q:real (card A) + real (card B) ∈ real ` {..2 * card R} if A ⊆ R B ⊆ R for A B
  proof −
    have card A + card B ≤ card R + card R
      by (intro add-mono card-mono fin-R that)
    also have ... = 2 * card R by simp
    finally show ?thesis by force
  qed

  thus ?thesis
    unfolding Z-def using fin-R card-mono abs-triangle-ineq
    by (intro integrable-pmf-iff-bounded[where C=Max (abs ` f ` real ` {..2*card R})] Max-ge
        finite-imageI imageI) auto
qed

lemma hit-count-prod-exp:
  assumes j1 ∈ B j2 ∈ B s+t ≤ k
```

assumes *lim-balls-and-bins k p*
defines $L \equiv \{(xs,ys).\ set\ xs \subseteq R \wedge set\ ys \subseteq R \wedge$
$(set\ xs \cap set\ ys = \{\} \vee j1 = j2) \wedge length\ xs = s \wedge length\ ys = t\}$
shows $(\int \omega.\ Z\ j1\ \omega\widehat{\ }s * Z\ j2\ \omega\widehat{\ }t\ \partial p) =$
$(\sum (xs,ys) \in L.\ (1/real\ (card\ B))\widehat{\ }(card\ (set\ xs \cup set\ ys)))$
(**is** *?L = ?R*)
proof −
define *W1* :: $'a \Rightarrow ('a \Rightarrow 'b) \Rightarrow real$
where $W1 = (\lambda i\ \omega.\ of\text{-}bool\ (\omega\ i = j1) :: real)$
define *W2* :: $'a \Rightarrow ('a \Rightarrow 'b) \Rightarrow real$
where $W2 = (\lambda i\ \omega.\ of\text{-}bool\ (\omega\ i = j2) :: real)$
define τ :: $'a\ list \times 'a\ list \Rightarrow 'a \Rightarrow 'b$
where $\tau = (\lambda l\ x.\ if\ x \in set\ (fst\ l)\ then\ j1\ else\ j2)$

have *τ-check-1*: $\tau\ l\ x = j1$ **if** $x \in set\ (fst\ l)$ **and** $l \in L$ **for** $x\ l$
using *that* **unfolding** *τ-def L-def* **by** *auto*
have *τ-check-2*: $\tau\ l\ x = j2$ **if** $x \in set\ (snd\ l)$ **and** $l \in L$ **for** $x\ l$
using *that* **unfolding** *τ-def L-def* **by** *auto*
have *τ-check-3*: $\tau\ l\ x \in B$ **for** $x\ l$
using *assms*(*1*,*2*) **unfolding** *τ-def* **by** *simp*

have *Z1-eq*: $Z\ j1\ \omega = (\sum i \in R.\ W1\ i\ \omega)$ **for** ω
using *fin-R* **unfolding** *Z-def W1-def*
by (*simp add*:*of-bool-def sum.If-cases Int-def*)

have *Z2-eq*: $Z\ j2\ \omega = (\sum i \in R.\ W2\ i\ \omega)$ **for** ω
using *fin-R* **unfolding** *Z-def W2-def*
by (*simp add*:*of-bool-def sum.If-cases Int-def*)

define α **where** $\alpha = 1\ /\ real\ (card\ B)$

have *a*: $(\int \omega.\ (\prod x \leftarrow a.\ W1\ x\ \omega) * (\prod y \leftarrow b.\ W2\ y\ \omega)\ \partial p) = 0$ (**is** *?L1 = 0*)
if $x \in set\ a \cap set\ b\ j1 \neq j2\ length\ a = s\ length\ b = t$ **for** $x\ a\ b$
proof −
have $(\prod x \leftarrow a.\ W1\ x\ \omega) * (\prod y \leftarrow b.\ W2\ y\ \omega) = 0$ **for** ω
proof −
have $W1\ x\ \omega = 0 \vee W2\ x\ \omega = 0$
unfolding *W1-def W2-def* **using** *that* **by** *simp*
hence $(\prod x \leftarrow a.\ W1\ x\ \omega) = 0 \vee (\prod y \leftarrow b.\ W2\ y\ \omega) = 0$
unfolding *prod-list-zero-iff* **using** *that*(*1*) **by** *auto*
thus *?thesis* **by** *simp*
qed
hence *?L1* $= (\int \omega.\ 0\ \partial p)$
by (*intro arg-cong2*[**where** *f*=*measure-pmf.expectation*]) *auto*
also have ... $= 0$
by *simp*
finally show *?thesis* **by** *simp*
qed

have *b*: *prob-space.indep-vars p* (λ-. *discrete*) $(\lambda i\ \omega.\ \omega\ i)$ $(set\ (fst\ x) \cup set\ (snd\ x))$
if $x \in L$ **for** x
proof −
have $card\ (set\ (fst\ x) \cup set\ (snd\ x)) \leq card\ (set\ (fst\ x)) + card\ (set\ (snd\ x))$
by (*intro card-Un-le*)
also have ... $\leq length\ (fst\ x) + length\ (snd\ x)$
by (*intro add-mono card-length*)
also have ... $= s + t$
using *that L-def* **by** *auto*

```
  also have ... ≤ k using assms(3) by simp
  finally have card (set (fst x) ∪ set (snd x)) ≤ k by simp
  moreover have set (fst x) ∪ set (snd x) ⊆ R
    using that L-def by auto
  ultimately show ?thesis
  by (intro prob-space.k-wise-indep-vars-subset[OF prob-space-measure-pmf indep[OF assms(4)]])
     auto
qed

have c: (∫ ω. of-bool (ω x = z) ∂p) = α (is ?L1 = -)
  if z ∈ B x ∈ R for x z
proof −
  have ?L1 = (∫ ω. indicator {ω. ω x = z} ω ∂p)
    unfolding indicator-def by simp
  also have ... = measure p {ω. ω x = z}
    by simp
  also have ... = measure (map-pmf (λω. ω x) p) {z}
    by (subst measure-map-pmf) (simp add:vimage-def)
  also have ... = measure (pmf-of-set B) {z}
    using that by (subst ran[OF assms(4)]) auto
  also have ... = 1/card B
    using fin-B that by (subst measure-pmf-of-set) auto
  also have ... = α
    unfolding α-def by simp
  finally show ?thesis by simp
qed

have d: abs x ≤ 1 ⟹ abs y ≤ 1 ⟹ abs (x*y) ≤ 1 for x y :: real
  by (simp add:abs-mult mult-le-one)
have e:(⋀x. x ∈ set xs ⟹ abs x ≤1) ⟹ abs(prod-list xs) ≤ 1  for xs :: real list
  using d by (induction xs, simp, simp)

have ?L = (∫ ω. (∑ j ∈ R. W1 j ω)^s * (∑ j ∈ R. W2 j ω)^t ∂p)
  unfolding Z1-eq Z2-eq by simp
also have ... = (∫ ω. (∑ xs | set xs ⊆ R ∧ length xs = s. (∏ x ← xs. W1 x ω)) *
                      (∑ ys | set ys ⊆ R ∧ length ys = t. (∏ y ← ys. W2 y ω)) ∂p)
  unfolding sum-power-distrib[OF fin-R] by simp
also have ... = (∫ ω.
  (∑ l∈{xs. set xs ⊆ R ∧ length xs = s} × {ys. set ys ⊆ R ∧ length ys = t}.
    (∏ x←fst l. W1 x ω) * (∏ y←snd l. W2 y ω)) ∂p)
  by (intro arg-cong[where f=integral^L p])
    (simp add: sum-product sum.cartesian-product case-prod-beta)
also have ... = (∑ l∈{xs. set xs ⊆ R ∧ length xs = s} × {ys. set ys ⊆ R ∧ length ys = t}.
  (∫ ω. (∏ x←fst l. W1 x ω) * (∏ y←snd l. W2 y ω) ∂p))
  unfolding W1-def W2-def
  by (intro integral-sum integrable-pmf-iff-bounded[where C=1] d e) auto
also have ... = (∑ l∈ L. (∫ ω. (∏ x←fst l. W1 x ω) * (∏ y←snd l. W2 y ω) ∂p))
  unfolding L-def using a by (intro sum.mono-neutral-right finite-cartesian-product
      finite-lists-length-eq fin-R) auto
also have ... = (∑ l∈ L. (∫ ω. (∏ x←fst l.
  of-bool(ω x = τ l x)) * (∏ y←snd l. of-bool(ω y = τ l y)) ∂p))
  unfolding W1-def W2-def using τ-check-1 τ-check-2
  by (intro sum.cong arg-cong[where f=integral^L p] ext arg-cong2[where f=(*)]
      arg-cong[where f=prod-list]) auto
also have ... = (∑ l∈ L. (∫ ω. (∏ x←(fst l@snd l). of-bool(ω x = τ l x))∂ p))
  by simp
also have ... = (∑ l∈ L. (∫ ω. (∏ x ∈ set (fst l@snd l).
  of-bool(ω x = τ l x)^count-list (fst l@snd l) x) ∂ p))
```

```
    unfolding prod-list-eval by simp
  also have ... = (∑ l∈ L. (∫ ω. (∏ x ∈ set (fst l) ∪ set (snd l).
    of-bool(ω x = τ l x)^count-list (fst l@snd l) x) ∂ p))
    by simp
  also have ... = (∑ l∈ L. (∫ ω. (∏ x ∈ set (fst l) ∪ set (snd l). of-bool(ω x = τ l x)) ∂ p))
    using count-list-gr-1
    by (intro sum.cong arg-cong[where f=integral^L p] ext prod.cong) force+
  also have ... = (∑ l∈ L. (∏ x ∈ set (fst l) ∪ set (snd l). (∫ ω. of-bool(ω x = τ l x) ∂ p)))
    by (intro sum.cong prob-space.indep-vars-lebesgue-integral[OF prob-space-measure-pmf]
        integrable-pmf-iff-bounded[where C=1]
        prob-space.indep-vars-compose2[OF prob-space-measure-pmf b]) auto
  also have ... = (∑ l∈ L. (∏ x ∈ set (fst l) ∪ set (snd l). α))
    using τ-check-3 unfolding L-def by (intro sum.cong prod.cong c) auto
  also have ... = (∑ l ∈ L. α^(card (set (fst l) ∪ set (snd l))))
    by simp
  also have ... = ?R
    unfolding L-def α-def by (simp add:case-prod-beta)
  finally show ?thesis by simp
qed

lemma hit-count-prod-pow-eq:
  assumes i ∈ B j ∈ B
  assumes lim-balls-and-bins k p
  assumes lim-balls-and-bins k q
  assumes s+t ≤ k
  shows (∫ ω. (Z i ω)^s * (Z j ω)^t ∂p) = (∫ ω. (Z i ω)^s * (Z j ω)^t ∂q)
    unfolding hit-count-prod-exp[OF assms(1,2,5,3)]
    unfolding hit-count-prod-exp[OF assms(1,2,5,4)]
    by simp

lemma hit-count-sum-pow-eq:
  assumes i ∈ B j ∈ B
  assumes lim-balls-and-bins k p
  assumes lim-balls-and-bins k q
  assumes s ≤ k
  shows (∫ ω. (Z i ω + Z j ω)^s ∂p) = (∫ ω. (Z i ω + Z j ω)^s ∂q)
    (is ?L = ?R)
proof −
  have q2: |Z i x ^ l * Z j x ^ (s − l)| ≤ real (card R ^ s)
    if l ∈ {..s} for s i j l x
  proof −
    have |Z i x ^ l * Z j x ^ (s − l)| ≤ Z i x ^ l * Z j x ^ (s − l)
      unfolding Z-def by auto
    also have ... ≤ real (card R) ^ l * real (card R) ^ (s−l)
      unfolding Z-def
      by (intro mult-mono power-mono of-nat-mono card-mono fin-R) auto
    also have ... = real (card R)^s using that
      by (subst power-add[symmetric]) simp
    also have ... = real (card R^s)
      by simp
    finally show ?thesis by simp
  qed

  have ?L = (∫ ω. (∑ l≤s. real (s choose l) * (Z i ω^l * Z j ω^(s−l))) ∂p)
    by (subst binomial-ring) (simp add:algebra-simps)
  also have ... = (∑ l≤s. (∫ ω. real (s choose l) * (Z i ω^l * Z j ω^(s−l)) ∂p))
    by (intro integral-sum integrable-mult-right
        integrable-pmf-iff-bounded[where C=card R^s] q2) auto
```

```
  also have ... = (∑ l≤s. real (s choose l) * (∫ ω. (Z i ω^l * Z j ω^(s−l)) ∂p))
    by (intro sum.cong integral-mult-right
        integrable-pmf-iff-bounded[where C=card R^s] q2) auto
  also have ... = (∑ l≤s. real (s choose l) * (∫ ω. (Z i ω^l * Z j ω^(s−l)) ∂q))
    using assms(5)
    by (intro-cong [σ2 (*)] more: sum.cong hit-count-prod-pow-eq[OF assms(1−4)])
       auto
  also have ... = (∑ l≤s. (∫ ω. real (s choose l) * (Z i ω^l * Z j ω^(s−l)) ∂q))
    by (intro sum.cong integral-mult-right[symmetric]
        integrable-pmf-iff-bounded[where C=card R^s] q2) auto
  also have ... = (∫ ω. (∑ l≤s. real (s choose l) * (Z i ω^l * Z j ω^(s−l))) ∂q)
    by (intro integral-sum[symmetric] integrable-mult-right
        integrable-pmf-iff-bounded[where C=card R^s] q2) auto
  also have ... = ?R
    by (subst binomial-ring) (simp add:algebra-simps)
  finally show ?thesis by simp
qed

lemma hit-count-sum-poly-eq:
  assumes i ∈ B j ∈ B
  assumes lim-balls-and-bins k p
  assumes lim-balls-and-bins k q
  assumes f ∈ ℙ k
  shows (∫ ω. f (Z i ω + Z j ω) ∂p) = (∫ ω. f (Z i ω + Z j ω) ∂q)
    (is ?L = ?R)
proof −
  obtain fp where f-def: f = poly fp degree fp ≤ k
    using assms(5) unfolding Polynomials-def by auto

  have ?L = (∑ d≤degree fp. (∫ ω. poly.coeff fp d * (Z i ω + Z j ω) ^ d ∂p))
    unfolding f-def poly-altdef
    by (intro integral-sum integrable-mult-right Z-any-integrable-2[OF assms(3)])
  also have ... = (∑ d≤degree fp. poly.coeff fp d * (∫ ω. (Z i ω + Z j ω) ^ d ∂p))
    by (intro sum.cong integral-mult-right Z-any-integrable-2[OF assms(3)])
     simp
  also have ... = (∑ d≤degree fp. poly.coeff fp d *(∫ ω. (Z i ω + Z j ω) ^ d ∂q))
    using f-def
    by (intro sum.cong arg-cong2[where f=(*)] hit-count-sum-pow-eq[OF assms(1−4)]) auto
  also have ... = (∑ d≤degree fp. (∫ ω. poly.coeff fp d * (Z i ω + Z j ω) ^ d ∂q))
    by (intro sum.cong) auto
  also have ... = ?R
    unfolding f-def poly-altdef by (intro integral-sum[symmetric]
        integrable-mult-right Z-any-integrable-2[OF assms(4)])
  finally show ?thesis by simp
qed

lemma hit-count-poly-eq:
  assumes b ∈ B
  assumes lim-balls-and-bins k p
  assumes lim-balls-and-bins k q
  assumes f ∈ ℙ k
  shows (∫ ω. f (Z b ω) ∂p) = (∫ ω. f (Z b ω) ∂q) (is ?L = ?R)
proof −
  have a:(λa. f (a / 2)) ∈ ℙ (k*1)
    by (intro Polynomials-composeI[OF assms(4)] Polynomials-intros)
  have ?L = ∫ ω. f ((Z b ω + Z b ω)/2) ∂p
    by simp
  also have ... = ∫ ω. f ((Z b ω + Z b ω)/2) ∂q
```

```
    using a by (intro hit-count-sum-poly-eq[OF assms(1,1,2,3)]) simp
  also have ... = ?R by simp
  finally show ?thesis by simp
qed

lemma lim-balls-and-bins-from-ind-balls-and-bins:
  lim-balls-and-bins k Ω
proof −
  have prob-space.indep-vars (measure-pmf Ω) (λ-. discrete) (λx ω. ω x) R
    unfolding Ω-def using indep-vars-Pi-pmf[OF fin-R] by metis
  hence prob-space.indep-vars (measure-pmf Ω) (λ-. discrete) (λx ω. ω x) J if J ⊆ R for J
    using prob-space.indep-vars-subset[OF prob-space-measure-pmf - that] by auto
  hence a:prob-space.k-wise-indep-vars (measure-pmf Ω) k (λ-. discrete) (λx ω. ω x) R
    by (simp add:prob-space.k-wise-indep-vars-def[OF prob-space-measure-pmf])

  have b: map-pmf (λω. ω x) Ω = pmf-of-set B if x ∈ R for x
    using that unfolding Ω-def Pi-pmf-component[OF fin-R] by simp

  show ?thesis
    using a b fin-R fin-B unfolding lim-balls-and-bins-def by auto
qed

lemma hit-count-factorial-moments:
  assumes a:j ∈ B
  assumes s ≤ k
  assumes lim-balls-and-bins k p
  shows (∫ ω. ffact s (Z j ω) ∂p) = ffact s (real (card R)) * (1 / real (card B))^s
    (is ?L = ?R)
proof −
  have (λx. ffact s (x−0::real)) ∈ ℙ s
    by (intro Polynomials-intros)
  hence b: ffact s ∈ (ℙ k :: (real ⇒ real) set)
    using Polynomials-mono[OF assms(2)] by auto

  have ?L = (∫ ω. ffact s (Z j ω) ∂Ω)
    by (intro hit-count-poly-eq[OF a assms(3) lim-balls-and-bins-from-ind-balls-and-bins] b)
  also have ... = (∫ ω. ffact s (∑ i ∈ {j}. Z i ω) ∂Ω)
    by simp
  also have ... = ffact s (real (card R)) * (real (card {j}) / real (card B)) ^ s
    using assms(1)
    by (intro fact-moment-balls-and-bins fin-R fin-B) auto
  also have ... = ?R
    by simp
  finally show ?thesis by simp
qed

lemma hit-count-factorial-moments-2:
  assumes a:i ∈ B j ∈ B
  assumes i ≠ j s ≤ k card R ≤ card B
  assumes lim-balls-and-bins k p
  shows (∫ ω. ffact s (Z i ω + Z j ω) ∂p) ≤ 2^s
    (is ?L ≤ ?R)
proof −
  have (λx. ffact s (x−0::real)) ∈ ℙ s
    by (intro Polynomials-intros)
  hence b: ffact s ∈ (ℙ k :: (real ⇒ real) set)
    using Polynomials-mono[OF assms(4)] by auto
```

```
  have or-distrib: (a ∧ b) ∨ (a ∧ c) ⟷ a ∧ (b ∨ c) for a b c
    by auto
  have ?L = (∫ ω. ffact s  (Z i ω + Z j ω) ∂Ω)
    by (intro hit-count-sum-poly-eq[OF a assms(6) lim-balls-and-bins-from-ind-balls-and-bins] b)
  also have ... = (∫ ω. ffact s ((∑ t ∈ {i,j}. Z t ω)) ∂Ω)
    using assms(3) by simp
  also have ... = ffact s (real (card R)) * (real (card {i,j}) / real (card B)) ^ s
    using assms(1,2)
    by (intro fact-moment-balls-and-bins fin-R fin-B) auto
  also have ... = real (ffact s (card R)) * (real (card {i,j}) / real (card B)) ^ s
    by (simp add:of-nat-ffact)
  also have ... ≤ (card R)^s * (real (card {i,j}) / real (card B)) ^ s
    by (intro mult-mono of-nat-mono ffact-bound, simp-all)
  also have ... ≤ (card B)^s * (real (2) / real (card B)) ^ s
    using assms(3)
    by (intro mult-mono of-nat-mono power-mono assms(5), simp-all)
  also have ... = ?R
    using card-B-gt-0 by (simp add:divide-simps)
  finally show ?thesis by simp
qed

lemma balls-and-bins-approx-helper:
  fixes x :: real
  assumes x ≥ 2
  assumes real k ≥ 5*x / ln x
  shows k ≥ 2
    and 2^(k+3) / fact k ≤ (1/exp x)^2
    and 2 / fact k ≤ 1 / (exp 1 * exp x)
proof −
  have ln-inv: ln x = − ln (1/ x)  if x > 0 for x :: real
    using that by (subst ln-div, auto)

  have apx:
    exp 1 ≤ (5::real)
    4 * ln 4 ≤  (2 − 2*exp 1/5)*ln (450::real)
    ln 8 * 2 ≤ (450::real)
    4 / 5 * 2 * exp 1 + ln (5 / 4) * exp 1 ≤ (5::real)
    exp 1 ≤ (2::real)^4
    by (approximation 10)+

  have 2 ≤ 5 * (x / (x−1))
    using assms(1) by (simp add:divide-simps)
  also have ... ≤ 5 * (x / ln x)
    using assms(1)
    by (intro mult-left-mono divide-left-mono ln-le-minus-one mult-pos-pos) auto
  also have ... ≤ k using assms(2) by simp
  finally show k-ge-2: k ≥ 2 by simp

  have ln x * (2 * exp 1) = ln (((4/5) * x)*(5/4)) * (2 * exp 1)
    by simp
  also have ... = ln ((4/5) * x) * (2 * exp 1) + ln((5/4))*(2 * exp 1)
    using assms(1) by (subst ln-mult, simp-all add:algebra-simps)
  also have ... < (4/5)* x * (2 * exp 1) + ln (5/4) * (x * exp 1)
    using assms(1) by (intro add-less-le-mono mult-strict-right-mono ln-less-self
        mult-left-mono mult-right-mono)  (auto simp add:algebra-simps)
  also have ... = ((4/5) * 2 * exp 1 + ln(5/4) * exp 1) * x
    by (simp add:algebra-simps)
  also have ... ≤ 5 * x
```

```
  using assms(1) apx(4) by (intro mult-right-mono, simp-all)
finally have 1: ln x * (2 * exp 1) ≤ 5 * x by simp

have ln 8 ≤ 3 * x - 5 * x * ln(2*exp 1 /5 * ln x) / ln x
proof (cases x ∈ {2..450})
  case True
  then show ?thesis by (approximation 10 splitting: x=10)
next
  case False
  hence x-ge-450: x ≥ 450 using assms(1) by simp

  have 4 * ln 4 ≤  (2 - 2*exp 1/5)*ln (450::real)
    using apx(2) by (simp)
  also have ... ≤ (2 - 2*exp 1/5)* ln x
    using x-ge-450 apx(1)
    by (intro mult-left-mono iffD2[OF ln-le-cancel-iff], simp-all)
  finally have (2 - 2*exp 1/5) * ln x ≥ 4 * ln 4 by simp
  hence 2*exp 1/5 * ln x + 0 ≤ 2 * exp 1 / 5 * ln x + ((2 - 2*exp 1/5) * ln x - 4 * ln 4)
    by (intro add-mono) auto
  also have ... = 4 * (1/2) * ln x - 4 * ln 4
    by (simp add:algebra-simps)
  also have ... = 4 * (ln (x powr (1/2)) - ln 4)
    using x-ge-450 by (subst ln-powr, auto)
  also have ... = 4 * (ln (x powr (1/2)/4))
    using x-ge-450 by (subst ln-div) auto
  also have ... < 4 * (x powr (1/2)/4)
    using x-ge-450 by (intro mult-strict-left-mono ln-less-self) auto
  also have ... = x powr (1/2) by simp
  finally have §: 2 * exp 1/ 5 * ln x ≤ x powr (1/2) by simp
  hence ln(2* exp 1/ 5 * ln x) ≤ ln (x powr (1/2))
    using x-ge-450 by (intro ln-mono; simp)
  hence 0:ln(2* exp 1/ 5 * ln x) / ln x ≤ 1/2
    using x-ge-450 by (subst (asm) ln-powr, auto)
  have ln 8 ≤ 3 * x - 5 * x * (1/2)
    using x-ge-450 apx(3) by simp
  also have ... ≤ 3 * x - 5 * x * (ln(2* exp 1/ 5 * ln x) / ln x)
    using x-ge-450 by (intro diff-left-mono mult-left-mono 0) auto
  finally show ?thesis by simp
qed

hence 2 * x + ln 8 ≤ 2 * x + (3 * x - 5 * x * ln(2*exp 1 /5 * ln x) / ln x)
  by (intro add-mono, auto)
also have ... = 5 * x + 5 * x * ln(5 / (2*exp 1*ln x)) / ln x
  using assms(1) by (subst ln-inv) (auto simp add:algebra-simps)
also have ... = 5 * x * (ln x + ln(5 / (2*exp 1*ln x))) / ln x
  using assms(1) by (simp add:algebra-simps add-divide-distrib)
also have ... = 5 * x * (ln (5 * x / (2 * exp 1 * ln x))) / ln x
  using assms(1) by (simp add: ln-mult ln-div)
also have ... = (5 * x / ln x) * ln ((5 * x / ln x) / (2 * exp 1))
  by (simp add:algebra-simps)
also have ... ≤ k * ln (k / (2*exp 1))
  using assms(1,2) 1 k-ge-2
  by (intro mult-mono iffD2[OF ln-le-cancel-iff] divide-right-mono)
   auto
finally have k * ln (k/(2*exp 1)) ≥ 2*x + ln 8 by simp
hence k * ln(2*exp 1/k) ≤ -2*x - ln 8
  using k-ge-2 by (subst ln-inv, auto)
hence ln ((2*exp 1/k) powr k) ≤ ln(exp(-2*x)) - ln 8
```

```
    using k-ge-2 by (subst ln-powr, auto)
  also have ... = ln(exp(−2∗x)/8)
    by (simp add:ln-div)
  finally have ln ((2∗exp 1/k) powr k) ≤  ln (exp(−2∗x)/8) by simp
  hence 1: (2∗exp 1/k) powr k ≤  exp(−2∗x)/8
    using k-ge-2 assms(1) by (subst (asm) ln-le-cancel-iff) auto
  have 2^(k+3)/fact k ≤ 2^(k+3)/(k / exp 1)^k
    using k-ge-2 by (intro divide-left-mono fact-lower-bound-1) auto
  also have ... = 8 ∗ 2^k ∗ (exp 1 / k)^k
    by (simp add:power-add algebra-simps power-divide)
  also have ... = 8 ∗ (2∗exp 1/k) powr k
    using k-ge-2 powr-realpow
    by (simp add:power-mult-distrib[symmetric])
  also have ... ≤ 8 ∗ (exp(−2∗x)/8)
    by (intro mult-left-mono 1) auto
  also have ... = exp((−x)∗2)
    by simp
  also have ... = exp(−x)^2
    by (subst exp-powr[symmetric], simp)
  also have ... = (1/exp x)^2
    by (simp add: exp-minus inverse-eq-divide)
  finally show 2:2^(k+3)/fact k ≤ (1/exp x)^2 by simp

  have (2::real)/fact k = (2^(k+3)/fact k)/(2^(k+2))
    by (simp add:divide-simps power-add)
  also have ... ≤ (1/exp x)^2/(2^(k+2))
    by (intro divide-right-mono 2, simp)
  also have ... ≤ (1/exp x)^1/(2^(k+2))
    using assms(1) by (intro divide-right-mono power-decreasing) auto
  also have ... ≤ (1/exp x)^1/(2^4)
    using k-ge-2 by (intro divide-left-mono power-increasing) auto
  also have ... ≤ (1/exp x)^1/exp(1)
    using k-ge-2 apx(5) by (intro divide-left-mono) auto
  also have ... = 1/(exp 1 ∗ exp x) by simp
  finally show (2::real)/fact k ≤ 1/(exp 1 ∗ exp x) by simp
qed
```

Bounds on the expectation and variance in the k-wise independent case. Here the indepedence assumption is improved by a factor of two compared to the result in the paper.

```
lemma
  assumes card R ≤ card B
  assumes ⋀c. lim-balls-and-bins (k+1) (p c)
  assumes ε ∈ {0<..1/exp(2)}
  assumes k ≥ 5 ∗ ln (card B / ε) / ln (ln (card B / ε))
  shows
    exp-approx: |measure-pmf.expectation (p True) Y − measure-pmf.expectation (p False) Y| ≤
      ε ∗ real (card R) (is ?A) and
    var-approx: |measure-pmf.variance (p True) Y − measure-pmf.variance (p False) Y| ≤ ε^2
      (is ?B)
proof −
  let ?p1 = p False
  let ?p2 = p True

  have exp (2::real) = 1/ (1/exp 2) by simp
  also have ... ≤ 1/ ε
    using assms(3) by (intro divide-left-mono) auto
  also have ... ≤ real (card B)/ ε
    using assms(3) card-B-gt-0 by (intro divide-right-mono) auto
```

finally have *exp 2* $\leq$ *real* (*card B*) / ε **by** *simp*
hence *k-condition-h*: *2* $\leq$ *ln* (*card B* / ε)
  using *assms*(*3*) *card-B-gt-0* **by** (*subst ln-ge-iff*) *auto*
have *k-condition-h-2*: *0* < *real* (*card B*) / ε
  using *assms*(*3*) *card-B-gt-0* **by** (*intro divide-pos-pos*) *auto*

note *k-condition* = *balls-and-bins-approx-helper*[*OF k-condition-h assms*(*4*)]

define φ :: *real* $\Rightarrow$ *real* **where** φ = (λ*x*. *min x 1*)

define *f* **where** *f* = (λ*x*. *1* − (−*1*)^*k* / *real* (*fact k*) * *ffact k* (*x*−*1*))
define *g* **where** *g* = (λ*x*. φ *x* − *f x*)
have φ*-exp*: φ *x* = *f x* + *g x* **for** *x*
  unfolding *g-def* **by** *simp*

have *k-ge-2*: *k* $\geq$ *2*
  using *k-condition*(*1*) **by** *simp*

define γ **where** γ = *1* / *real* (*fact k*)
have γ*-nonneg*: γ $\geq$ *0*
  unfolding γ*-def* **by** *simp*

have *k-le-k-plus-1*: *k* $\leq$ *k*+*1*
  by *simp*

have *f* $\in$ $\mathbb{P}$ *k*
  unfolding *f-def* **by** (*intro Polynomials-intros*)
hence *f-poly*: *f* $\in$ $\mathbb{P}$ (*k*+*1*)
  using *Polynomials-mono*[*OF k-le-k-plus-1*] **by** *auto*

have *g-diff*: |*g x* − *g* (*x*−*1*)| = *ffact* (*k*−*1*) (*x*−*2*) / *fact* (*k*−*1*)
  if *x* $\geq$ *k* **for** *x* :: *real*
proof −
  have *x* $\geq$ *2* **using** *k-ge-2 that* **by** *simp*
  hence φ *x* = φ (*x*− *1*)
    unfolding φ*-def* **by** *simp*
  hence |*g x* − *g* (*x*−*1*)| = |*f* (*x*−*1*) − *f x*|
    unfolding *g-def* **by** (*simp add*:*algebra-simps*)
  also have ... = |(−*1*)^*k* / *real* (*fact k*) * (*ffact k* (*x*−*2*) − *ffact k* (*x*−*1*))|
    unfolding *f-def* **by** (*simp add*:*algebra-simps*)
  also have ... = *1* / *real* (*fact k*) * |*ffact k* (*x*−*1*) − *ffact k* ((*x*−*1*)−*1*)|
    by (*simp add*:*abs-mult*)
  also have ... = *1* / *real* (*fact k*) * *real k* * |*ffact* (*k*−*1*) (*x*−*2*)|
    by (*subst ffact-suc-diff*, *simp add*:*abs-mult*)
  also have ... = |*ffact* (*k*−*1*) (*x*−*2*)| / *fact* (*k*−*1*)
    using *k-ge-2* **by** (*subst fact-reduce*) *auto*
  also have ... = *ffact* (*k*−*1*) (*x*−*2*) / *fact* (*k*−*1*)
    unfolding *ffact-eq-fact-mult-binomial* **using** *that k-ge-2*
    by (*intro arg-cong2*[**where** *f*=(/)] *abs-of-nonneg ffact-nonneg*) *auto*
  finally show *?thesis* **by** *simp*
qed

have *f-approx-*φ: *f x* = φ *x* **if** *f-approx-*φ*-1*: *x* $\in$ *real* ‘ {*0*..*k*} **for** *x*
proof (*cases x* = *0*)
  case *True*
  hence *f x* = *1* − (−*1*)^*k* / *real* (*fact k*) * (∏ *i* = *0*..<*k*. − (*real i*+*1*))
    unfolding *f-def prod-ffact*[*symmetric*] **by** (*simp add*:*algebra-simps*)

```
  also have ... = 1 - (-1)^k / real (fact k) * ((∏ i = 0..<k. (- 1)::real) * (∏ i = 0..<k. real i+1))
    by (subst prod.distrib[symmetric]) simp
  also have ... = 1 - (-1)^k / real (fact k) * ((-1)^k * (∏ i ∈ (λx. x + 1) ` {0..<k}. real i))
    by (subst prod.reindex, auto simp add:inj-on-def comp-def algebra-simps)
  also have ... = 1 - (-1)^k / real (fact k) * ((-1)^k * (∏ i ∈ {1..k}. real i))
    by (intro arg-cong2[where f=(-)] arg-cong2[where f=(*)] prod.cong refl) auto
  also have ... = 0
    unfolding fact-prod by simp
  also have ... = φ x
    using True φ-def by simp
  finally show ?thesis by simp
next
  case False
  hence a: x ≥ 1 using that by auto
  obtain x' where x'-def: x' ∈ {0..k} x = real x'
    using f-approx-φ-1 by auto
  hence x' - 1 ∈ {0..<k} using k-ge-2 by simp
  moreover have x- real 1 = real (x'-1)
    using False x'-def(2) by simp
  ultimately have b: x - 1 = real (x' - 1) x' - 1 < k
    by auto

  have f x =  1 - (- 1) ^ k / real (fact k) * real (ffact k (x' - 1))
    unfolding f-def b of-nat-ffact by simp
  also have ... = 1
    using b by (subst ffact-nat-triv, auto)
  also have ... = φ x
    unfolding φ-def using a by auto
  finally show ?thesis by simp
qed

have q2: |Z i x ^ l * Z j x ^ (s - l)| ≤ real (card R ^ s)
  if l ∈ {..s} for s i j l x
proof -
  have |Z i x ^ l * Z j x ^ (s - l)| ≤ Z i x ^ l * Z j x ^ (s - l)
    unfolding Z-def by auto
  also have ... ≤ real (card R) ^ l * real (card R) ^ (s-l)
    unfolding Z-def
    by (intro mult-mono power-mono of-nat-mono card-mono fin-R) auto
  also have ... = real (card R)^s using that
    by (subst power-add[symmetric]) simp
  also have ... = real (card R^s)
    by simp
  finally show ?thesis by simp
qed

have q:real (card A) + real (card B) ∈ real ` {..2 * card R} if A ⊆ R B ⊆ R for A B
proof -
  have card A + card B ≤ card R + card R
    by (intro add-mono card-mono fin-R that)
  also have ... = 2 * card R by simp
  finally show ?thesis by force
qed

have g-eq-0-iff-2: abs (g x) * y = 0 if x ∈ ℤ x ≥ 0 x ≤ k for x y :: real
proof -
  have ∃ x'. x = real-of-int x' ∧ x' ≤ k ∧ x' ≥ 0
```

```
    using that Ints-def by fastforce
  hence ∃x'. x = real x' ∧ x' ≤ k
    by (metis nat-le-iff of-nat-nat)
  hence x ∈ real ‘ {0..k}
    by auto
  hence g x = 0
    unfolding g-def using f-approx-φ by simp
  thus ?thesis by simp
qed

have g-bound-abs: |∫ ω. g (f ω) ∂p| ≤ (∫ ω. ffact (k+1) (f ω) ∂p) * γ
  (is ?L ≤ ?R)
  if range f ⊆ real ‘ {..m} for m and p :: ('a ⇒ 'b) pmf and f :: ('a ⇒ 'b) ⇒ real
proof −
  have f-any-integrable:
    integrable p (λω. h (f ω)) for h :: real ⇒ real
    using that
    by (intro integrable-pmf-iff-bounded[where C=Max (abs ‘ h‘ real ‘ {..m})]
        Max-ge finite-imageI imageI) auto

  have f-val: f ω ∈ real ‘ {..m} for ω using that by auto
  hence f-nat: f ω ∈ ℕ for ω
    unfolding Nats-def by auto

  have f-int: f ω ≥ real y + 1 if f ω > real y for y ω
  proof −
    obtain x where x-def: f ω = real x x ≤ m using f-val by auto
    hence y < x using that by simp
    hence y + 1 ≤ x by simp
    then show ?thesis using x-def by simp
  qed

  have f-nonneg: f ω ≥ 0 for ω
  proof −
    obtain x where x-def: f ω = real x x ≤ m using f-val by auto
    hence x ≥ 0 by simp
    then show ?thesis using x-def by simp
  qed

  have ¬(real x ≤ f ω) if x > m for x ω
  proof −
    obtain x where x-def: f ω = real x x ≤ m using f-val by auto
    then show ?thesis using x-def that by simp
  qed

  hence max-Z1: measure p {ω. real x ≤ f ω} = 0 if x > m for x
    using that by auto

  have ?L ≤ (∫ ω. |g (f ω)| ∂p)
    by (intro integral-abs-bound)
  also have ... = (∑ y∈real ‘ {..m}. |g y| * measure p {ω. f ω = y})
    using that by (intro pmf-exp-of-fin-function) auto
  also have ... = (∑ y∈{..m}. |g (real y)| * measure p {ω. f ω = real y})
    by (subst sum.reindex) (auto simp add:comp-def)
  also have ... = (∑ y∈{..m}. |g (real y)| *
    (measure p ({ω. f ω = real y} ∪ {ω. f ω > y}) − measure p {ω. f ω > y}))
    by (subst measure-Union) auto
  also have ... = (∑ y∈{..m}. |g (real y)| * (measure p {ω. f ω ≥ y} − measure p {ω. f ω >
```

```
y}))
    by (intro sum.cong arg-cong2[where f=(*)] arg-cong2[where f=(-)]
        arg-cong[where f=measure p]) auto
  also have ... = (∑ y∈{..m}. |g (real y)| * measure p {ω. f ω ≥ y}) -
    (∑ y∈{..m}. |g (real y)| * measure p {ω. f ω > y})
    by (simp add:algebra-simps sum-subtractf)
  also have ... = (∑ y∈{..m}. |g (real y)| * measure p {ω. f ω ≥ y}) -
                  (∑ y∈{..m}. |g (real y)| * measure p {ω. f ω ≥ real (y+1)})
    using f-int
    by (intro sum.cong arg-cong2[where f=(-)] arg-cong2[where f=(*)]
        arg-cong[where f=measure p]) fastforce+
  also have ... = (∑ y∈{..m}. |g (real y)     | * measure p {ω. f ω ≥ real y}) -
          (∑ y∈Suc ` {..m}. |g (real y - 1)| * measure p {ω. f ω ≥ real y})
    by (subst sum.reindex) (auto simp add:comp-def)
  also have ... = (∑ y∈{..m}. |g (real y)     | * measure p {ω. f ω ≥ real y}) -
                (∑ y∈{1..m}. |g (real y - 1)| * measure p {ω. f ω ≥ real y})
    using max-Z1 image-Suc-atMost
    by (intro arg-cong2[where f=(-)] sum.mono-neutral-cong) auto
  also have ... = (∑ y∈{k+1..m}. |g (real y)     | * measure p {ω. f ω ≥ y}) -
                  (∑ y∈{k+1..m}. |g (real y - 1)| * measure p {ω. f ω ≥ y})
    using k-ge-2
    by (intro arg-cong2[where f=(-)] sum.mono-neutral-cong-right ballI g-eq-0-iff-2)
      auto
  also have ... = (∑ y∈{k+1..m}. (|g (real y)| - |g (real y-1)|) * measure p {ω. f ω ≥ y})
    by (simp add:algebra-simps sum-subtractf)
  also have ... ≤ (∑ y∈{k+1..m}. |g (real y)- g (real y-1)| *
    measure p {ω. ffact (k+1) (f ω) ≥ ffact (k+1) (real y)})
    using ffact-mono by (intro sum-mono mult-mono pmf-mono) auto
  also have ... = (∑ y∈{k+1..m}. (ffact (k-1) (real y-2) / fact (k-1)) *
    measure p {ω. ffact (k+1) (f ω) ≥ ffact (k+1) (real y)})
    by (intro sum.cong, simp-all add: g-diff)
  also have ... ≤ (∑ y∈{k+1..m}. (ffact (k-1) (real y-2) / fact (k-1)) *
    ((∫ ω. ffact (k+1) (f ω)∂p) / ffact (k+1) (real y)))
    using k-ge-2 f-nat
    by (intro sum-mono mult-left-mono pmf-markov f-any-integrable
        divide-nonneg-pos ffact-of-nat-nonneg ffact-pos) auto
  also have ... = (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) * (∑ y∈{k+1..m}.
     ffact (k-1) (real y - 2) / ffact (Suc (Suc (k-1))) (real y))
    using k-ge-2 by (simp add:algebra-simps sum-distrib-left)
  also have ... = (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) * (∑ y∈{k+1..m}.
     ffact (k-1) (real y - 2) / (real y * (real y - 1) * ffact (k-1) (real y - 2)))
    by (subst ffact-Suc, subst ffact-Suc, simp)
  also have ... = (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) *
    (∑ y∈{k+1..m}. 1 / (real y * (real y - 1)))
    using order.strict-implies-not-eq[OF ffact-pos] k-ge-2
    by (intro arg-cong2[where f=(*)] sum.cong) auto
  also have ... = (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) *
    (∑ y∈{Suc k..m}. 1 / (real y - 1) - 1/(real y))
    using k-ge-2 by (intro arg-cong2[where f=(*)] sum.cong) (auto simp add: divide-simps)
  also have ... = (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) *
    (∑ y∈{Suc k..m}.  (-1/(real y)) - (-1 / (real (y - 1))))
    using k-ge-2 by (intro arg-cong2[where f=(*)] sum.cong) (auto)
  also have ... =  (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) *
    (of-bool (k ≤ m) * (1/real k-1/real m))
    by (subst sum-telescope-eq, auto)
  also have ... ≤  (∫ ω. ffact (k+1) (f ω) ∂p) / fact (k-1) * (1 / real k)
    using k-ge-2 f-nat
    by (intro mult-left-mono divide-nonneg-nonneg integral-nonneg
```

```
      ffact-of-nat-nonneg) auto
  also have ... = ?R
    using k-ge-2 unfolding γ-def by (cases k) (auto simp add:algebra-simps)
  finally show ?thesis by simp
qed

have z1-g-bound: |∫ ω. g (Z i ω) ∂p c| ≤ (real (card R) / real (card B)) * γ
  (is ?L1 ≤ ?R1) if i ∈ B for i c
proof −

  have ?L1 ≤ (∫ ω. ffact (k+1) (Z i ω) ∂p c) * γ
    unfolding Z-def using fin-R
    by (intro g-bound-abs[where m1=card R]) (auto intro!:imageI card-mono)
  also have ... = ffact (k+1) (real (card R)) * (1 / real (card B))^(k+1) * γ
    using that by (subst hit-count-factorial-moments[OF - - assms(2)], simp-all)
  also have ... = real (ffact (k+1) (card R)) * (1 / real (card B))^(k+1) * γ
    by (simp add:of-nat-ffact)
  also have ... ≤ real (card R^(k+1)) * (1 / real (card B))^(k+1) * γ
    using γ-nonneg
    by (intro mult-right-mono of-nat-mono ffact-bound, simp-all)
  also have ... ≤ (real (card R) / real (card B))^(k+1) * γ
    by (simp add:divide-simps)
  also have ... ≤ (real (card R) / real (card B))^1 * γ
    using assms(1) card-B-gt-0 γ-nonneg by (intro mult-right-mono power-decreasing) auto
  also have ... = ?R1 by simp
  finally show ?thesis by simp
qed

have g-add-bound: |∫ ω. g (Z i ω + Z j ω) ∂p c| ≤ 2^(k+1) * γ
  (is ?L1 ≤ ?R1) if ij-in-B: i ∈ B j ∈ B i ≠ j for i j c
proof −
  have ?L1 ≤ (∫ ω. ffact (k+1) (Z i ω + Z j ω) ∂p c) * γ
    unfolding Z-def using assms(1)
    by (intro g-bound-abs[where m1=2*card R]) (auto intro!:imageI q)
  also have ... ≤ 2^(k+1) * γ
  by (intro γ-nonneg mult-right-mono hit-count-factorial-moments-2[OF that(1,2,3) - assms(1,2)])
     auto
  finally show ?thesis by simp
qed

have Z-poly-diff:
  |(∫ ω. φ (Z i ω) ∂?p1) − (∫ ω. φ (Z i ω) ∂?p2)| ≤ 2 * ((real (card R) / card B) * γ)
  (is ?L ≤ 2 * ?R) if i ∈ B for i
proof −
  note Z-poly-eq =
    hit-count-poly-eq[OF that assms(2)[of True] assms(2)[of False] f-poly]

  have ?L = |(∫ ω. f (Z i ω) ∂?p1) + (∫ ω. g (Z i ω) ∂?p1) −
    (∫ ω. f (Z i ω) ∂?p2) − (∫ ω. g (Z i ω) ∂?p2)|
    using Z-integrable[OF assms(2)] unfolding φ-exp by simp
  also have ... = |(∫ ω. g (Z i ω) ∂?p1) + (− (∫ ω. g (Z i ω) ∂?p2))|
    by (subst Z-poly-eq) auto
  also have ... ≤ |(∫ ω. g (Z i ω) ∂?p1)| + |(∫ ω. g (Z i ω) ∂?p2)|
    by simp
  also have ... ≤ ?R + ?R
    by (intro add-mono z1-g-bound that)
  also have ... = 2 * ?R
    by (simp add:algebra-simps)
```

```
  finally show ?thesis by simp
qed

have Z-poly-diff-2: |(∫ ω. φ (Z i ω) ∂?p1) − (∫ ω. φ (Z i ω) ∂?p2)| ≤ 2 * γ
  (is ?L ≤ ?R) if i ∈ B for i
proof −
  have ?L ≤ 2 * ((real (card R) / real (card B)) * γ)
    by (intro Z-poly-diff that)
  also have ... ≤ 2 * (1 * γ)
    using assms fin-B that γ-nonneg card-gt-0-iff
    by (intro mult-mono that iffD2[OF pos-divide-le-eq]) auto
  also have ... = ?R by simp
  finally show ?thesis by simp
qed

 have Z-poly-diff-3: |(∫ ω. φ (Z i ω + Z j ω) ∂?p2) − (∫ ω. φ (Z i ω + Z j ω) ∂?p1)| ≤
2^(k+2)*γ
  (is ?L ≤ ?R) if i ∈ B j ∈ B i ≠ j for i j
proof −
  note Z-poly-eq-2 =
    hit-count-sum-poly-eq[OF that(1,2) assms(2)[of True] assms(2)[of False] f-poly]

  have ?L = |(∫ ω. f (Z i ω + Z j ω) ∂?p2) + (∫ ω. g (Z i ω + Z j ω) ∂?p2) −
    (∫ ω. f (Z i ω + Z j ω) ∂?p1) − (∫ ω. g (Z i ω + Z j ω) ∂?p1)|
    using Z-any-integrable-2[OF assms(2)] unfolding φ-exp by simp
  also have ... = |(∫ ω. g (Z i ω + Z j ω) ∂?p2) + (− (∫ ω. g (Z i ω + Z j ω) ∂?p1))|
    by (subst Z-poly-eq-2) auto
  also have ... ≤ |(∫ ω. g (Z i ω + Z j ω) ∂?p1)| + |(∫ ω. g (Z i ω + Z j ω) ∂?p2)|
    by simp
  also have ... ≤ 2^(k+1)*γ + 2^(k+1)*γ
    by (intro add-mono g-add-bound that)
  also have ... = ?R
    by (simp add:algebra-simps)
  finally show ?thesis by simp
qed


have Y-eq: Y ω = (∑ i ∈ B. φ (Z i ω)) if ω ∈ set-pmf (p c) for c ω
proof −
  have ω ‘ R ⊆ B
  proof (rule image-subsetI)
    fix x assume a:x ∈ R
    have ω x ∈ set-pmf (map-pmf (λω. ω x) (p c))
      using that by (subst set-map-pmf) simp
    also have ... = set-pmf (pmf-of-set B)
      by (intro arg-cong[where f=set-pmf] assms ran[OF assms(2)] a)
    also have ... = B
      by (intro set-pmf-of-set fin-B B-ne)
    finally show ω x ∈ B by simp
  qed
  hence (ω ‘ R) = B ∩ ω ‘ R
    by auto
  hence Y ω = card (B ∩ ω ‘ R)
    unfolding Y-def by auto
  also have  ... = (∑ i ∈ B. of-bool (i ∈ ω ‘ R))
    unfolding of-bool-def using fin-B by (subst sum.If-cases) auto
  also have  ... = (∑ i ∈ B. of-bool (card {r ∈ R.  ω r = i} > 0))
    using fin-R by (intro sum.cong arg-cong[where f=of-bool])
```

```
    (auto simp add:card-gt-0-iff)
  also have ... = (∑ i ∈ B. φ(Z i ω))
    unfolding φ-def Z-def by (intro sum.cong) (auto simp add:of-bool-def)
  finally show ?thesis by simp
qed

let ?φ2 = (λx y. φ x + φ y − φ (x+y))
let ?Bd = {x ∈ B × B. fst x ≠ snd x}

have Y-sq-eq': Y ω^ 2 = (∑ i ∈ ?Bd. ?φ2 (Z (fst i) ω) (Z (snd i) ω)) + Y ω
  (is ?L = ?R) if ω ∈ set-pmf (p c) for c ω
proof −
  have a: φ (Z x ω) = of-bool(card {r ∈ R. ω r = x} > 0) for x
    unfolding φ-def Z-def by auto
  have b: φ (Z x ω + Z y ω) =
    of-bool( card {r ∈ R. ω r = x} > 0 ∨ card {r ∈ R. ω r = y} > 0) for x y
    unfolding φ-def Z-def by auto
  have c: φ (Z x ω) ∗ φ(Z y ω) = ?φ2 (Z x ω) (Z y ω)  for x y
    unfolding a b of-bool-def by auto
  have d: φ (Z x ω) ∗ φ(Z x ω) = φ (Z x ω)  for x
    unfolding a of-bool-def by auto

  have ?L = (∑ i∈B × B. φ (Z (fst i) ω) ∗ φ (Z (snd i) ω))
    unfolding Y-eq[OF that] power2-eq-square sum-product sum.cartesian-product
    by (simp add:case-prod-beta)
  also have ... = (∑ i∈?Bd ∪ {x ∈ B × B. fst x = snd x}. φ (Z (fst i) ω) ∗ φ (Z (snd i) ω))
    by (intro sum.cong refl) auto
  also have ... = (∑ i∈?Bd. φ (Z (fst i) ω) ∗ φ (Z (snd i) ω)) +
    (∑ i∈{x ∈ B × B. fst x = snd x}. φ (Z (fst i) ω) ∗ φ (Z (snd i) ω))
    using assms fin-B  by (intro sum.union-disjoint, auto)
  also have ... = (∑ i∈?Bd. ?φ2 (Z (fst i) ω) (Z (snd i) ω)) +
    (∑ i∈{x ∈ B × B. fst x = snd x}. φ (Z (fst i) ω) ∗ φ (Z (fst i) ω))
    unfolding c by (intro arg-cong2[where f=(+)] sum.cong) auto
  also have ... = (∑ i∈?Bd. ?φ2 (Z (fst i) ω) (Z (snd i) ω)) +
    (∑ i∈fst ‘ {x ∈ B × B. fst x = snd x}. φ (Z i ω) ∗ φ (Z i ω))
    by (subst sum.reindex, auto simp add:inj-on-def)
  also have ... = (∑ i∈?Bd. ?φ2 (Z (fst i) ω) (Z (snd i) ω)) + (∑ i ∈ B. φ (Z i ω))
    using d by (intro arg-cong2[where f=(+)] sum.cong refl d) (auto simp add:image-iff)
  also have ... = ?R
    unfolding Y-eq[OF that] by simp
  finally show ?thesis by simp
qed

have |integral^L  ?p1 Y − integral^L  ?p2 Y| =
  |(∫ ω. (∑ i ∈ B. φ(Z i ω)) ∂?p1) − (∫ ω. (∑ i ∈ B. φ(Z i ω)) ∂?p2)|
  by (intro arg-cong[where f=abs] arg-cong2[where f=(−)]
      integral-cong-AE AE-pmfI Y-eq) auto
also have ... =
  |(∑ i ∈ B. (∫ ω. φ(Z i ω) ∂?p1)) − (∑ i ∈ B. (∫ ω. φ(Z i ω) ∂?p2))|
  by (intro arg-cong[where f=abs] arg-cong2[where f=(−)]
      integral-sum Z-integrable[OF assms(2)])
also have ... = |(∑ i ∈ B. (∫ ω. φ(Z i ω) ∂?p1) − (∫ ω. φ(Z i ω) ∂?p2))|
  by (subst sum-subtractf) simp
also have ... ≤ (∑ i ∈ B. |(∫ ω. φ(Z i ω) ∂?p1) − (∫ ω. φ(Z i ω) ∂?p2)|)
  by simp
also have ... ≤ (∑ i ∈ B. 2 ∗ ((real (card R) / real (card B)) ∗γ))
  by (intro sum-mono Z-poly-diff)
also have ... ≤ 2 ∗ real (card R) ∗γ
```

using γ-*nonneg* **by** (*simp*)
finally have *Y-exp-diff-1*: $|integral^L\ ?p1\ Y - integral^L\ ?p2\ Y| \leq 2 * real\ (card\ R) * \gamma$
by *simp*

have $|integral^L\ ?p1\ Y - integral^L\ ?p2\ Y| \leq (2\ /\ fact\ k) * real\ (card\ R)$
using *Y-exp-diff-1* **by** (*simp add*:*algebra-simps* γ-*def*)
also have ... $\leq$ $1\ /\ (exp\ 1 * (real\ (card\ B)\ /\ \varepsilon)) * card\ R$
using *k-condition*(*3*) *k-condition-h-2* **by** (*intro mult-right-mono*) *auto*
also have ... = $\varepsilon\ /\ (exp\ 1 * real\ (card\ B)) * card\ R$
by *simp*
also have ... $\leq \varepsilon\ /\ (1 * 1) * card\ R$
using *assms*(*3*) *card-B-gt-0*
by (*intro mult-right-mono divide-left-mono mult-mono*) *auto*
also have ... = $\varepsilon * card\ R$
by *simp*
finally show *?A*
by *simp*

have $|integral^L\ ?p1\ Y - integral^L\ ?p2\ Y| \leq$ $2 * real\ (card\ R) * \gamma$
using *Y-exp-diff-1* **by** *simp*
also have ... $\leq 2 * real\ (card\ B) * \gamma$
by (*intro mult-mono of-nat-mono assms* γ-*nonneg*) *auto*
finally have *Y-exp-diff-2*:
$|integral^L\ ?p1\ Y - integral^L\ ?p2\ Y| \leq 2 * \gamma * real\ (card\ B)$
by (*simp add*:*algebra-simps*)

have *int-Y*: *integrable* (*measure-pmf* (*p c*)) *Y* **for** *c*
using *fin-R card-image-le* **unfolding** *Y-def*
by (*intro integrable-pmf-iff-bounded*[**where** *C*=*card R*]) *auto*

have *int-Y-sq*: *integrable* (*measure-pmf* (*p c*)) ($\lambda\omega.\ Y\ \omega\hat{}2$) **for** *c*
using *fin-R card-image-le* **unfolding** *Y-def*
by (*intro integrable-pmf-iff-bounded*[**where** *C*=*real* (*card R*)$\hat{}2$]) *auto*

have $|(\int \omega.\ (\sum i \in ?Bd.\ ?\varphi 2\ (Z\ (fst\ i)\ \omega)\ (Z\ (snd\ i)\ \omega))\ \partial ?p1)\ -$
$(\int \omega.\ (\sum i \in ?Bd.\ ?\varphi 2\ (Z\ (fst\ i)\ \omega)\ (Z\ (snd\ i)\ \omega))\ \partial ?p2)|$
$\leq |(\sum i \in ?Bd.$
$(\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega)\ \partial ?p1) + (\int \omega.\ \varphi(Z\ (snd\ i)\ \omega)\ \partial ?p1) -$
$(\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p1) - (\ (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega)\ \partial ?p2) +$
$(\int \omega.\ \varphi\ (Z\ (snd\ i)\ \omega)\ \partial ?p2) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p2)))|$ (**is** *?R3* $\leq$ -)
using *Z-integrable*[*OF assms*(*2*)] *Z-any-integrable-2*[*OF assms*(*2*)]
by (*simp add*:*integral-sum sum-subtractf*)
also have ... = $|(\sum i \in ?Bd.$
$((\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega)\ \partial ?p2)) +$
$((\int \omega.\ \varphi\ (Z\ (snd\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (snd\ i)\ \omega)\ \partial ?p2)) +$
$((\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p2) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p1)))|$
by (*intro arg-cong*[**where** *f*=*abs*] *sum.cong*) *auto*
also have ... $\leq (\sum i \in ?Bd.\ |$
$((\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega)\ \partial ?p2)) +$
$((\int \omega.\ \varphi\ (Z\ (snd\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (snd\ i)\ \omega)\ \partial ?p2)) +$
$((\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p2) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p1))|)$
by (*intro sum-abs*)
also have ... $\leq (\sum i \in ?Bd.$
$|(\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega)\ \partial ?p2)| +$
$|(\int \omega.\ \varphi\ (Z\ (snd\ i)\ \omega)\ \partial ?p1) - (\int \omega.\ \varphi(Z\ (snd\ i)\ \omega)\ \partial ?p2)| +$
$|(\int \omega.\ \varphi\ (Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p2) - (\int \omega.\ \varphi(Z\ (fst\ i)\ \omega + Z\ (snd\ i)\ \omega)\ \partial ?p1)|)$
by (*intro sum-mono*) *auto*
also have ... $\leq (\sum i \in ?Bd.\ 2*\gamma + 2 * \gamma + 2\hat{}(k+2)*\gamma)$

by (*intro sum-mono add-mono Z-poly-diff-2 Z-poly-diff-3*) *auto*
also have ... = (2^(k+2)+4) ∗γ ∗ *real* (*card ?Bd*)
by (*simp add*:*algebra-simps*)
finally have *Y-sq-exp-diff-1*:*?R3* ≤ (2^(k+2)+4) ∗γ ∗ *real* (*card ?Bd*)
by *simp*

have |(∫ ω. *Y* ω ^2 ∂*?p1*) − (∫ ω. *Y* ω^2 ∂*?p2*)| =
|(∫ ω. (∑ *i* ∈ *?Bd*. *?φ2* (*Z* (*fst i*) ω) (*Z* (*snd i*) ω)) + *Y* ω ∂*?p1*) −
(∫ ω. (∑ *i* ∈ *?Bd*. *?φ2* (*Z* (*fst i*) ω) (*Z* (*snd i*) ω)) + *Y* ω ∂*?p2*)|
by (*intro-cong* [σ_2 (−), σ_1 *abs*] *more*: *integral-cong-AE AE-pmfI Y-sq-eq'*) *auto*
also have ... ≤ |(∫ ω. *Y* ω ∂*?p1*) − (∫ ω. *Y* ω ∂*?p2*)| +
|(∫ ω. (∑ *i* ∈ *?Bd*. *?φ2* (*Z* (*fst i*) ω) (*Z* (*snd i*) ω)) ∂*?p1*) −
(∫ ω. (∑ *i* ∈ *?Bd*. *?φ2* (*Z* (*fst i*) ω) (*Z* (*snd i*) ω)) ∂*?p2*)|
using *Z-integrable*[*OF assms*(*2*)] *Z-any-integrable-2*[*OF assms*(*2*)] *int-Y* **by** *simp*
also have ... ≤ 2 ∗γ ∗ *real* (*card B*) + *?R3*
by (*intro add-mono Y-exp-diff-2*, *simp*)
also have ... ≤ (2^(k+2)+4) ∗γ ∗ *real* (*card B*) + (2^(k+2)+4) ∗γ ∗ *real* (*card ?Bd*)
using γ-*nonneg* **by** (*intro add-mono Y-sq-exp-diff-1 mult-right-mono*) *auto*
also have ... = (2^(k+2)+4) ∗γ ∗ (*real* (*card B*) + *real* (*card ?Bd*))
by (*simp add*:*algebra-simps*)
also have ... = (2^(k+2)+4) ∗ γ ∗ *real* (*card B*)^2
using *power2-nat-le-imp-le*
by (*simp add*:*card-distinct-pairs of-nat-diff*)
finally have *Y-sq-exp-diff*:
|(∫ ω. *Y* ω ^2 ∂*?p1*) − (∫ ω. *Y* ω^2 ∂*?p2*)| ≤ (2^(k+2)+4) ∗γ ∗ *real* (*card B*)^2 **by** *simp*

have *Y-exp-rough-bound*: |*integral*^{L} (*p c*) *Y*| ≤ *card B* (**is** *?L* ≤ *?R*) **for** *c*
proof −
have *?L* ≤ (∫ ω. |*Y* ω| ∂(*p c*))
by (*intro integral-abs-bound*)
also have ... ≤ (∫ ω. *real* (*card R*) ∂(*p c*))
unfolding *Y-def* **using** *card-image-le*[*OF fin-R*]
by (*intro integral-mono integrable-pmf-iff-bounded*[**where** *C*=*card R*])
auto
also have ... = *card R* **by** *simp*
also have ... ≤ *card B* **using** *assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed

have |*measure-pmf.variance ?p1 Y* − *measure-pmf.variance ?p2 Y*| =
|(∫ ω. *Y* ω ^2 ∂*?p1*) − (∫ ω. *Y* ω ∂ *?p1*)^2 − ((∫ ω. *Y* ω ^2 ∂*?p2*) − (∫ ω. *Y* ω ∂ *?p2*)^2)|
by (*intro-cong* [σ_2 (−), σ_1 *abs*] *more*: *measure-pmf.variance-eq int-Y int-Y-sq*)
also have ... ≤ |(∫ ω. *Y* ω^2 ∂*?p1*) − (∫ ω. *Y* ω^2 ∂*?p2*)| + |(∫ ω. *Y* ω ∂ *?p1*)^2 − (∫ ω. *Y* ω ∂ *?p2*)^2|
by *simp*
also have ... = |(∫ ω. *Y* ω^2 ∂*?p1*) − (∫ ω. *Y* ω^2 ∂*?p2*)| +
|(∫ ω. *Y* ω ∂ *?p1*) − (∫ ω. *Y* ω ∂ *?p2*)|∗|(∫ ω. *Y* ω ∂ *?p1*) + (∫ ω. *Y* ω ∂ *?p2*)|
by (*simp add*:*power2-eq-square algebra-simps abs-mult*[*symmetric*])
also have ... ≤ (2^(k+2)+4) ∗γ ∗ *real* (*card B*)^2 + (2∗γ ∗*real* (*card B*)) ∗
(|∫ ω. *Y* ω ∂*?p1*| + |∫ ω. *Y* ω ∂ *?p2*|)
using γ-*nonneg*
by (*intro add-mono mult-mono divide-left-mono Y-sq-exp-diff Y-exp-diff-2*) *auto*
also have ... ≤ (2^(k+2)+4)∗γ ∗ *real* (*card B*)^2 + (2∗γ ∗ *real* (*card B*)) ∗
(*real* (*card B*) + *real* (*card B*))
using γ-*nonneg* **by** (*intro add-mono mult-left-mono Y-exp-rough-bound*) *auto*
also have ... = (2^(k+2)+2^3) ∗ γ ∗ *real* (*card B*)^2
by (*simp add*:*algebra-simps power2-eq-square*)
also have ... ≤ (2^(k+2)+2^(k+2)) ∗ γ ∗ *real* (*card B*)^2

```
    using k-ge-2 γ-nonneg
    by (intro mult-right-mono add-mono power-increasing, simp-all)
  also have ... = (2^(k+3) / fact k) * card B^2
    by (simp add:power-add γ-def)
  also have ... ≤ (1 / (real (card B) / ε))^2 * card B^2
    using k-condition(2) k-condition-h-2
    by (intro mult-right-mono) auto
  also have ... = ε^2
    using card-B-gt-0 by (simp add:divide-simps)
  finally show ?B
    by simp
qed

lemma
  assumes card R ≤ card B
  assumes lim-balls-and-bins (k+1) p
  assumes k ≥ 7.5 * (ln (card B) + 2)
  shows exp-approx-2: |measure-pmf.expectation p Y − μ| ≤ card R / sqrt (card B)
      (is ?AL ≤ ?AR)
    and var-approx-2: measure-pmf.variance p Y ≤ real (card R)^2 / card B
      (is ?BL ≤ ?BR)
proof −
  define q where q = (λc. if c then Ω else p)

  have q-altdef: q True = Ω q False = p
    unfolding q-def by auto

  have a:lim-balls-and-bins (k+1) (q c) for c
    unfolding q-def using assms lim-balls-and-bins-from-ind-balls-and-bins by auto

  define ε :: real where ε = min (sqrt (1/card B)) (1 / exp 2)

  have c: ε ∈ {0<..1 / exp 2}
    using card-B-gt-0 unfolding ε-def by auto

  have b: 5 * ln (card B / ε) / ln (ln (card B / ε)) ≤ real k
  proof (cases card B ≥ exp 4)
    case True
    hence sqrt(1/card B) ≤ sqrt(1/exp 4)
      using card-B-gt-0 by (intro real-sqrt-le-mono divide-left-mono) auto
    also have ... = (1/exp 2)
      by (subst powr-half-sqrt[symmetric]) (auto simp add:powr-divide exp-powr)
    finally have sqrt(1/card B) ≤ (1 / exp 2) by simp
    hence ε-eq: ε = sqrt(1 /card B)
      unfolding ε-def by simp

    have exp (6::real) = (exp 4) powr (3/2)
      by (simp add:exp-powr)
    also have ...≤ card B powr (3/2)
      by (intro powr-mono2 True) auto
    finally have q4:exp 6 ≤ card B powr (3/2) by simp

    have (2::real) ≤ exp 6
      by (approximation 5)
    hence q1: 2 ≤ real (card B) powr (3 / 2)
      using q4 by argo
    have (1::real) < ln(exp 6)
      by (approximation 5)
```

```
  also have ... ≤ ln (card B powr (3 / 2))
    using card-B-gt-0 by (intro iffD2[OF ln-le-cancel-iff] q4) auto
  finally have q2: 1 < ln (card B powr (3 / 2)) by simp
  have exp (exp (1::real)) ≤ exp 6
    by (approximation 5)
  also have ... ≤ card B powr (3/2) using q4 by simp
  finally have exp (exp 1) ≤ card B powr (3/2)
    by simp
  hence q3: 1≤ln(ln (card B powr (3/2)))
    using card-B-gt-0 q1 by (intro iffD2[OF ln-ge-iff] ln-gt-zero, auto)

  have 5 * ln (card B / ε) / ln (ln (card B / ε)) =
    5 * ln (card B powr (1+1/2)) / ln(ln (card B powr (1+1/2)))
    unfolding powr-add by (simp add:real-sqrt-divide  powr-half-sqrt[symmetric] ε-eq)
  also have ... ≤ 5 * ln (card B powr (1+1/2)) / 1
    using True q1 q2 q3 by (intro divide-left-mono mult-nonneg-nonneg mult-pos-pos
        ln-ge-zero ln-gt-zero) auto
  also have ... = 5 * (1+1/2) * ln(card B)
    using card-B-gt-0 by (subst ln-powr) auto
  also have ... = 7.5 * ln(card B) by simp
  also have ... ≤ k using assms(3) by simp
  finally show ?thesis by simp
next
  case False
  have (1::real) / exp 2 ≤ sqrt(1 / exp 4)
    by (simp add:real-sqrt-divide powr-half-sqrt[symmetric] exp-powr)
  also have ...≤ sqrt(1 /card B)
    using False card-B-gt-0
    by (intro real-sqrt-le-mono divide-left-mono mult-pos-pos) auto
  finally have 1 / exp 2 ≤ sqrt(1/card B)
    by simp
  hence ε-eq: ε = 1/ exp 2
    unfolding ε-def by simp

  have q2:5 * (ln x + 2) / ln (ln x + 2) ≤ 7.5 * (ln x + 2)
    if x ∈ {1..exp 4} for x:: real
    using that by (approximation 10 splitting: x=10)

  have 5 * ln (card B / ε) / ln (ln (card B / ε)) =
       5 * (ln (card B) +2) / ln (ln (card B) + 2)
    using card-B-gt-0 unfolding ε-eq by (simp add:ln-mult)
  also have ... ≤ 7.5 * (ln (card B) + 2)
    using False card-B-gt-0 by (intro q2) auto
  also have ... ≤ k using assms(3) by simp
  finally show ?thesis by simp
qed

have ?AL = |(∫ ω. Y ω ∂(q True)) − (∫ ω. Y ω ∂(q False))|
  using exp-balls-and-bins unfolding q-def by simp
also have ... ≤ ε * card R
  by (intro exp-approx[OF assms(1) a c b])
also have ... ≤ sqrt (1 / card B) * card R
  unfolding ε-def by (intro mult-right-mono) auto
also have ... = ?AR using real-sqrt-divide by simp
finally show ?AL ≤ ?AR by simp

show ?BL ≤ ?BR
proof (cases R= {})
```

```
    case True
    then show ?thesis unfolding Y-def by simp
  next
    case False
    hence card R > 0 using fin-R by auto
    hence card-R-ge-1: real (card R) ≥ 1 by simp

    have ?BL ≤ measure-pmf.variance (q True) Y +
      |measure-pmf.variance (q True) Y − measure-pmf.variance (q False) Y|
      unfolding q-def by auto
    also have ... ≤ measure-pmf.variance (q True) Y + ε^2
      by (intro add-mono var-approx[OF assms(1) a c b]) auto
    also have ... ≤ measure-pmf.variance (q True) Y + sqrt(1 / card B)^2
      unfolding ε-def by (intro add-mono power-mono) auto
    also have ... ≤ card R * (real (card R) − 1) / card B + sqrt(1 / card B)^2
      unfolding q-altdef by (intro add-mono var-balls-and-bins) auto
    also have ... = card R * (real (card R) − 1) / card B + 1 / card B
      by (auto simp add:power-divide real-sqrt-divide)
    also have ... ≤ card R * (real (card R) − 1) / card B + card R / card B
      by (intro add-mono divide-right-mono card-R-ge-1) auto
    also have ... = (card R * (real (card R) − 1) + card R) / card B
      by argo
    also have ... = ?BR
      by (simp add:algebra-simps power2-eq-square)
    finally show ?BL ≤ ?BR by simp
  qed
qed

lemma devitation-bound:
  assumes card R ≤ card B
  assumes lim-balls-and-bins k p
  assumes real k ≥ C_2 * ln (real (card B)) + C_3
  shows measure p {ω. |Y ω − μ| > 9 * real (card R) / sqrt (real (card B))} ≤ 1 / 2^6
    (is ?L ≤ ?R)
proof (cases card R > 0)
  case True

  define k' :: nat where k' = k − 1
  have (1::real) ≤ 7.5 * 0 + 16 by simp
  also have ... ≤ 7.5 * ln (real (card B)) + 16
    using card-B-ge-1 by (intro add-mono mult-left-mono ln-ge-zero) auto
  also have ... ≤ k using assms(3) unfolding C_2-def C_3-def by simp
  finally have k-ge-1: k ≥ 1 by simp
  have lim: lim-balls-and-bins (k'+1) p
    using k-ge-1 assms(2) unfolding k'-def by simp

  have k'-min: real k' ≥ 7.5 * (ln (real (card B)) + 2)
    using k-ge-1 assms(3) unfolding C_2-def C_3-def k'-def by simp

  let ?r = real (card R)
  let ?b = real (card B)
  have a: integrable p (λω. (Y ω)^2)
    unfolding Y-def
    by (intro integrable-pmf-iff-bounded[where C=real (card R)^2])
     (auto intro!: card-image-le[OF fin-R])

  have ?L ≤ P(ω in measure-pmf p. |Y ω− (∫ ω. Y ω ∂p)| ≥ 8*?r / sqrt ?b)
  proof (rule pmf-mono)
```

fix ω **assume** $\omega \in$ *set-pmf p*
assume *a*:$\omega \in \{\omega.\ 9 * real\ (card\ R)\ /\ sqrt\ (real\ (card\ B)) < |Y\ \omega - \mu|\}$
have $8 * ?r\ /\ sqrt\ ?b = 9 * ?r\ /\ sqrt\ ?b - ?r\ /\ sqrt\ ?b$
by *simp*
also have ... $\leq\ |Y\ \omega - \mu| - |(\int \omega.\ Y\ \omega\ \partial p) - \mu|$
using *a* **by** (*intro diff-mono exp-approx-2*[*OF assms*(*1*) *lim k'-min*]) *simp*
also have ... $\leq |Y\ \omega - (\int \omega.\ Y\ \omega\ \partial p)|$
by *simp*
finally have $8 * ?r\ /\ sqrt\ ?b \leq |Y\ \omega - (\int \omega.\ Y\ \omega\ \partial p)|$ **by** *simp*
thus $\omega \in \{\omega \in space\ (measure\text{-}pmf\ p).\ 8 * ?r\ /\ sqrt\ ?b \leq |Y\ \omega - (\int \omega.\ Y\ \omega\ \partial p)|\}$
by *simp*
qed
also have ... $\leq$ *measure-pmf.variance p Y* / $(8*?r\ /\ sqrt\ ?b)\hat{}2$
using *True card-B-gt-0 a*
by (*intro measure-pmf.Chebyshev-inequality*) *auto*
also have ... $\leq (?r\hat{}2\ /\ ?b)\ /\ (8*?r\ /\ sqrt\ ?b)\hat{}2$
by (*intro divide-right-mono var-approx-2*[*OF assms*(*1*) *lim k'-min*]) *simp*
also have ... $= 1/2\hat{}6$
using *card-B-gt-0 True*
by (*simp add*:*power2-eq-square*)
finally show *?thesis* **by** *simp*
next
case *False*
hence $R = \{\}$ *card R* $= 0$ **using** *fin-R* **by** *auto*
thus *?thesis*
unfolding *Y-def* μ*-def* **by** *simp*
qed

end

unbundle *no intro-cong-syntax*

end

5 Tail Bounds for Expander Walks

theory *Distributed-Distinct-Elements-Tail-Bounds*
imports
Distributed-Distinct-Elements-Preliminary
Expander-Graphs.Pseudorandom-Objects-Expander-Walks
HOL−Decision-Procs.Approximation
begin

This section introduces tail estimates for random walks in expander graphs, specific to the verification of this algorithm (in particular to two-stage expander graph sampling and obtained tail bounds for subgaussian random variables). They follow from the more fundamental results *regular-graph.kl-chernoff-property* and *regular-graph.uniform-property* which are verified in the AFP entry for expander graphs [10].

hide-fact *Henstock-Kurzweil-Integration.integral-sum*

unbundle *intro-cong-syntax*

lemma *x-ln-x-min*:
assumes $x \geq (0::real)$
shows $x * ln\ x \geq -exp\ (-1)$
proof −
define f **where** $f\ x = x * ln\ x$ **for** $x :: real$

```
  define f' where f' x = ln x + 1 for x :: real

  have 0:(f has-real-derivative (f' x)) (at x) if x > 0 for x
    unfolding f-def f'-def using that
    by (auto intro!: derivative-eq-intros)

  have f' x ≥ 0 if exp (−1) ≤ x for x :: real
  proof −
    have ln x ≥ −1
      using that order-less-le-trans[OF exp-gt-zero]
      by (intro iffD2[OF ln-ge-iff]) auto
    thus ?thesis
      unfolding f'-def by (simp)
  qed

  hence ∃ y. (f has-real-derivative y) (at x) ∧ 0 ≤ y if x ≥ exp (−1) for x :: real
    using that order-less-le-trans[OF exp-gt-zero]
    by (intro exI[where x=f' x] conjI 0) auto
  hence f (exp (−1)) ≤ f x if exp(−1) ≤ x
    by (intro DERIV-nonneg-imp-nondecreasing[OF that]) auto
  hence 2:?thesis if  exp(−1) ≤ x
    unfolding f-def using that by simp

  have f' x ≤ 0 if x > 0 x ≤ exp (−1) for x :: real
  proof −
    have ln x ≤ ln (exp (−1))
      by (intro iffD2[OF ln-le-cancel-iff] that exp-gt-zero)
    also have ... = −1
      by simp
    finally have ln x ≤ −1 by simp
    thus ?thesis unfolding f'-def by simp
  qed

  hence ∃ y. (f has-real-derivative y) (at x) ∧ y ≤ 0 if x > 0  x ≤ exp (−1) for x :: real
    using that by (intro exI[where x=f' x] conjI 0) auto
  hence f (exp (−1)) ≤ f x if x > 0 x ≤ exp(−1)
    using that(1) by (intro DERIV-nonpos-imp-nonincreasing[OF that(2)]) auto
  hence 3:?thesis if x > 0 x ≤ exp(−1)
    unfolding f-def using that by simp

  have ?thesis if x = 0
    using that by simp
  thus ?thesis
    using 2 3 assms by fastforce
qed

theorem (in regular-graph) walk-tail-bound:
  assumes l > 0
  assumes S ⊆ verts G
  defines μ ≡ real (card S) / card (verts G)
  assumes γ < 1 μ + Λ_a ≤ γ
  shows measure (pmf-of-multiset (walks G l)) {w. real (card {i ∈ {..<l}. w ! i ∈ S}) ≥ γ*l}
    ≤ exp (− real l * (γ * ln (1/(μ+Λ_a)) − 2 * exp(−1))) (is ?L ≤ ?R)
proof (cases μ > 0)
  case True

  have 0 < μ + Λ_a
    by (intro add-pos-nonneg Λ-ge-0 True)
```

also have ... $\leq \gamma$
  using *assms*(*5*) **by** *simp*
finally have γ-*gt-0*: *0* $< \gamma$ **by** *simp*

hence γ-*ge-0*: *0* $\leq \gamma$
  by *simp*

have *card S* $\leq$ *card* (*verts G*)
  by (*intro card-mono assms*(*2*)) *auto*
hence μ-*le-1*: $\mu \leq$ *1*
  unfolding μ-*def* **by** (*simp add*:*divide-simps*)

have *2*: *0* $< \mu + \Lambda_a * (1 - \mu)$
  using μ-*le-1* **by** (*intro add-pos-nonneg True mult-nonneg-nonneg* Λ-*ge-0*) *auto*

have $\mu + \Lambda_a * (1 - \mu) \leq \mu + \ \Lambda_a * 1$
  using Λ-*ge-0 True* **by** (*intro add-mono mult-left-mono*) *auto*
also have ... $\leq \gamma$
  using *assms*(*5*) **by** *simp*
also have ... $<$ *1*
  using *assms*(*4*) **by** *simp*
finally have *4*:$\mu + \Lambda_a * (1 - \mu) <$ *1* **by** *simp*
hence *3*: *1* $\leq$ *1* / (*1* $- (\mu + \Lambda_a * (1 - \mu)))$
  using *2* **by** (*subst pos-le-divide-eq*) *simp-all*

have *card S* $\leq$ *n*
  unfolding *n-def* **by** (*intro card-mono assms*(*2*)) *auto*
hence *0*:$\mu \leq$ *1*
  unfolding μ-*def n-def*[*symmetric*] **using** *n-gt-0* **by** *simp*

have $\gamma *$ *ln* (*1* / $(\mu + \Lambda_a)) - 2*$*exp* $(-$ *1*$) = \gamma *$ *ln* (*1* / $(\mu + \Lambda_a*1))+0 - 2*$*exp* $(-$ *1*$)$
  by *simp*
also have ... $\leq \gamma *$ *ln* (*1* / $(\mu + \Lambda_a*(1-\mu)))+0-2*$*exp*$(-1)$
  using *True* γ-*ge-0* Λ-*ge-0 0 2*
  by (*intro diff-right-mono mult-left-mono iffD2*[*OF ln-le-cancel-iff*] *divide-pos-pos*
    divide-left-mono add-mono) *auto*
also have ... $\leq \gamma *$ *ln* (*1* / $(\mu + \Lambda_a*(1-\mu)))+(1-\gamma)*ln(1/(1-(\mu+\Lambda_a*(1-\mu))))-2*$ *exp*(-1)
  using *assms*(*4*) *3* **by** (*intro add-mono diff-mono mult-nonneg-nonneg ln-ge-zero*) *auto*
also have ... $= (-exp(-1))+\gamma*ln(1/(\mu+\Lambda_a*(1-\mu)))+(-exp(-1))+(1-\gamma)*ln(1/(1-(\mu+\Lambda_a*(1-\mu))))$
  by *simp*
also have ... $\leq \gamma*ln\ \gamma+\gamma*ln(1/(\mu+\Lambda_a*(1-\mu)))+(1-\gamma)*ln(1-\gamma)+(1-\gamma)*ln(1/(1-(\mu+\Lambda_a*(1-\mu))))$
  using *assms*(*4*) γ-*ge-0* **by** (*intro add-mono x-ln-x-min*) *auto*
also have ... $= \gamma*(ln\ \gamma+ln(1/(\mu+\Lambda_a*(1-\mu))))+(1-\gamma)*(ln(1-\gamma)+ln(1/(1-(\mu+\Lambda_a*(1-\mu)))))$
  by (*simp add*:*algebra-simps*)
also have ... $= \gamma *$ *ln* $(\gamma*(1/(\mu+\Lambda_a*(1-\mu))))+(1-\gamma)*ln((1-\gamma)*(1/(1-(\mu+\Lambda_a*(1-\mu)))))$
  using *2 4* **by** (*simp add*: *ln-mult ln-div*)
also have ... $=$ *KL-div* γ $(\mu+\Lambda_a*(1-\mu))$
  unfolding *KL-div-def* **by** *simp*
finally have *1*: $\gamma *$ *ln* (*1* / $(\mu + \Lambda_a)) - 2 *$ *exp* $(-$ *1*$) \leq$ *KL-div* γ $(\mu + \Lambda_a * (1 - \mu))$
  by *simp*

have $\mu+\Lambda_a*(1-\mu) \leq \mu+\Lambda_a*1$
  using *True*
  by (*intro add-mono mult-left-mono* Λ-*ge-0*) *auto*
also have ... $\leq \gamma$
  using *assms*(*5*) **by** *simp*
finally have $\mu+\Lambda_a*(1-\mu) \leq \gamma$ **by** *simp*
moreover have $\mu+\Lambda_a*(1-\mu) >$ *0*

```
    using 0 by (intro add-pos-nonneg True mult-nonneg-nonneg Λ-ge-0) auto
  ultimately have μ+Λa*(1−μ) ∈ {0<..γ} by simp
  hence ?L ≤ exp (− real l * KL-div γ (μ+Λa*(1−μ)))
    using assms(4) unfolding μ-def by (intro kl-chernoff-property assms(1,2)) auto
  also have ... ≤ ?R
    using assms(1) 1 by simp
  finally show ?thesis by simp
next
  case False
  hence μ ≤ 0 by simp
  hence card S = 0
    unfolding μ-def n-def[symmetric] using n-gt-0 by (simp add:divide-simps)
  moreover have finite S
    using finite-subset[OF assms(2) finite-verts] by auto
  ultimately have 0:S = {} by auto
  have μ = 0
    unfolding μ-def 0 by simp
  hence μ + Λa ≥0
    using Λ-ge-0 by simp
  hence γ ≥ 0
    using assms(5) by simp
  hence γ * real l ≥ 0
    by (intro mult-nonneg-nonneg) auto
  thus ?thesis using 0 by simp
qed

theorem (in regular-graph) walk-tail-bound-2:
  assumes l > 0 Λa ≤ Λ Λ > 0
  assumes S ⊆ verts G
  defines μ ≡ real (card S) / card (verts G)
  assumes γ < 1 μ + Λ ≤ γ
  shows measure (pmf-of-multiset (walks G l)) {w. real (card {i ∈ {..<l}. w ! i ∈ S}) ≥ γ*l}
    ≤ exp (− real l * (γ * ln (1/(μ+Λ)) − 2 * exp(−1))) (is ?L ≤ ?R)
proof (cases μ > 0)
  case True

  have 0: 0 < μ + Λa
    by (intro add-pos-nonneg Λ-ge-0 True)
  hence 0 < μ + Λ
    using assms(2) by simp
  hence 1: 0 < (μ + Λ) * (μ + Λa)
    using 0 by simp

  have 3:μ + Λa ≤ γ
    using assms(2,7) by simp
  have 2: 0 ≤ γ
    using 3 True Λ-ge-0 by simp

  have ?L ≤ exp (− real l * (γ * ln (1/(μ+Λa)) − 2 * exp(−1)))
    using 3 unfolding μ-def by (intro walk-tail-bound assms(1,4,6))
  also have ... = exp (− (real l * (γ * ln (1/(μ+Λa)) − 2 * exp(−1))))
    by simp
  also have ... ≤ exp (− (real l * (γ * ln (1/(μ+Λ)) − 2 * exp(−1))))
    using True assms(2,3) using 0 1 2
    by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono diff-mono iffD2[OF ln-le-cancel-iff]
        divide-left-mono le-imp-neg-le) simp-all
  also have ... = ?R
    by simp
```

```
  finally show ?thesis by simp
next
  case False
  hence μ ≤ 0 by simp
  hence card S = 0
    unfolding μ-def n-def[symmetric] using n-gt-0 by (simp add:divide-simps)
  moreover have finite S
    using finite-subset[OF assms(4) finite-verts] by auto
  ultimately have 0:S = {} by auto
  have μ = 0
    unfolding μ-def 0 by simp
  hence μ + Λ_a ≥0
    using Λ-ge-0 by simp
  hence γ ≥ 0
    using assms by simp
  hence γ * real l ≥ 0
    by (intro mult-nonneg-nonneg) auto
  thus ?thesis using 0 by simp
qed

lemma disjI-safe: (¬x ⟹ y) ⟹ x ∨ y by auto

lemma walk-tail-bound:
  fixes T
  assumes l > 0 Λ > 0
  assumes measure (sample-pro S) {w. T w} ≤ μ
  assumes γ ≤ 1 μ + Λ ≤ γ μ ≤ 1
  shows measure (sample-pro (ℰ l Λ S)) {w. real (card {i ∈ {..<l}. T (w i)}) ≥ γ*l}
    ≤ exp (− real l * (γ * ln (1/(μ+Λ)) − 2 * exp(−1))) (is ?L ≤ ?R)
proof −
  have μ-ge-0: μ ≥ 0 using assms(3) measure-nonneg order.trans by metis
  hence γ-gt-0: γ > 0 using assms(2,5) by auto
  hence γ-ge-0: γ ≥ 0 by simp

  have μ + Λ * (1 − μ) ≤ μ + Λ * 1 using assms(2,6) μ-ge-0 by auto
  also have ... ≤ γ using assms(5) by simp
  finally have 1:μ + Λ * (1 − μ) ≤ γ by simp

  have 2: 0 < μ + Λ * (1 − μ)
  proof (cases μ = 1)
    case True then show ?thesis by simp
  next
    case False
    then show ?thesis  using assms(2,6)
      by (intro add-nonneg-pos μ-ge-0 linordered-semiring-strict-class.mult-pos-pos) auto
  qed

  have 3: 0 < μ + Λ using μ-ge-0 assms(2) by simp

  have γ * ln (1 / (μ + Λ)) − 2*exp (− 1) = γ * ln (1 / (μ + Λ*1))+0 −2*exp (−1) by simp
  also have ... ≤ γ * ln (1 / (μ + Λ*(1−μ)))+0−2*exp(−1)
    using 2 3 γ-ge-0 μ-ge-0 assms(2) by (intro diff-right-mono add-mono mult-left-mono
      iffD2[OF ln-le-cancel-iff] divide-left-mono divide-pos-pos) simp-all
  also have ... ≤ γ * ln (1 / (μ + Λ*(1−μ)))+(1−γ)*ln(1/(1−(μ+Λ*(1−μ))))−2* exp(−1)
  proof (cases γ < 1)
    case True
    hence μ + Λ * (1 − μ) < 1 using 1 by simp
    thus ?thesis using assms(4) 2
```

```
    by (intro diff-right-mono add-mono mult-nonneg-nonneg order.refl ln-ge-zero) auto
  next
    case False
    hence γ=1 using assms(4) by simp
    thus ?thesis by simp
  qed
 also have ... = (−exp(−1))+γ∗ln(1/(μ+Λ∗(1−μ)))+(−exp(−1))+(1−γ)∗ln(1/(1−(μ+Λ∗(1−μ))))
    by simp
 also have ... ≤ γ∗ln γ+γ∗ln(1/(μ+Λ∗(1−μ)))+(1−γ)∗ln(1−γ)+(1−γ)∗ln(1/(1−(μ+Λ∗(1−μ))))
    using assms(4) γ-ge-0 by (intro add-mono x-ln-x-min) auto
 also have ... = γ∗(ln γ+ln(1/(μ+Λ∗(1−μ))))+(1−γ)∗(ln(1−γ)+ln(1/(1−(μ+Λ∗(1−μ)))))
    by (simp add:algebra-simps)
 also have ... = γ ∗ ln (γ∗(1/(μ+Λ∗(1−μ))))+(1−γ)∗ln((1−γ)∗(1/(1−(μ+Λ∗(1−μ)))))
    using 2 1 assms(4)  by (simp add: ln-mult ln-div)
 also have ... = KL-div γ (μ+Λ∗(1−μ)) unfolding KL-div-def by simp
 finally have 4: γ ∗ ln (1 / (μ + Λ)) − 2 ∗ exp (− 1) ≤ KL-div γ (μ + Λ ∗ (1 − μ))
    by simp

 have ?L ≤ exp (− real l ∗ KL-div γ (μ+Λ∗(1−μ)))
    using 1 by (intro expander-kl-chernoff-bound assms)
 also have ... ≤ exp (− real l ∗ (γ ∗ ln (1 / (μ + Λ)) − 2 ∗ exp (− 1)))
    by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono-neg 4) auto
 finally show ?thesis by simp
qed

definition C_1 :: real where C_1 = exp 2 + exp 3 + (exp 1 − 1)

lemma deviation-bound:
 fixes f :: 'a ⇒ real
 assumes l > 0
 assumes Λ ∈ {0<..exp (−real l ∗ ln (real l)^3)}
 assumes ⋀x. x ≥ 20 ⟹ measure (sample-pro S) {v. f v ≥ x} ≤ exp (−x ∗ ln x^3)
 shows measure (sample-pro (ℰ l Λ S)) {ω. (∑ i<l. f (ω i)) ≥ C_1 ∗ l} ≤ exp (− real l)  (is ?L
≤ ?R)
proof −
 let ?w = sample-pro (ℰ l Λ S)
 let ?p = sample-pro S
 let ?a = real l∗(exp 2 + exp 3)

 define b :: real where b = exp 1 − 1
 have b-gt-0: b > 0 unfolding b-def by (approximation 5)

 define L where
   L k = measure ?w {w. exp (real k)∗card{i∈{..<l}.f(w i)≥exp(real k)} ≥ real l/real k^2} for k

 define k-max where k-max = max 4 (MAX v ∈ pro-set S. nat ⌊ln (f v)⌋+1)

 have k-max-ge-4: k-max ≥ 4 unfolding k-max-def by simp
 have k-max-ge-3: k-max ≥ 3 unfolding k-max-def by simp

  have 1:of-bool(⌊ln(max x (exp 1))⌋+1=int k)=(of-bool(x≥exp(real k−1))−of-bool(x ≥ exp
k)::real)
   (is ?L1 = ?R1) if k ≥ 3 for k x
 proof −
   have a1: real k − 1 ≤ k by simp
   have ?L1 = of-bool(⌊ln(max x (exp 1))⌋=int k−1) by simp
  also have ... = of-bool(ln(max x (exp 1))∈{real k−1..<real k}) unfolding floor-eq-iff by simp
   also have ... = of-bool(exp(ln(max x (exp 1)))∈{exp (real k−1)..<exp (real k)}) by simp
```

```
  also have ... = of-bool(max x (exp 1) ∈{exp (real k−1)..<exp (real k)})
    by (subst exp-ln) (auto intro!:max.strict-coboundedI2)
  also have ... = of-bool(x ∈{exp (real k−1)..<exp (real k)})
  proof (cases x ≥ exp 1)
    case True
    then show ?thesis by simp
  next
    case False
    have {exp (real k − 1)..<exp (real k)} ⊆ {exp (real k − 1)..} by auto
    also have ... ⊆ {exp 1..} using that by simp
    finally have {exp (real k − 1)..<exp (real k)} ⊆ {exp 1..} by simp
    moreover have x ∉ {exp 1..} using False by simp
    ultimately have x ∉ {exp (real k − 1)..<exp (real k)} by blast
    hence of-bool(x ∈{exp (real k−1)..<exp (real k)}) = 0 by simp
    also have ... = of-bool(max x (exp 1) ∈{exp (real k−1)..<exp (real k)})
      using False that by simp
    finally show ?thesis by metis
  qed
  also have ... = ?R1 using order-trans[OF iffD2[OF exp-le-cancel-iff a1]] by auto
  finally show ?thesis by simp
qed

have 0: {nat ⌊ln (max (f x) (exp 1))⌋+1} ⊆ {2..k-max} (is {?L1} ⊆ ?R2)
  if x ∈ pro-set S for x
proof (cases f x ≥ exp 1)
  case True
  hence ?L1 = nat ⌊ln (f x)⌋+1 by simp
  also have ... ≤ (MAX v ∈ pro-set S. nat ⌊ln (f v)⌋+1)
    by (intro Max-ge finite-imageI imageI that finite-pro-set)
  also have ... ≤ k-max unfolding k-max-def by simp
  finally have le-0: ?L1 ≤ k-max by simp
  have (1::nat) ≤ nat ⌊ln (exp (1::real))⌋ by simp
  also have ... ≤ nat ⌊ln (f x)⌋
    using True order-less-le-trans[OF exp-gt-zero]
    by (intro nat-mono floor-mono iffD2[OF ln-le-cancel-iff]) auto
  finally have 1 ≤ nat ⌊ln (f x)⌋ by simp
  hence ?L1 ≥ 2 using True by simp
  hence ?L1 ∈ ?R2 using le-0 by simp
  then show ?thesis by simp
next
  case False
  hence {?L1} = {2} by simp
  also have ... ⊆ ?R2 using k-max-ge-3 by simp
  finally show ?thesis by simp
qed

have 2:(∑ i<l. f (w i)) ≤ ?a+b*(∑ k=3..<k-max. exp k * card {i∈{..<l}. f (w i)≥exp k})
  (is ?L1 ≤ ?R1) if w ∈ pro-set (ℰ l Λ S) for w
proof −
  have s-w: w i ∈ pro-set S for i
    using that expander-pro-range[OF assms(1)] assms(2)
    unfolding set-sample-pro[where S=ℰ l Λ S] by auto

  have ?L1 ≤ (∑ i<l. exp( ln (max (f (w i)) (exp 1))))
    by (intro sum-mono) (simp add:less-max-iff-disj)
  also have ... ≤ (∑ i<l. exp (of-nat (nat ⌊ln (max (f (w i)) (exp 1))⌋+1)))
    by (intro sum-mono iffD2[OF exp-le-cancel-iff]) linarith
   also have ... = (∑ i<l. (∑ k=2..k-max. exp k * of-bool (k=nat ⌊ln (max (f (w i))(exp
```

1))⌊+1)))
using *Int-absorb1*[*OF 0*] *s-w* **by** (*intro sum.cong map-cong refl*)
(*simp add*:*of-bool-def if-distrib if-distribR sum.If-cases*)
also have ...=
(∑ *i*<*l*.(∑ *k*∈(*insert 2*{*3..k-max*}). *exp k*∗ *of-bool*(*k*=*nat*⌊*ln*(*max*(*f* (*w i*))(*exp 1*))⌋+*1*)))
using *k-max-ge-3* **by** (*intro-cong* [*σ*$_1$ *sum-list*] *more*:*map-cong sum.cong*) *auto*
also have ... = (∑ *i*<*l*. *exp 2*∗ *of-bool* (*2*=*nat* ⌊*ln* (*max* (*f* (*w i*))(*exp 1*))⌋+*1*) +
(∑ *k*=*3..k-max*. *exp k* ∗ *of-bool* (*k*=*nat* ⌊*ln* (*max* (*f* (*w i*))(*exp 1*))⌋+*1*)))
by (*subst sum.insert*) *auto*
also have ...≤(∑ *i*<*l*. *exp 2*∗*1*+(∑ *k*=*3..k-max*. *exp k*∗ *of-bool*(*k*=*nat*⌊*ln*(*max*(*f* (*w i*))(*exp 1*))⌋+*1*)))
by (*intro sum-mono add-mono mult-left-mono*) *auto*
also have ...=(∑ *i*<*l*. *exp 2*+(∑ *k*=*3..k-max*. *exp k*∗ *of-bool*(⌊*ln*(*max*(*f* (*w i*))(*exp 1*))⌋+*1*=*int k*)))
by (*intro-cong* [*σ*$_1$ *sum-list*,*σ*$_1$ *of-bool*, *σ*$_2$(+),*σ*$_2$(∗)] *more*:*map-cong sum.cong*) *auto*
also have ... =
(∑ *i*<*l*. *exp 2*+(∑ *k*=*3..k-max*. *exp k*∗(*of-bool*(*f* (*w i*)≥*exp* (*real k*−*1*))−*of-bool*(*f* (*w i*)≥*exp k*))))
by (*intro-cong* [*σ*$_1$ *sum-list*,*σ*$_1$ *of-bool*, *σ*$_2$(+),*σ*$_2$(∗)] *more*:*map-cong sum.cong 1*) *auto*
also have ... = (∑ *i*<*l*.
exp 2+(∑ *k*=*2*+*1..*<*k-max*+*1*. *exp k*∗(*of-bool*(*f* (*w i*)≥*exp*(*real k*−*1*))−*of-bool*(*f* (*w i*)≥*exp k*))))
by (*intro-cong* [*σ*$_2$(+)] *more*:*map-cong sum.cong*) *auto*
also have ... = (∑ *i*<*l*.
exp 2+(∑ *k*=*2..*<*k-max*. *exp* (*k*+*1*)∗(*of-bool*(*f* (*w i*)≥*exp k*)−*of-bool*(*f* (*w i*)≥*exp* (*Suc k*)))))
by (*subst sum.shift-bounds-nat-ivl*) *simp*
also have ... = (∑ *i*<*l*. *exp 2*+ (∑ *k*=*2..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(∑ *k*=*2..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp* (*k*+*1*))))
by (*simp add*:*sum-subtractf algebra-simps*)
also have ... = (∑ *i*<*l*. *exp 2*+ (∑ *k*=*2..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(∑ *k*=*3..*<*k-max*+*1*. *exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*)))
by (*subst sum.shift-bounds-nat-ivl*[*symmetric*]) (*simp cong*:*sum.cong*)
also have ... = (∑ *i*<*l*. *exp 2*+ (∑ *k*∈ *insert 2* {*3..*<*k-max*}. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(∑ *k*=*3..*<*k-max*+*1*. *exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*)))
using *k-max-ge-3* **by** (*intro-cong* [*σ*$_2$ (+), *σ*$_2$ (−)] *more*: *map-cong sum.cong*) *auto*
also have ... = (∑ *i*<*l*. *exp 2*+ *exp 3* ∗ *of-bool* (*f* (*w i*) ≥ *exp 2*) +
(∑ *k*=*3..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*)) −
(∑ *k*=*3..*<*k-max*+*1*. *exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*)))
by (*subst sum.insert*) (*simp-all add*:*algebra-simps*)
also have ... ≤ (∑ *i*<*l*. *exp 2*+*exp 3*+(∑ *k*=*3..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(∑ *k*=*3..*<*k-max*+*1*. *exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*)))
by (*intro sum-mono add-mono diff-mono*) *auto*
also have ... = (∑ *i*<*l*. *exp 2*+*exp 3*+(∑ *k*=*3..*<*k-max*. *exp* (*k*+*1*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(∑ *k*∈ *insert k-max* {*3..*<*k-max*}. *exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*)))
using *k-max-ge-3* **by** (*intro-cong* [*σ*$_2$ (+), *σ*$_2$ (−)] *more*: *map-cong sum.cong*) *auto*
also have ... = (∑ *i*<*l*. *exp 2*+*exp 3*+(∑ *k*=*3..*<*k-max*. (*exp* (*k*+*1*)−*exp k*)∗ *of-bool*(*f* (*w i*)≥*exp k*))−
(*exp k-max* ∗ *of-bool* (*f* (*w i*)≥ *exp k-max*)))
by (*subst sum.insert*) (*auto simp add*:*sum-subtractf algebra-simps*)
also have ...≤(∑ *i*<*l*. *exp 2*+*exp 3*+(∑ *k*=*3..*<*k-max*. (*exp* (*k*+*1*)−*exp k*)∗ *of-bool*(*f*(*w i*)≥*exp k*))−*0*)
by (*intro sum-mono add-mono diff-mono*) *auto*
also have ... ≤(∑ *i*<*l*. *exp 2*+*exp 3*+ (∑ *k*=*3..*<*k-max*. (*exp* (*k*+*1*)−*exp k*)∗ *of-bool*(*f*(*w i*)≥*exp k*)))
by *auto*
also have ... = (∑ *i*<*l*. *exp 2*+*exp 3*+(∑ *k*=*3..*<*k-max*.(*exp 1*−*1*)∗(*exp k*∗ *of-bool*(*f* (*w i*)≥*exp k*))))

```
    by (simp add:exp-add algebra-simps)
  also have ... = (∑ i<l. exp 2+exp 3+b*(∑ k=3..<k-max. exp k* of-bool(f (w i)≥exp k)))
    unfolding b-def by (subst sum-distrib-left) simp
  also have ... = ?a+b*(∑ i<l. (∑ k=3..<k-max. exp k* of-bool(f (w i)≥exp k)))
     by (simp add: sum-distrib-left[symmetric])
  also have ... = ?R1
    by (subst sum.swap) (simp add:ac-simps Int-def)
  finally show ?thesis by simp
qed

have 3: ∃ k∈{3..<k-max}. g k ≥ l/real k^2 if (∑ k=3..<k-max. g k) ≥ real l for g
proof (rule ccontr)
  assume a3: ¬(∃ k∈{3..<k-max}. g k ≥ l/real k^2)
  hence g k < l/real k^2 if k ∈{3..<k-max} for k using that by force
  hence (∑ k=3..<k-max. g k) < (∑ k=3..<k-max. l/real k^2)
    using k-max-ge-4 by (intro sum-strict-mono) auto
  also have ... ≤ (∑ k=3..<k-max. l/ (real k*(real k-1)))
    by (intro sum-mono divide-left-mono) (auto simp:power2-eq-square)
  also have ... = l * (∑ k=3..<k-max. 1 / (real k-1) - 1/k)
    by (simp add:sum-distrib-left field-simps)
  also have ... = l * (∑ k=2+1..<(k-max-1)+1. (-1)/k - (-1) / (real k-1))
    by (intro sum.cong arg-cong2[where f=(*)]) auto
  also have ... = l * (∑ k=2..<(k-max-1). (-1)/(Suc k) - (-1) / k)
    by (subst sum.shift-bounds-nat-ivl) auto
  also have ... = l * (1/2 - 1 / real (k-max - 1))
    using k-max-ge-3 by (subst sum-Suc-diff') auto
  also have ... ≤ real l * (1 - 0) by (intro mult-left-mono diff-mono) auto
  also have ... = l by simp
  finally have (∑ k=3..<k-max. g k) < l by simp
  thus False using that by simp
qed

have 4: L k ≤ exp(-real l-k+2) if k ≥ 3 for k
proof (cases k ≤ ln l)
  case True
  define γ where γ = 1 / (real k)^2 / exp (real k)
  define μ where μ = exp (-exp(real k) * real k^3)

  have exp-k-ubound: exp (real k) ≤ real l using True assms(1) by (simp add: ln-ge-iff)

  have 20 ≤ exp (3::real) by (approximation 10)
  also have ... ≤ exp (real k) using that by simp
  finally have exp-k-lbound: 20 ≤ exp (real k) by simp

  have measure (sample-pro S) {v. f v≥exp(real k)} ≤ exp (-exp(real k) * ln (exp (real k)) ^ 3)
    by (intro assms(3) exp-k-lbound)
  also have ... = exp (-(exp(real k) * real k^3)) by simp
 finally have μ-bound: measure (sample-pro S) {v. f v ≥ exp (real k)} ≤ μ by (simp add:μ-def)

  have μ+Λ ≤ exp (-exp(real k) * real k^3) + exp (- real l * ln (real l) ^ 3)
    unfolding μ-def using assms by (intro add-mono) auto
  also have ... =  exp (-(exp(real k) * real k^3)) + exp (- (real l * ln (real l) ^ 3)) by simp
  also have ... ≤ exp (-(exp(real k) * real k^3)) + exp (-(exp(real k) * ln(exp (real k))^3))
    using assms(1) exp-k-ubound by (intro add-mono iffD2[OF exp-le-cancel-iff] le-imp-neg-le
        mult-mono power-mono iffD2[OF ln-le-cancel-iff]) simp-all
  also have ... = 2 * exp (-exp(real k) * real k^3) by simp
  finally have μ-Λ-bound: μ+Λ ≤ 2 * exp (-exp(real k) * real k^3) by simp
```

have $\mu{+}\Lambda \leq 2 * exp\ (-exp(real\ k) * real\ k\hat{}3)$ **by** (*intro* μ-Λ-*bound*)
also have ... = $exp\ (-exp(real\ k) * real\ k\hat{}3 + ln\ 2)$ **unfolding** *exp-add* **by** *simp*
also have ... = $exp\ (-(exp(real\ k) * real\ k\hat{}3 - ln\ 2))$ **by** *simp*
also have ... ≤ $exp\ (-((1{+}\ real\ k) * real\ k\hat{}3 - ln\ 2))$
using *that* **by** (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le diff-mono mult-right-mono exp-ge-add-one-self-aux*) *auto*
also have ... = $exp\ (-(real\ k\hat{}4 + (real\ k\hat{}3{-}\ ln\ 2)))$
by (*simp add*:*power4-eq-xxxx power3-eq-cube algebra-simps*)
also have ... ≤ $exp\ (-(real\ k\hat{}4 + (2\hat{}3{-}\ ln\ 2)))$ **using** *that*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le add-mono diff-mono power-mono*) *auto*
also have ... ≤ $exp\ (-(real\ k\hat{}4 + 0))$
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le add-mono order.refl*) (*approximation 5*)
also have ... ≤ $exp\ (-(real\ k\hat{}3 * real\ k))$
by (*simp add*:*power4-eq-xxxx power3-eq-cube algebra-simps*)
also have ... ≤ $exp\ (-(2\hat{}3 * real\ k))$ **using** *that*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le mult-right-mono power-mono*) *auto*
also have ... ≤ $exp\ (-3{*}\ real\ k)$ **by** (*intro iffD2*[*OF exp-le-cancel-iff*]) *auto*
also have ... = $exp\ (-(real\ k + 2 * real\ k)\ )$ **by** *simp*
also have ... ≤ $exp\ (-(real\ k + 2 * ln\ k)\ )$
using *that*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le add-mono mult-left-mono ln-bound*) *auto*
also have ... = $exp\ (-(real\ k + ln(k\hat{}2)))$ **using** *that* **by** (*subst ln-powr*[*symmetric*]) *auto*
also have ... = γ
using *that* **unfolding** γ-*def exp-minus exp-add inverse-eq-divide* **by** (*simp add*:*algebra-simps*)
finally have μ-Λ-*le*-γ: $\mu{+}\Lambda{\leq}\gamma$ **by** *simp*

have $\mu \geq 0$ **unfolding** μ-*def* **by** *simp*
hence μ-Λ-*gt-0*: $\mu{+}\Lambda{>}0$ **using** *assms*(*2*) **by** *auto*

have $\gamma = 1\ /\ ((real\ k)^2 * exp\ (real\ k))$ **unfolding** γ-*def* **by** *simp*
also have ... ≤ $1\ /\ (2\hat{}2 * exp\ 2)$
using *that* **by** (*intro divide-left-mono mult-mono power-mono*) (*auto*)
finally have γ-*ubound*: $\gamma \leq 1\ /\ (4 * exp\ 2)$ **by** *simp*

have $\gamma \leq 1\ /\ (4 * exp\ 2)$ **by** (*intro* γ-*ubound*)
also have ... < *1* **by** (*approximation 5*)
finally have γ-*lt-1*: $\gamma < 1$ **by** *simp*

have γ-*ge-0*: $\gamma \geq 0$ **using** *that* **unfolding** γ-*def* **by** (*intro divide-nonneg-pos*) *auto*
have μ-*le-1*: $\mu \leq 1$ **unfolding** μ-*def* **by** *simp*

have $L\ k = measure\ ?w\ \{w.\ \gamma{*}l \leq real\ (card\ \{i \in \{..{<}l\}.\ exp\ (real\ k) \leq f\ (w\ i)\})\}$
unfolding *L-def* γ-*def* **using** *that*
by (*intro-cong* [σ_2 *measure*] *more*:*Collect-cong*) (*simp add*:*field-simps*)
also have ... ≤ $exp\ (-\ real\ l * (\gamma * ln\ (1/(\mu{+}\Lambda)) - 2 * exp(-1)))$
using γ-*lt-1 assms*(*2*) **by** (*intro walk-tail-bound* μ-*bound assms*(*1*) μ-Λ-*le*-γ μ-*le-1*) *auto*
also have ... = $exp\ (\ real\ l * (\gamma * ln\ (\mu{+}\Lambda) + 2 * exp\ (-1)))$
using μ-Λ-*gt-0* **by** (*simp-all add*:*ln-div algebra-simps*)
also have ... ≤ $exp\ (\ real\ l * (\gamma * ln\ (2 * exp\ (-exp(real\ k) * real\ k\hat{}3)) + 2 * exp(-1)))$
using μ-Λ-*gt-0* μ-Λ-*bound* γ-*ge-0*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *mult-left-mono add-mono iffD2*[*OF ln-le-cancel-iff*])
simp-all
also have ... = $exp\ (real\ l * (\gamma * (ln\ 2 - exp\ (real\ k) * real\ k\ \hat{}\ 3) + 2 * exp\ (-\ 1)))$
by (*simp add*:*ln-mult*)
also have ... = $exp\ (real\ l * (\gamma * ln\ 2 - real\ k + 2 * exp\ (-\ 1)))$
using *that* **unfolding** γ-*def* **by** (*simp add*:*field-simps power2-eq-square power3-eq-cube*)
also have ... ≤ $exp\ (real\ l * (ln\ 2\ /\ (4 * exp\ 2) - real\ k + 2 * exp\ (-1)))$
using γ-*ubound* **by** (*intro iffD2*[*OF exp-le-cancel-iff*] *mult-left-mono add-mono diff-mono*)

```
    (auto simp:divide-simps)
  also have ... = exp (real l * (ln 2 / (4 * exp 2) + 2 *exp(-1) - real k))
    by simp
  also have ... ≤ exp (real l * (1 - real k))
    by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono diff-mono order.refl of-nat-0-le-iff)
     (approximation 12)
  also have ... ≤ exp (-real l - real k + 2)
  proof (intro iffD2[OF exp-le-cancel-iff])
    have 1 * (real k-2) ≤ real l * (real k-2)
      using assms(1) that by (intro mult-right-mono) auto
    thus real l * (1 - real k) ≤ - real l - real k + 2 by argo
  qed
  finally show ?thesis by simp
next
  case False
  hence k-gt-l: k ≥ ln l by simp
  define γ where γ = 1 / (real k)^2 / exp (real k)

  have 20 ≤ exp (3::real) by (approximation 10)
  also have ... ≤ exp (real k) using that by simp
  finally have exp-k-lbound: 20 ≤ exp (real k) by simp

  have γ-gt-0: 0 < γ using that unfolding γ-def by (intro divide-pos-pos) auto

  hence γ-l-gt-0: 0 < γ * real l using assms(1) by auto

  have L k = measure ?w {w. γ*l ≤ real (card {i ∈ {..<l}. exp (real k) ≤ f (w i)})}
    unfolding L-def γ-def using that
    by (intro-cong [σ2 measure] more:Collect-cong) (simp add:field-simps)
  also have ... ≤ (∫ w. real (card {i ∈ {..<l}. exp (real k) ≤ f (w i)}) ∂?w) / (γ*l)
    by (intro pmf-markov γ-l-gt-0) simp-all
  also have ... = (∫ w. (∑ i<l. of-bool (exp(real k) ≤ f (w i)))∂?w) / (γ*l)
    by (intro-cong [σ2 (/)] more:integral-cong-AE AE-pmfI) (auto simp add:Int-def)
  also have ... = (∑ i<l. (∫ w. of-bool (exp(real k) ≤ f (w i))∂?w)) / (γ*l)
    by (intro-cong [σ2 (/)] more:integral-sum integrable-measure-pmf-finite finite-pro-set)
  also have ... = (∑ i<l. (∫ v. of-bool (exp(real k) ≤ f v)∂(map-pmf (λw. w i) ?w))) / (γ*l)
    by simp
  also have ... = (∑ i<l. (∫ v. of-bool (exp(real k) ≤ f v)∂?p)) / (γ*l) using assms(1,2)
  by (intro-cong [σ2(/),σ2(integral^L),σ1 measure-pmf] more:sum.cong expander-uniform-property)
     simp-all
  also have ... = (∑ i<l. (∫ v. indicat-real {v. (exp(real k) ≤ f v)} v∂?p)) / (γ*l)
    by (intro-cong [σ2(/),σ2(integral^L)] more:sum.cong) auto
  also have ... = (∑ i<l. (measure ?p {v. f v ≥ exp (real k)})) / (γ*l) by simp
  also have ... ≤ (∑ i<l. exp (- exp (real k) * ln (exp (real k)) ^ 3)) / (γ*l)
    using γ-l-gt-0 by (intro divide-right-mono sum-mono assms(3) exp-k-lbound) auto
  also have ... = exp (- exp (real k) * real k ^ 3) / γ using assms(1) by simp
  also have ... = exp (real k + ln (k^2) - exp (real k) * real k ^ 3)
    using that unfolding γ-def
    by (simp add:exp-add exp-diff exp-minus algebra-simps inverse-eq-divide)
  also have ... = exp (real k + 2 * ln k - exp (real k) * real k ^ 3)
    using that by (subst ln-powr[symmetric]) auto
  also have ... ≤ exp (real k + 2 * real k - exp (ln l) * real k^3)
    using that k-gt-l ln-bound
    by (intro iffD2[OF exp-le-cancel-iff] add-mono diff-mono mult-left-mono mult-right-mono)
auto
  also have ... = exp (3* real k - l * (real k^3-1) -l)
    using assms(1) by (subst exp-ln) (auto simp add:algebra-simps)
  also have ... ≤ exp (3* real k - 1 * (real k^3-1) -l)
```

```
    using assms(1) that by (intro iffD2[OF exp-le-cancel-iff] diff-mono mult-right-mono) auto
  also have ... = exp (3* real k - real k * real k^2-1 -l+2)
    by (simp add:power2-eq-square power3-eq-cube)
  also have ... ≤ exp (3* real k - real k * 2^2-0 -l+2)
    using assms(1) that
    by (intro iffD2[OF exp-le-cancel-iff] add-mono diff-mono mult-left-mono power-mono) auto
  also have ... = exp (- real l - real k + 2) by simp
  finally show ?thesis by simp
qed

have ?L ≤ measure ?w
  {w. ?a+b*(∑ k=3..<k-max. exp (real k) * card {i∈{..<l}. f (w i)≥exp (real k)}) ≥ C1*l}
  using order-trans[OF - 2] by (intro pmf-mono) simp
also have ... = measure ?w
  {w. (∑ k=3..<k-max. exp(real k)*card{i∈{..<l}.f(w i)≥exp(real k)})≥l}
  unfolding C1-def b-def[symmetric] using b-gt-0
  by (intro-cong [σ2 measure] more:Collect-cong) (simp add:algebra-simps)
also have ... ≤ measure ?w
  {w. (∃ k∈{3..<k-max}. exp (real k)*card{i∈{..<l}.f(w i)≥exp(real k)} ≥ real l/real k^2)}
  using 3 by (intro pmf-mono) simp
also have ... = measure ?w
  (⋃ k∈{3..<k-max}. {w. exp (real k)*card{i∈{..<l}.f(w i)≥exp(real k)} ≥ real l/real k^2})
  by (intro-cong [σ2 measure]) auto
also have ... ≤ (∑ k=3..<k-max. L k)
  unfolding L-def by (intro finite-measure.finite-measure-subadditive-finite) auto
also have ... ≤ (∑ k=3..<k-max. exp (- real l - real k + 2)) by (intro sum-mono 4) auto
also have ... = (∑ k=0+3..<(k-max-3)+3. exp (- real l - real k + 2))
  using k-max-ge-3 by (intro sum.cong) auto
also have ... = (∑ k=0..<k-max-3. exp (-1 - real l - real k))
  by (subst sum.shift-bounds-nat-ivl) ( simp add:algebra-simps)
also have ... = exp(-1-real l) * (∑ k<k-max-3. exp (real k*(-1)))
  using atLeast0LessThan
  by (simp add:exp-diff exp-add sum-distrib-left exp-minus inverse-eq-divide)
also have ... = exp (-1-real l) * ((exp (- 1) ^(k-max - 3) - 1) / (exp (- 1) - 1))
  unfolding exp-of-nat-mult by (subst geometric-sum) auto
also have ... = exp(-1-real l) * (1-exp (- 1) ^(k-max - 3)) / (1-exp (- 1))
  by (simp add:field-simps)
also have ... ≤ exp(-1-real l) * (1-0) / (1-exp (- 1))
  using k-max-ge-3 by (intro mult-left-mono divide-right-mono diff-mono) auto
also have ... = exp (-real l) * (exp (-1) / (1-exp(-1)))
  by (simp add:exp-diff exp-minus inverse-eq-divide)
also have ... ≤ exp (-real l) * 1
  by (intro mult-left-mono exp-ge-zero) (approximation 10)
finally show ?thesis by simp
qed

unbundle no intro-cong-syntax

end
```

6 Inner Algorithm

This section introduces the inner algorithm (as mentioned it is already a solution to the cardinality estimation with the caveat that, if ε is too small it requires to much space. The outer algorithm in Section 10 resolved this problem.

The algorithm makes use of the balls and bins model, more precisely, the fact that the

number of hit bins can be used to estimate the number of balls thrown (even if there are collusions). I.e. it assigns each universe element to a bin using a k-wise independent hash function. Then it counts the number of bins hit.

This strategy however would only work if the number of balls is roughly equal to the number of bins, to remedy that the algorithm performs an adaptive sub-sampling strategy. This works by assigning each universe element a level (using a second hash function) with a geometric distribution. The algorithm then selects a level that is appropriate based on a rough estimate obtained using the maximum level in the bins.

To save space the algorithm drops information about small levels, whenever the space usage would be too high otherwise. This level will be called the cutoff-level. This is okey as long as the cutoff level is not larger than the sub-sampling threshold. A lot of the complexity in the proof is devoted to verifying that the cutoff-level will not cross it, it works by defining a third value s_M that is both an upper bound for the cutoff level and a lower bound for the subsampling threshold simultaneously with high probability.

theory *Distributed-Distinct-Elements-Inner-Algorithm*
imports
Universal-Hash-Families.Pseudorandom-Objects-Hash-Families
Distributed-Distinct-Elements-Preliminary
Distributed-Distinct-Elements-Balls-and-Bins
Distributed-Distinct-Elements-Tail-Bounds
Prefix-Free-Code-Combinators.Prefix-Free-Code-Combinators
begin

unbundle *intro-cong-syntax*
hide-const *Abstract-Rewriting.restrict*

definition C_4 :: *real* **where** $C_4 = 3\hat{}2*2\hat{}23$
definition C_5 :: *int* **where** $C_5 = 33$
definition C_6 :: *real* **where** $C_6 = 4$
definition C_7 :: *nat* **where** $C_7 = 2\hat{}5$

locale *inner-algorithm* =
fixes n :: *nat*
fixes δ :: *real*
fixes ε :: *real*
assumes *n-gt-0*: $n > 0$
assumes δ*-gt-0*: $\delta > 0$ **and** δ*-lt-1*: $\delta < 1$
assumes ε*-gt-0*: $\varepsilon > 0$ **and** ε*-lt-1*: $\varepsilon < 1$
begin

definition *b-exp* **where** *b-exp* = *nat* $\lceil log\ 2\ (C_4\ /\ \varepsilon\hat{}2)\rceil$
definition *b* :: *nat* **where** *b* = $2\hat{}$*b-exp*
definition *l* **where** *l* = *nat* $\lceil C_6 * ln\ (2/\ \delta)\rceil$
definition *k* **where** *k* = *nat* $\lceil C_2{*}ln\ b + C_3\rceil$
definition Λ :: *real* **where** $\Lambda = min\ (1/16)\ (exp\ (-l * ln\ l\hat{}3))$
definition ϱ :: *real* $\Rightarrow$ *real* **where** $\varrho\ x = b * (1 - (1-1/b)\ powr\ x)$
definition ϱ*-inv* :: *real* $\Rightarrow$ *real* **where** ϱ*-inv* $x = ln\ (1-x/b)\ /\ ln\ (1-1/b)$

lemma *l-lbound*: $C_6 * ln\ (2\ /\ \delta) \leq l$
unfolding *l-def* **by** *linarith*

lemma *k-min*: $C_2 * ln\ (real\ b) + C_3 \leq real\ k$
unfolding *k-def* **by** *linarith*

lemma Λ*-gt-0*: $\Lambda > 0$
unfolding Λ*-def min-less-iff-conj* **by** *auto*

lemma Λ-*le-1*: $\Lambda \leq 1$
unfolding Λ-*def* **by** *auto*

lemma *l-gt-0*: $l > 0$
proof −
have $0 < C_6 * ln\ (2\ /\ \delta)$
unfolding C_6-*def* **using** δ-*gt-0* δ-*lt-1*
by (*intro Rings.mult-pos-pos ln-gt-zero*) *auto*
also have ... $\leq l$
by (*intro l-lbound*)
finally show *?thesis*
by *simp*
qed

lemma *l-ubound*: $l \leq C_6 * ln(1\ /\ \delta) + C_6 * ln\ 2 + 1$
proof −
have $l = of\text{-}int\ \lceil C_6 * ln\ (2/\ \delta)\rceil$
using *l-gt-0* **unfolding** *l-def*
by (*intro of-nat-nat*) *simp*
also have ... $\leq C_6 * ln\ (1/\ \delta * 2) + 1$
by *simp*
also have ... $= C_6 * ln\ (1/\ \delta) + C_6 * ln\ 2 + 1$
using δ-*gt-0* δ-*lt-1*
by (*subst ln-mult*) (*auto simp add*:*algebra-simps*)
finally show *?thesis* **by** *simp*
qed

lemma *b-exp-ge-26*: $b\text{-}exp \geq 26$
proof −
have $2\ powr\ 25 < C_4\ /\ 1$ **unfolding** C_4-*def* **by** *simp*
also have ... $\leq C_4\ /\ \varepsilon\hat{}2$
using ε-*gt-0* ε-*lt-1* **unfolding** C_4-*def*
by (*intro divide-left-mono power-le-one*) *auto*
finally have $2\ powr\ 25 < C_4\ /\ \varepsilon\hat{}2$ **by** *simp*
hence $log\ 2\ (C_4\ /\ \varepsilon\hat{}2) > 25$
using ε-*gt-0* **unfolding** C_4-*def*
by (*intro iffD2*[*OF less-log-iff*] *divide-pos-pos zero-less-power*) *auto*
hence $\lceil log\ 2\ (C_4\ /\ \varepsilon\hat{}2)\rceil \geq 26$ **by** *simp*
thus *?thesis*
unfolding *b-exp-def* **by** *linarith*
qed

lemma *b-min*: $b \geq 2\hat{}26$
unfolding *b-def*
by (*meson b-exp-ge-26 nat-power-less-imp-less not-less power-eq-0-iff power-zero-numeral*)

lemma *k-gt-0*: $k > 0$
proof −
have (*0*::*real*) $< 7.5 * 0 + 16$ **by** *simp*
also have ... $\leq 7.5 * ln(real\ b) + 16$
using *b-min*
by (*intro add-mono mult-left-mono ln-ge-zero*) *auto*
finally have $0 < real\ k$
using *k-min* **unfolding** C_2-*def* C_3-*def* **by** *simp*
thus *?thesis* **by** *simp*
qed

lemma *b-ne*: $\{..<b\} \neq \{\}$
proof −
  have *0* $\in \{0..<b\}$
    using *b-min* **by** *simp*
  thus *?thesis*
    by *auto*
qed

lemma *b-lower-bound*: C_4 / $\varepsilon\hat{}2 \leq$ *real b*
proof −
  have C_4 / $\varepsilon\hat{}2$ = *2 powr* (*log 2* (C_4 / $\varepsilon\hat{}2$))
    using *ε-gt-0* **unfolding** C_4*-def* **by** (*intro powr-log-cancel*[*symmetric*] *divide-pos-pos*) *auto*
  also have ... ≤ *2 powr* (*nat* ⌈*log 2* (C_4 / $\varepsilon\hat{}2$)⌉)
    by (*intro powr-mono of-nat-ceiling*) *simp*
  also have ... = *real b*
    unfolding *b-def b-exp-def* **by** (*simp add*:*powr-realpow*)
  finally show *?thesis* **by** *simp*
qed

definition *n-exp* **where** *n-exp* = *max* (*nat* ⌈*log 2 n*⌉) *1*

lemma *n-exp-gt-0*: *n-exp* > *0*
  unfolding *n-exp-def* **by** *simp*

abbreviation Ψ_1 **where** $\Psi_1 \equiv \mathcal{H}$ *2 n* ($\mathcal{G}$ *n-exp*)
abbreviation Ψ_2 **where** $\Psi_2 \equiv \mathcal{H}$ *2 n* ($\mathcal{N}$ (C_7∗b^2))
abbreviation Ψ_3 **where** $\Psi_3 \equiv \mathcal{H}$ *k* (C_7∗b^2) ($\mathcal{N}$ *b*)

definition Ψ **where** Ψ = Ψ_1 ×$_P$ Ψ_2 ×$_P$ Ψ_3

abbreviation Ω **where** Ω ≡ $\mathcal{E}$ *l* Λ Ψ

type-synonym *state* = (*nat* ⇒ *nat* ⇒ *int*) × (*nat*)

fun *is-too-large* :: (*nat* ⇒ *nat* ⇒ *int*) ⇒ *bool* **where**
  is-too-large B = ((∑ (*i*,*j*) ∈ {..<*l*} × {..<*b*}. ⌊*log 2* (*max* (*B i j*) (−*1*) + *2*)⌋) > C_5 ∗ *b* ∗ *l*)

fun *compress-step* :: *state* ⇒ *state* **where**
  compress-step (*B*,*q*) = (λ *i j*. *max* (*B i j* − *1*) (−*1*), *q*+*1*)

function *compress* :: *state* ⇒ *state* **where**
  compress (*B*,*q*) = (
    if is-too-large B
      then (*compress* (*compress-step* (*B*,*q*)))
      else (*B*,*q*))
  by *auto*

fun *compress-termination* :: *state* ⇒ *nat* **where**
  compress-termination (*B*,*q*) = (∑ (*i*,*j*) ∈ {..<*l*} × {..<*b*}. *nat* (*B i j* + *1*))

lemma *compress-termination*:
  assumes *is-too-large B*
  shows *compress-termination* (*compress-step* (*B*,*q*)) < *compress-termination* (*B*,*q*)
proof (*rule ccontr*)
  let *?I* = {..<*l*} × {..<*b*}
  have *a*: *nat* (*max* (*B i j* − *1*) (− *1*) + *1*) ≤ *nat* (*B i j* + *1*) **for** *i j*
    by *simp*
  assume ¬ *compress-termination* (*compress-step* (*B*, *q*)) < *compress-termination* (*B*, *q*)

```
  hence (∑ (i,j) ∈ ?I. nat (B i j + 1)) ≤ (∑ (i,j) ∈ ?I. nat (max (B i j − 1) (−1) + 1))
    by simp
  moreover have (∑ (i,j) ∈ ?I. nat (B i j + 1)) ≥ (∑ (i,j) ∈ ?I. nat (max (B i j − 1) (−1)
+ 1))
    by (intro sum-mono) auto
  ultimately have b:
    (∑ (i,j) ∈ ?I. nat (max (B i j − 1) (−1) + 1)) = (∑ (i,j) ∈ ?I. nat (B i j + 1))
    using order-antisym by simp
  have nat (B i j + 1) = nat (max (B i j − 1) (−1) + 1) if (i,j) ∈ ?I for i j
    using sum-mono-inv[OF b] that a by auto
  hence max (B i j) (−1) = −1 if (i,j) ∈ ?I for i j
    using that by fastforce
  hence (∑(i,j) ∈ ?I. ⌊log 2 (max (B i j) (−1) + 2)⌋) = (∑(i,j) ∈ ?I. 0)
    by (intro sum.cong, auto)
  also have ... = 0 by simp
  also have ... ≤ C₅ * b * l unfolding C₅-def by simp
  finally have ¬ is-too-large B by simp
  thus False using assms by simp
qed

termination compress
  using measure-def compress-termination
  by (relation Wellfounded.measure (compress-termination), auto)

fun merge1 :: state ⇒ state ⇒ state where
  merge1 (B1,q₁) (B2, q₂) = (
    let q = max q₁ q₂ in (λ i j. max (B1 i j + q₁ − q) (B2 i j + q₂ − q), q))

fun merge :: state ⇒ state ⇒ state where
  merge x y = compress (merge1 x y)

type-synonym seed = nat ⇒ (nat ⇒ nat) × (nat ⇒ nat) × (nat ⇒ nat)

fun single1 :: seed ⇒ nat ⇒ state where
  single1 ω x = (λ i j.
     let (f,g,h) = ω i in (
     if h (g x) = j ∧ i < l then int (f x) else (−1)), 0)

fun single :: seed ⇒ nat ⇒ state where
  single ω x = compress (single1 ω x)

fun estimate1 :: state ⇒ nat ⇒ real where
  estimate1 (B,q) i = (
    let s = max 0 (Max ((B i) ` {..<b}) + q − ⌊log 2 b⌋ + 9);
        p = card { j. j ∈ {..<b} ∧ B i j + q ≥ s } in
        2 powr s * ln (1−p/b) / ln(1−1/b))

fun estimate :: state ⇒ real where
  estimate x = median l (estimate1 x)
```

6.1 History Independence

```
fun τ₀ :: ((nat ⇒ nat) × (nat ⇒ nat) × (nat ⇒ nat)) ⇒ nat set ⇒ nat ⇒ int
  where τ₀ (f,g,h) A j = Max ({ int (f a) | a . a ∈ A ∧ h (g a) = j } ∪ {−1})

definition τ₁ :: ((nat ⇒ nat) × (nat ⇒ nat) × (nat ⇒ nat)) ⇒ nat set ⇒ nat ⇒ nat ⇒ int
  where τ₁ ψ A q j = max (τ₀ ψ A j − q) (−1)
```

definition τ_2 :: *seed* $\Rightarrow$ *nat set* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *int*
where $\tau_2\ \omega\ A\ q\ i\ j = (\textit{if}\ i < l\ \textit{then}\ \tau_1\ (\omega\ i)\ A\ q\ j\ \textit{else}\ (-1))$

definition τ_3 :: *seed* $\Rightarrow$ *nat set* $\Rightarrow$ *nat* $\Rightarrow$ *state*
where $\tau_3\ \omega\ A\ q = (\tau_2\ \omega\ A\ q,\ q)$

definition q :: *seed* $\Rightarrow$ *nat set* $\Rightarrow$ *nat*
where $q\ \omega\ A = (\textit{LEAST}\ q\ .\ \neg(\textit{is-too-large}\ (\tau_2\ \omega\ A\ q)))$

definition τ :: *seed* $\Rightarrow$ *nat set* $\Rightarrow$ *state*
where $\tau\ \omega\ A = \tau_3\ \omega\ A\ (q\ \omega\ A)$

lemma τ_2*-step*: $\tau_2\ \omega\ A\ (x+y) = (\lambda i\ j.\ \textit{max}\ (\tau_2\ \omega\ A\ x\ i\ j - y)\ (-\ 1))$
by (*intro ext*) (*auto simp add*:τ_2*-def* τ_1*-def*)

lemma τ_3*-step*: *compress-step* $(\tau_3\ \omega\ A\ x) = \tau_3\ \omega\ A\ (x+1)$
unfolding τ_3*-def* **using** τ_2*-step*[**where** *y=1*] **by** *simp*

lemma Ψ_1: *is-prime-power* (*pro-size* ($\mathcal{G}$ *n-exp*))
unfolding *geom-pro-size* **by** (*intro is-prime-powerI n-exp-gt-0*) *auto*

lemma Ψ_2: *is-prime-power* (*pro-size* ($\mathcal{N}$ (C_7 * *b*^2)))
proof −
have *0*:*pro-size* ($\mathcal{N}$ (C_7 * *b*^2)) = *2* ^ (*5* + *2* * *b-exp*)
unfolding C_7*-def b-def* **by** (*subst nat-pro-size*) (*auto simp add*: *power-add power-even-eq*)
thus *?thesis* **unfolding** *0* **by** (*intro is-prime-powerI*) *auto*
qed

lemma Ψ_3: *is-prime-power* (*pro-size* ($\mathcal{N}$ *b*))
proof −
have *0*:*pro-size* ($\mathcal{N}$ *b*) = *2* ^ *b-exp* **unfolding** *b-def* **by** (*subst nat-pro-size*) *auto*
thus *?thesis* **using** *b-exp-ge-26* **unfolding** *0* **by** (*intro is-prime-powerI*) *auto*
qed

lemma *sample-pro-*Ψ:
sample-pro Ψ = *pair-pmf* (*sample-pro* Ψ_1) (*pair-pmf* (*sample-pro* Ψ_2) (*sample-pro* Ψ_3))
unfolding Ψ*-def* **by** (*simp add*:*prod-pro*)

lemma *sample-set-*Ψ: *pro-set* Ψ = *pro-set* Ψ_1 × *pro-set* Ψ_2 × *pro-set* Ψ_3
unfolding Ψ*-def* **by** (*simp add*:*prod-pro-set*)

lemma *f-range*:
assumes (*f*,*g*,*h*) ∈ *pro-set* Ψ
shows *f x* ≤ *n-exp*
proof −
have *f* ∈ *pro-set* Ψ_1 **using** *sample-set-*Ψ *assms* **by** *auto*
hence *f* ∈ *pro-select* Ψ_1 ‘ {..<*pro-size* Ψ_1} **unfolding** *set-sample-pro* **by** *auto*
hence *f x* ∈ *pro-set* ($\mathcal{G}$ *n-exp*) **using** *hash-pro-range*[*OF* Ψ_1] **by** *auto*
thus *?thesis* **using** *geom-pro-range* **by** *auto*
qed

lemma *g-range-1*:
assumes *g* ∈ *pro-set* Ψ_2
shows *g x* < C_7*b^2
proof −
have *g* ∈ *pro-select* Ψ_2 ‘ {..<*pro-size* Ψ_2} **using** *assms* **unfolding** *set-sample-pro* **by** *auto*
hence *g x* ∈ *pro-set* ($\mathcal{N}$ (C_7*b^2)) **using** *hash-pro-range*[*OF* Ψ_2] **by** *auto*
moreover have C_7*b^2 > *0* **unfolding** C_7*-def b-def* **by** *simp*

ultimately show *?thesis* **using** *nat-pro-set* **by** *auto*
qed

lemma *h-range-1*:
assumes $h \in$ *pro-set* Ψ_3
shows $h\ x < b$
proof −
have $h \in$ *pro-select* Ψ_3 ‘ {..<*pro-size* Ψ_3} **using** *assms* **unfolding** *set-sample-pro* **by** *auto*
hence $h\ x \in$ *pro-set* $(\mathcal{N}\ b)$ **using** *hash-pro-range*[*OF* Ψ_3] **by** *auto*
moreover have $b > 0$ **unfolding** *b-def* **by** *simp*
ultimately show *?thesis* **using** *nat-pro-set* **by** *auto*
qed

lemma *g-range*:
assumes $(f,g,h) \in$ *pro-set* Ψ
shows $g\ x < C_7 * b\hat{}2$
using *g-range-1 sample-set-*Ψ *assms* **by** *simp*

lemma *h-range*:
assumes $(f,g,h) \in$ *pro-set* Ψ
shows $h\ x < b$
using *h-range-1 sample-set-*Ψ *assms* **by** *simp*

lemma *fin-f*:
assumes $(f,g,h) \in$ *pro-set* Ψ
shows *finite* { *int* $(f\ a)$ | *a*. *P a* } (**is** *finite ?M*)
proof −
have *finite* (*range f*)
using *f-range*[*OF assms*] *finite-nat-set-iff-bounded-le* **by** *auto*
hence *finite* (*range* (*int* ∘ *f*))
by (*simp add*:*image-image*[*symmetric*])
moreover have *?M* ⊆ (*range* (*int* ∘ *f*))
using *image-mono* **by** (*auto simp add*: *setcompr-eq-image*)
ultimately show *?thesis*
using *finite-subset* **by** *auto*
qed

lemma *Max-int-range*: $x \leq (y::int) \Longrightarrow Max\ \{x..y\} = y$
by *auto*

lemma Ω: $l > 0$ $\Lambda > 0$ **using** *l-gt-0* Λ*-gt-0* **by** *auto*

lemma ω*-range*:
assumes $\omega \in$ *pro-set* Ω
shows $\omega\ i \in$ *pro-set* Ψ
proof −
have $\omega \in$ *pro-select* Ω ‘ {..<*pro-size* Ω} **using** *assms* **unfolding** *set-sample-pro* **by** *auto*
thus $\omega\ i \in$ *pro-set* Ψ **using** *expander-pro-range*[*OF* Ω] *assms* **by** *auto*
qed

lemma *max-q-1*:
assumes $\omega \in$ *pro-set* Ω
shows $\tau_2\ \omega\ A$ (*nat* ⌈*log 2 n*⌉+2) *i j* = (−1)
proof (*cases i* < *l*)
case *True*
obtain *f g h* **where** *w-i*: $\omega\ i = (f,g,h)$ **by** (*metis prod-cases3*)

let *?max-q* = *max* ⌈*log 2* (*real n*)⌉ *1*

```
  have c: (f,g,h) ∈ pro-set Ψ using ω-range[OF assms] w-i[symmetric] by auto
  have a:int (f x) ≤ ?max-q for x
  proof −
    have int (f x) ≤ int n-exp
      using f-range[OF c] by auto
    also have ... = ?max-q unfolding n-exp-def by simp
    finally show ?thesis by simp
  qed
  have τ0 (ω i) A j ≤ Max {(−1)..?max-q}
    unfolding w-i τ0.simps using a by (intro Max-mono)  auto
  also have ... = ?max-q
    by (intro Max-int-range) auto
  finally have τ0 (ω i) A j ≤ ?max-q by simp
  hence max (τ0 (ω i) A j − int (nat ⌈log 2 (real n)⌉ + 2)) (− 1) = (−1)
    by (intro max-absorb2) linarith
  thus ?thesis
    unfolding τ2-def τ1-def using True by auto
next
  case False
  thus ?thesis unfolding τ2-def τ1-def by simp
qed

lemma max-q-2:
  assumes ω ∈ pro-set Ω
  shows ¬ (is-too-large (τ2 ω A (nat ⌈log 2 n⌉+2)))
  using max-q-1[OF assms] by (simp add:C5-def case-prod-beta mult-less-0-iff)

lemma max-s-3:
  assumes ω ∈ pro-set Ω
  shows q ω A ≤ (nat ⌈log 2 n⌉+2)
  unfolding q-def by (intro wellorder-Least-lemma(2) max-q-2 assms)

lemma max-mono: x ≤ (y::'a::linorder) ⟹ max x z ≤ max y z
  using max.coboundedI1 by auto

lemma max-mono-2: y ≤ (z::'a::linorder) ⟹ max x y ≤ max x z
  using max.coboundedI2 by auto

lemma τ0-mono:
  assumes ψ ∈ pro-set Ψ
  assumes A ⊆ B
  shows τ0 ψ A j ≤ τ0 ψ B j
proof −
  obtain f g h where w-i: ψ = (f,g,h)
    by (metis prod-cases3)
  show ?thesis
    using assms fin-f unfolding τ0.simps w-i
    by (intro Max-mono) auto
qed

lemma τ2-mono:
  assumes ω ∈ pro-set Ω
  assumes A ⊆ B
  shows τ2 ω A x i j ≤ τ2 ω B x i j
proof −
  have max (τ0 (ω i) A j − int x) (− 1) ≤ max (τ0 (ω i) B j − int x) (− 1) if i < l
   using that ω-range[OF assms(1)] by (intro max-mono diff-mono τ0-mono assms(2) order.refl)
```

thus *?thesis* **by** (*cases* $i < l$) (*auto simp add*:τ_2*-def* τ_1*-def*)
qed

lemma *is-too-large-antimono*:
assumes $\omega \in$ *pro-set* Ω
assumes $A \subseteq B$
assumes *is-too-large* ($\tau_2\ \omega\ A\ x$)
shows *is-too-large* ($\tau_2\ \omega\ B\ x$)
proof −
have $C_5 * b * l < (\sum\ (i,j) \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (\tau_2\ \omega\ A\ x\ i\ j)\ (-1) + 2)\rfloor)$
using *assms*(*3*) **by** *simp*
also have ... $= (\sum\ y \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (\tau_2\ \omega\ A\ x\ (fst\ y)\ (snd\ y))\ (-1) + 2)\rfloor)$
by (*simp add*:*case-prod-beta*)
also have ... $\leq (\sum\ y \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (\tau_2\ \omega\ B\ x\ (fst\ y)\ (snd\ y))\ (-1) + 2)\rfloor)$
by (*intro sum-mono floor-mono iffD2*[*OF log-le-cancel-iff*] *iffD2*[*OF of-int-le-iff*]
add-mono max-mono τ_2*-mono*[*OF assms*(*1*,*2*)]) *auto*
also have ... $= (\sum\ (i,j) \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (\tau_2\ \omega\ B\ x\ i\ j)\ (-1) + 2)\rfloor)$
by (*simp add*:*case-prod-beta*)
finally have $(\sum\ (i,j) \in \{..<l\} \times \{..<b\}.\ \lfloor log\ 2\ (max\ (\tau_2\ \omega\ B\ x\ i\ j)\ (-1) + 2)\rfloor) > C_5 * b * l$
by *simp*
thus *?thesis* **by** *simp*
qed

lemma *q-compact*:
assumes $\omega \in$ *pro-set* Ω
shows ¬ (*is-too-large* ($\tau_2\ \omega\ A\ (q\ \omega\ A)$))
unfolding *q-def* **using** *max-q-2*[*OF assms*]
by (*intro wellorder-Least-lemma*(*1*)) *blast*

lemma *q-mono*:
assumes $\omega \in$ *pro-set* Ω
assumes $A \subseteq B$
shows $q\ \omega\ A \leq q\ \omega\ B$
proof −
have ¬ (*is-too-large* ($\tau_2\ \omega\ A\ (q\ \omega\ B)$))
using *is-too-large-antimono*[*OF assms*] *q-compact*[*OF assms*(*1*)] **by** *blast*
hence (*LEAST q* . ¬(*is-too-large* ($\tau_2\ \omega\ A\ q$))) $\leq q\ \omega\ B$
by (*intro Least-le*) *blast*
thus *?thesis*
by (*simp add*:*q-def*)
qed

lemma *lt-s-too-large*: $x < q\ \omega\ A \Longrightarrow$ *is-too-large* ($\tau_2\ \omega\ A\ x$)
using *not-less-Least* **unfolding** *q-def* **by** *auto*

lemma *compress-result-1*:
assumes $\omega \in$ *pro-set* Ω
shows *compress* ($\tau_3\ \omega\ A\ (q\ \omega\ A - i)$) $= \tau\ \omega\ A$
proof (*induction i*)
case *0*
then show *?case* **using** *q-compact*[*OF assms*] **by** (*simp add*:τ_3*-def* τ*-def*)
next
case (*Suc i*)
show *?case*
proof (*cases* $i < q\ \omega\ A$)
case *True*
have *is-too-large* ($\tau_2\ \omega\ A\ (q\ \omega\ A - Suc\ i)$)
using *True* **by** (*intro lt-s-too-large*) *simp*

```
    hence compress (τ3 ω A (q ω A − Suc i)) = compress (compress-step (τ3 ω A (q ω A − Suc
i)))
      unfolding τ3-def compress.simps
      by (simp del: compress.simps compress-step.simps)
    also have ... = compress (τ3 ω A ((q ω A − Suc i)+1))
      by (subst τ3-step) blast
    also have ... = compress (τ3 ω A (q ω A − i))
      using True by (metis Suc-diff-Suc Suc-eq-plus1)
    also have ... = τ ω A using Suc by auto
    finally show ?thesis by simp
  next
    case False
    then show ?thesis using Suc by simp
  qed
qed

lemma compress-result:
  assumes ω ∈ pro-set Ω
  assumes x ≤ q ω A
  shows compress (τ3 ω A x) = τ ω A
proof −
  obtain i where i-def: x = q ω A − i using assms by (metis diff-diff-cancel)
  have compress (τ3 ω A x) = compress (τ3 ω A (q ω A − i))
    by (subst i-def) blast
  also have ... = τ ω A
    using compress-result-1[OF assms(1)] by blast
  finally show ?thesis by simp
qed

lemma τ0-merge:
  assumes (f,g,h) ∈ pro-set Ψ
  shows τ0 (f,g,h) (A ∪ B) j = max (τ0 (f,g,h) A j) (τ0 (f,g,h) B j) (is ?L = ?R)
proof−
  let ?f = λa. int (f a)
  have ?L = Max (({ int (f a) |  a . a ∈ A ∧ h (g a) = j } ∪ {−1}) ∪
                 ({ int (f a) | a . a ∈ B ∧ h (g a) = j } ∪ {−1}))
    unfolding τ0.simps
    by (intro arg-cong[where f=Max])  auto
  also have ... = max (Max ({ int (f a) | a . a ∈ A ∧ h (g a) = j } ∪ {−1}))
                      (Max ({ int (f a) | a . a ∈ B ∧ h (g a) = j } ∪ {−1}))
    by (intro Max-Un finite-UnI fin-f[OF assms]) auto
  also have ... = ?R
    by (simp)
  finally show ?thesis by simp
qed

lemma τ2-merge:
  assumes ω ∈ pro-set Ω
  shows τ2 ω (A ∪ B) x i j = max (τ2 ω A x i j) (τ2 ω B x i j)
proof (cases i < l)
  case True

  obtain f g h where w-i: ω i = (f,g,h) by (metis prod-cases3)

  have a: (f,g,h) ∈ pro-set Ψ using w-i[symmetric] ω-range[OF assms(1)] by auto
  show ?thesis
    unfolding τ2-def τ1-def
    using True by (simp add:w-i τ0-merge[OF a] del:τ0.simps)
```

next
case *False*
thus *?thesis* **by** (*simp add*:τ_2-*def*)
qed

lemma *merge1-result*:
assumes $\omega \in$ *pro-set* Ω
shows *merge1* $(\tau\ \omega\ A)\ (\tau\ \omega\ B) = \tau_3\ \omega\ (A \cup B)\ (max\ (q\ \omega\ A)\ (q\ \omega\ B))$
proof −
let *?qmax* = $max\ (q\ \omega\ A)\ (q\ \omega\ B)$
obtain u **where** *u-def*: $q\ \omega\ A + u =$ *?qmax*
by (*metis add.commute max.commute nat-minus-add-max*)
obtain v **where** *v-def*: $q\ \omega\ B + v =$ *?qmax*
by (*metis add.commute nat-minus-add-max*)

have $u = 0 \vee v = 0$ **using** *u-def v-def* **by** *linarith*
moreover have $\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j - u \geq (-1)$ **if** $u = 0$ **for** $i\ j$
using *that* **by** (*simp add*:τ_2-*def* τ_1-*def*)
moreover have $\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j - v \geq (-1)$ **if** $v = 0$ **for** $i\ j$
using *that* **by** (*simp add*:τ_2-*def* τ_1-*def*)
ultimately have *a*:*max* $(\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j - u)\ (\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j - v) \geq (-1)$ **for** $i\ j$
unfolding *le-max-iff-disj* **by** *blast*

have $\tau_2\ \omega\ (A \cup B)$ *?qmax* = $(\lambda\ i\ j.\ \ max\ (\tau_2\ \omega\ A$ *?qmax i j*$)\ (\tau_2\ \omega\ B$ *?qmax i j*$))$
using τ_2-*merge*[*OF assms*] **by** *blast*
also have ... = $(\lambda\ i\ j.\ \ max\ (\tau_2\ \omega\ A\ (q\ \omega\ A + u)\ i\ j)\ (\tau_2\ \omega\ B\ (q\ \omega\ B + v)\ i\ j))$
unfolding *u-def v-def* **by** *blast*
also have ... = $(\lambda\ i\ j.\ \ max\ (max\ (\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j - u)\ (-1))\ (max\ (\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j -$
$v)\ (-1)))$
by (*simp only*: τ_2-*step*)
also have ... = $(\lambda\ i\ j.\ \ max\ (max\ (\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j - u)\ (\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j - v))\ (-1))$
by (*metis* (*no-types*, *opaque-lifting*) *max.commute max.left-commute max.left-idem*)
also have ... = $(\lambda\ i\ j.\ max\ (\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j - u)\ (\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j - v))$
using *a* **by** *simp*
also have ... = $\ (\lambda i\ j.\ max\ (\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j + int\ (q\ \omega\ A) - ?qmax)$
$(\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j + int\ (q\ \omega\ B) - ?qmax))$
by (*subst u-def*[*symmetric*], *subst v-def*[*symmetric*]) *simp*
finally have $\tau_2\ \omega\ (A \cup B)\ (max\ (q\ \omega\ A)\ (q\ \omega\ B)) =$
$(\lambda i\ j.\ max\ (\tau_2\ \omega\ A\ (q\ \omega\ A)\ i\ j + int\ (q\ \omega\ A) - int\ (?qmax))$
$(\tau_2\ \omega\ B\ (q\ \omega\ B)\ i\ j + int\ (q\ \omega\ B) - int\ (?qmax)))$ **by** *simp*
thus *?thesis*
by (*simp add*:*Let-def* τ-*def* τ_3-*def*)
qed

lemma *merge-result*:
assumes $\omega \in$ *pro-set* Ω
shows *merge* $(\tau\ \omega\ A)\ (\tau\ \omega\ B) = \tau\ \omega\ (A \cup B)$ (**is** *?L* = *?R*)
proof −
have *a*:*max* $(q\ \omega\ A)\ (q\ \omega\ B) \leq q\ \omega\ (A \cup B)$
using *q-mono*[*OF assms*] **by** *simp*

have *?L* = *compress* (*merge1* $(\tau\ \omega\ A)\ (\tau\ \omega\ B)$)
by *simp*
also have ... = *compress* ($\tau_3\ \omega\ (A \cup B)\ (max\ (q\ \omega\ A)\ (q\ \omega\ B))$)
by (*subst merge1-result*[*OF assms*]) *blast*
also have ... = *?R*
by (*intro compress-result*[*OF assms*] *a Un-least*)
finally show *?thesis* **by** *blast*

qed

lemma *single1-result*: *single1* ω $x = \tau_3$ ω $\{x\}$ *0*
proof −
have (*case* ω *i of* (*f*, *g*, *h*) ⇒ *if h* (*g x*) = *j* ∧ *i* < *l then int* (*f x*) *else* − *1*) = τ_2 ω $\{x\}$ *0 i j*
for *i j*
proof −
obtain *f g h* **where** *w-i*:ω *i* = (*f*, *g*,*h*) **by** (*metis prod-cases3*)
show *?thesis*
by (*simp add*:*w-i* τ_2*-def* τ_1*-def*)
qed
thus *?thesis*
unfolding τ_3*-def* **by** *fastforce*
qed

lemma *single-result*:
assumes $\omega \in$ *pro-set* Ω
shows *single* ω *x* = τ ω $\{x\}$ (**is** *?L* = *?R*)
proof −
have *?L* = *compress* (*single1* ω *x*)
by (*simp*)
also have ... = *compress* (τ_3 ω $\{x\}$ *0*)
by (*subst single1-result*) *blast*
also have ... = *?R*
by (*intro compress-result*[*OF assms*]) *auto*
finally show *?thesis* **by** *blast*
qed

6.2 Encoding states of the inner algorithm

definition *is-state-table* :: (*nat* × *nat* ⇒ *int*) ⇒ *bool* **where**
is-state-table g = (*range g* ⊆ {−*1*..} ∧ *g* ‘ (−({..<*l*} × {..<*b*})) ⊆ {−*1*})

Encoding for state table values:

definition V_e :: *int encoding*
where V_e *x* = (*if x* ≥ −*1 then* N_e (*nat* (*x*+*1*)) *else None*)

Encoding for state table:

definition $T_e{}'$ :: (*nat* × *nat* ⇒ *int*) *encoding* **where**
$T_e{}'$ *g* = (
if is-state-table g
then (*List.product* [*0*..<*l*] [*0*..<*b*] →$_e$ V_e) (*restrict g* ({..<*l*}×{..<*b*}))
else None)

definition T_e :: (*nat* ⇒ *nat* ⇒ *int*) *encoding*
where T_e *f* = $T_e{}'$ (*case-prod f*)

definition *encode-state* :: *state encoding*
where *encode-state* = T_e ×$_e$ Nb_e (*nat* ⌈*log 2 n*⌉+*3*)

lemma *inj-on-restrict*:
assumes *B* ⊆ {*f*. *f* ‘ (− *A*) ⊆ {*c*}}
shows *inj-on* (λ*x*. *restrict x A*) *B*
proof (*rule inj-onI*)
fix *f g* **assume** *a*:*f* ∈ *B g* ∈ *B restrict f A* = *restrict g A*

have *f x* = *g x* **if** *x* ∈ *A* **for** *x*
by (*intro restrict-eq-imp*[*OF a*(*3*) *that*])

```
  moreover have f x = g x if x ∉ A  for x
  proof −
    have f x = c g x = c
      using that a(1,2) assms(1) by auto
    thus ?thesis by simp
  qed
  ultimately show f = g
    by (intro ext) auto
qed

lemma encode-state: is-encoding encode-state
proof −
  have is-encoding V_e
    unfolding V_e-def
    by (intro encoding-compose[OF exp-golomb-encoding] inj-onI) auto
  hence 0:is-encoding (List.product [0..<l] [0..<b] →_e V_e)
    by (intro fun-encoding)
  have is-encoding T_e'
    unfolding T_e'-def is-state-table-def
    by (intro encoding-compose[OF 0] inj-on-restrict[where c=−1]) auto
  moreover have  inj case-prod
    by (intro injI)  (metis curry-case-prod)
  ultimately have is-encoding T_e
    unfolding T_e-def by (rule encoding-compose-2)

  thus ?thesis
    unfolding encode-state-def
    by (intro dependent-encoding bounded-nat-encoding)
qed

lemma state-bit-count:
  assumes ω ∈ pro-set Ω
  shows bit-count (encode-state (τ ω A)) ≤  2^36 * (ln(1/δ)+1)/ ε^2 + log 2 (log 2 n + 3)
    (is ?L ≤ ?R)
proof −
  define t where t = τ_2 ω A (q ω A)

  have log 2 (real n) ≥ 0
    using n-gt-0 by simp
  hence 0: − 1 < log 2 (real n)
    by simp

  have t x y = −1 if x < l y ≥ b for x y
  proof −
    obtain f g h where ω-def: ω x = (f,g,h)
      by (metis prod-cases3)
    have (f, g, h) ∈ pro-set Ψ
      using ω-range[OF assms] unfolding Pi-def ω-def[symmetric] by auto
    hence h (g a) < b for a
      using h-range by auto
    hence y ≠ h (g a) for a
      using that(2) not-less by blast
    hence aux-4: {int (f a) |a. a ∈ A ∧ h (g a) = y} = {}
      by auto
    hence max (Max (insert (− 1) {int (f a) |a. a ∈ A ∧ h (g a) = y}) − int (q ω A)) (− 1) =
− 1
      unfolding aux-4 by simp
    thus ?thesis
```

unfolding *t-def* τ_2*-def* τ_1*-def* **by** (*simp add:*ω*-def*)
qed
moreover have $t\ x\ y = -1$ **if** $x \geq l$ **for** $x\ y$
using *that* **unfolding** *t-def* τ_2*-def* τ_1*-def* **by** *simp*
ultimately have *1*: $t\ x\ y = -1$ **if** $x \geq l \vee y \geq b$ **for** $x\ y$
using *that* **by** (*meson not-less*)

have *2*: $t\ x\ y \geq -1$ **for** $x\ y$
unfolding *t-def* τ_2*-def* τ_1*-def* **by** *simp*
hence *3*: $t\ x\ y + 1 \geq 0$ **for** $x\ y$
by (*metis add.commute le-add-same-cancel1 minus-add-cancel*)

have *4*:*is-state-table* (*case-prod t*)
using *2 1* **unfolding** *is-state-table-def* **by** *auto*

have *bit-count*(T_e (τ_2 ω A (q ω A))) = *bit-count*(T_e t)
unfolding *t-def* **by** *simp*
also have ... = *bit-count* ((*List.product* [*0*..<*l*] [*0*..<*b*] →$_e$ V_e) (λ(x, y)∈{..<*l*}×{..<*b*}. $t\ x\ y$))
using *4* **unfolding** T_e*-def* T_e*'-def* **by** *simp*
also have ... =
(∑ *x*←*List.product* [*0*..<*l*] [*0*..<*b*]. *bit-count* (V_e ((λ(x, y)∈{..<*l*} × {..<*b*}. $t\ x\ y$) x)))
using *restrict-extensional atLeast0LessThan* **by** (*simp add:fun-bit-count*)
also have ... = (∑ (*x*,*y*)←*List.product* [*0*..<*l*] [*0*..<*b*]. *bit-count* (V_e ($t\ x\ y$)))
by (*intro arg-cong*[**where** *f*=*sum-list*] *map-cong refl*)
(*simp add:atLeast0LessThan case-prod-beta*)
also have ... = (∑ *x*∈{*0*..<*l*} × {*0*..<*b*}. *bit-count* (V_e (t (*fst x*) (*snd x*))))
by (*subst sum-list-distinct-conv-sum-set*)
(*auto intro:distinct-product simp add:case-prod-beta*)
also have ... = (∑ *x*∈{..<*l*} × {..<*b*}. *bit-count* (N_e (*nat* (t (*fst x*) (*snd x*)+*1*))))
using *2* **unfolding** V_e*-def not-less*[*symmetric*]
by (*intro sum.cong refl arg-cong*[**where** *f*=*bit-count*]) *auto*
also have ...=(∑ *x*∈{..<*l*}×{..<*b*}. *1*+*2*∗ *of-int*⌊*log 2*(*1*+*real*(*nat*(*t* (*fst x*)(*snd x*)+*1*)))⌋)
unfolding *exp-golomb-bit-count-exact is-too-large.simps not-less* **by** *simp*
also have ...=(∑ *x*∈{..<*l*}×{..<*b*}. *1*+*2*∗ *of-int*⌊*log 2*(*2*+ *of-int*(*t* (*fst x*)(*snd x*)))⌋)
using *3* **by** (*subst of-nat-nat*) (*auto simp add:ac-simps*)
also have ...=*b*∗*l* + *2*∗ *of-int* (∑ (*i*,*j*)∈{..<*l*}×{..<*b*}. ⌊*log 2*(*2*+ *of-int*(*max* (*t i j*) (−*1*)))⌋)
using *2* **by** (*subst max-absorb1*) (*auto simp add:case-prod-beta sum.distrib sum-distrib-left*)
also have ... ≤ *b*∗*l* + *2* ∗ *of-int* (C_5 ∗ *int b* ∗ *int l*)
using *q-compact*[*OF assms*, **where** *A*=*A*] **unfolding** *is-too-large.simps not-less t-def*[*symmetric*]
by (*intro add-mono ereal-mono iffD2*[*OF of-int-le-iff*] *mult-left-mono order.refl*)
(*simp-all add:ac-simps*)
also have ... = (*2* ∗ C_5 + *1*) ∗ *b* ∗ *l*
by (*simp add:algebra-simps*)
finally have *5*:*bit-count* (T_e (τ_2 ω A (q ω A))) ≤ (*2* ∗ C_5 + *1*) ∗ *b* ∗ *l*
by *simp*

have $C_4 \geq 1$
unfolding C_4*-def* **by** *simp*
moreover have $\varepsilon^2 \leq 1$
using *ε-lt-1 ε-gt-0*
by (*intro power-le-one*) *auto*
ultimately have *0* ≤ *log 2* (C_4 / ε^2)
using *ε-gt-0 ε-lt-1*
by (*intro iffD2*[*OF zero-le-log-cancel-iff*] *divide-pos-pos*)*auto*
hence *6*: − *1* < *log 2* (C_4 / ε^2)
by *simp*

have *b* = *2 powr* (*real* (*nat* ⌈*log 2* (C_4 / ε^2)⌉))

unfolding *b-def b-exp-def* **by** (*simp add:powr-realpow*)
also have ... = *2 powr* (⌈*log 2* (*C*$_4$ / ε^2)⌉)
using *6* **by** (*intro arg-cong2*[**where** *f*=(*powr*)] *of-nat-nat refl*) *simp*
also have ... ≤ *2 powr* (*log 2* (*C*$_4$ / ε^2) + *1*)
by (*intro powr-mono*) *auto*
also have ... = *2* ∗ *C*$_4$ / ε^2
using *ε-gt-0* **unfolding** *powr-add C*$_4$*-def*
by (*subst powr-log-cancel*) (*auto intro:divide-pos-pos*)
finally have *7*:*b* ≤ *2* ∗ *C*$_4$ / ε^2 **by** *simp*

have *l* ≤ *C*$_6$ ∗ *ln* (*1* / δ) + *C*$_6$ ∗ *ln 2* + *1*
by (*intro l-ubound*)
also have ... ≤ *4* ∗ *ln*(*1*/δ) + *3*+*1*
unfolding *C*$_6$*-def* **by** (*intro add-mono order.refl*) (*approximation 5*)
also have ... = *4* ∗ (*ln*(*1*/δ)+*1*)
by *simp*
finally have *8*:*l* ≤ *4* ∗ (*ln*(*1*/δ)+*1*)
by *simp*

have ε^2 = *0* + ε^2
by *simp*
also have ... ≤ *ln* (*1* / δ) + *1*
using *δ-gt-0 δ-lt-1 ε-gt-0 ε-lt-1*
by (*intro add-mono power-le-one*) *auto*
finally have *9*: ε^2 ≤ *ln* (*1* / δ) + *1*
by *simp*

have *10*: *0* ≤ *ln* (*1* / δ) + *1*
using *δ-gt-0 δ-lt-1* **by** (*intro add-nonneg-nonneg*) *auto*

have *?L* = *bit-count* (*T*$_e$ (τ$_2$ ω *A* (*q* ω *A*))) + *bit-count* (*Nb*$_e$ (*nat* ⌈*log 2* (*real n*)⌉+*3*) (*q* ω *A*))
unfolding *encode-state-def τ-def* τ$_3$*-def* **by** (*simp add:dependent-bit-count*)
also have ...=*bit-count*(*T*$_e$(τ$_2$ ω *A* (*q* ω *A*)))+*ereal* (*1*+ *of-int*⌊*log 2* (*2* + *real* (*nat* ⌈*log 2 n*⌉))⌋)
using *max-s-3*[*OF assms*] **by** (*subst bounded-nat-bit-count-2*)
(*simp-all add:numeral-eq-Suc le-imp-less-Suc floorlog-def*)
also have ... = *bit-count*(*T*$_e$(τ$_2$ ω *A* (*q* ω *A*)))+*ereal* (*1*+ *of-int*⌊*log 2* (*2* + *of-int* ⌈*log 2 n*⌉)⌋)
using *0* **by** *simp*
also have ... ≤ *bit-count*(*T*$_e$(τ$_2$ ω *A* (*q* ω *A*)))+*ereal* (*1*+*log 2* (*2* + *of-int* ⌈*log 2 n*⌉))
by (*intro add-mono ereal-mono*) *simp-all*
also have ... ≤ *bit-count*(*T*$_e$(τ$_2$ ω *A* (*q* ω *A*)))+*ereal* (*1*+*log 2* (*2* + (*log 2 n* + *1*)))
using *0 n-gt-0* **by** (*intro add-mono ereal-mono iffD2*[*OF log-le-cancel-iff*] *add-pos-nonneg*) *auto*
also have ... = *bit-count*(*T*$_e$(τ$_2$ ω *A* (*q* ω *A*)))+*ereal* (*1*+*log 2* (*log 2 n* + *3*))
by (*simp add:ac-simps*)
also have ... ≤ *ereal* ((*2* ∗ *C*$_5$ + *1*) ∗ *b* ∗ *l*) + *ereal* (*1*+*log 2* (*log 2 n* + *3*))
by (*intro add-mono 5*) *auto*
also have ... = (*2* ∗ *C*$_5$ + *1*) ∗ *real b* ∗ *real l* + *log 2* (*log 2 n* + *3*) + *1*
by *simp*
also have ... ≤ (*2* ∗ *C*$_5$ + *1*) ∗ (*2* ∗ *C*$_4$ / ε^2) ∗ *real l* + *log 2* (*log 2 n* + *3*) + *1*
unfolding *C*$_5$*-def*
by (*intro ereal-mono mult-right-mono mult-left-mono add-mono 7*) *auto*
also have ... = (*4* ∗ *of-int C*$_5$+*2*)∗*C*$_4$∗*real l*/ ε^2 + *log 2* (*log 2 n* + *3*) + *1*
by *simp*
also have ... ≤ (*4* ∗ *of-int C*$_5$+*2*)∗*C*$_4$∗(*4*∗(*ln*(*1*/ δ)+*1*))/ ε^2 + *log 2* (*log 2 n* + *3*) + *1*
using *ε-gt-0* **unfolding** *C*$_5$*-def C*$_4$*-def*
by (*intro ereal-mono add-mono order.refl divide-right-mono mult-left-mono 8*) *auto*
also have ... = ((*2*∗*33*+*1*)∗*9*∗*2*^*26*)∗(*ln*(*1*/ δ)+*1*)/ ε^2 + *log 2* (*log 2 n* + *3*) + *1*
unfolding *C*$_5$*-def C*$_4$*-def* **by** *simp*
also have ... ≤ (*2*^*36*−*1*) ∗ (*ln*(*1*/δ)+*1*)/ ε^2 + *log 2* (*log 2 n* + *3*) + (*ln* (*1*/ δ)+*1*)/ ε^2

```
    using ε-gt-0 δ-gt-0 ε-lt-1 9 10
    by (intro add-mono ereal-mono divide-right-mono mult-right-mono mult-left-mono) simp-all
  also have ... = 2^36* (ln(1/δ)+1)/ ε^2 + log 2 (log 2 n + 3)
    by (simp add:divide-simps)
  finally show ?thesis
    by simp
qed

lemma random-bit-count:
  pro-size Ω ≤ 2 powr (4 * log 2 n + 48 * (log 2 (1 / ε) + 16)^2 + (55 + 60 * ln (1 / δ))^3)
  (is ?L ≤ ?R)
proof −
  have 1:log 2 (real n) ≥ 0
    using n-gt-0 by simp
  hence 0: − 1 < log 2 (real n)
    by simp

  have 10: log 2 C_4 ≤ 27
    unfolding C_4-def by (approximation 10)
  have ε^2 ≤ 1
    using ε-gt-0 ε-lt-1 by (intro power-le-one) auto
  also have ... ≤ C_4
    unfolding C_4-def by simp
  finally have ε^2 ≤ C_4 by simp
  hence 9: 0 ≤ log 2 (C_4 / ε^2)
    using ε-gt-0 unfolding C_4-def
    by (intro iffD2[OF zero-le-log-cancel-iff]) simp-all
  hence 2: − 1 < log 2 (C_4 / ε^2)
    by simp

  have 3: 0 < C_7 * b^2 unfolding C_7-def using b-min by (intro Rings.mult-pos-pos) auto

  have 0 ≤ log 2 (real C_7) + real (b-exp * 2)
    unfolding C_7-def by (intro add-nonneg-nonneg) auto
  hence 4: −1 < log 2 (real C_7) + real (b-exp * 2) by simp
  have (2, n-exp) = split-power (pro-size (G n-exp))
    unfolding geom-pro-size by (intro split-power-prime[symmetric] n-exp-gt-0) auto
  hence real (pro-size Ψ_1) = real (2 ^ (2 * max n-exp (nat ⌈log (real 2) (real n)⌉)))
    by (intro arg-cong[where f=real] hash-pro-size'[OF Ψ_1 n-gt-0])
  also have ... = 2 ^ (2 * max n-exp (nat ⌈log 2 (real n)⌉)) by simp
  also have ... = 2 ^ (2 * max 1 (nat ⌈log 2 (real n)⌉)) unfolding n-exp-def by simp
  also have ... ≤ 2 powr (2 * max (nat ⌈log 2 (real n)⌉) 1)
    by (subst powr-realpow) auto
  also have ... = 2 powr (2 * max (real (nat ⌈log 2 (real n)⌉)) 1)
    using n-gt-0 unfolding of-nat-mult of-nat-max by simp
  also have ... = 2 powr (2 * max (of-int ⌈log 2 (real n)⌉) 1)
    using 0 by (subst of-nat-nat) simp-all
  also have ... ≤ 2 powr (2 * max (log 2 (real n) + 1) 1)
    by (intro powr-mono mult-left-mono max-mono) auto
  also have ... = 2 powr (2 * (log 2 (real n) + 1))
    using 1 by (subst max-absorb1) auto
  finally have 5:real (pro-size Ψ_1) ≤ 2 powr (2 * log 2 n + 2)
    by simp

  have (2, 5 + b-exp * 2) = split-power (2^(5+b-exp*2))
    by (intro split-power-prime[symmetric]) auto
  also have ... = split-power (C_7 * b^2)
    unfolding C_7-def b-def power-mult[symmetric] power-add by simp
```

also have ... = *split-power* (*pro-size* ($\mathcal{N}$ (C_7 * b^2)))
  unfolding C_7*-def b-def* **by** (*subst nat-pro-size*) *auto*
finally have (*2*, *5* + *b-exp* * *2*) = *split-power* (*pro-size* ($\mathcal{N}$ (C_7 * b^2))) **by** *simp*
hence *real* (*pro-size* Ψ_2) = *real* (*2* ^ (*2* * *max* (*5* + *b-exp* * *2*) (*nat* ⌈*log* (*real 2*) (*real n*)⌉)))
  by (*intro arg-cong*[**where** *f*=*real*] *hash-pro-size'*[*OF* Ψ_2 *n-gt-0*])
also have ... = *2* ^ (*max* (*5* + *b-exp* * *2*) (*nat* ⌈*log 2* (*real n*)⌉) * *2*) **by** *simp*
also have ... ≤ *2* ^ (((*5* + *b-exp* * *2*) + (*nat* ⌈*log 2* (*real n*)⌉)) * *2*)
  by (*intro power-increasing mult-right-mono*) *auto*
also have ... = *2 powr* ((*5* + *b-exp* * *2* + *real* (*nat* ⌈*log 2* (*real n*)⌉))*2)
  by (*subst powr-realpow*[*symmetric*]) *auto*
also have ... = *2 powr* ((*5* + *of-int b-exp* * *2* + *of-int* ⌈*log 2* (*real n*)⌉)*2)
  using *0* **by** (*subst of-nat-nat*) *auto*
also have ... ≤ *2 powr* ((*5* + *of-int b-exp* * *2* + (*log 2* (*real n*) + *1*))*2)
  by (*intro powr-mono mult-right-mono add-mono*) *simp-all*
also have ... = *2 powr* (*12* + *4* * *real*(*nat* ⌈*log 2* (C_4 / ε^2)⌉) + *log 2* (*real n*) * *2*)
  unfolding *b-exp-def* **by** (*simp add*:*ac-simps*)
also have ... = *2 powr* (*12* + *4* * *real-of-int* ⌈*log 2* (C_4 / ε^2)⌉ + *log 2* (*real n*) * *2*)
  using *2* **by** (*subst of-nat-nat*) *simp-all*
also have ... ≤ *2 powr* (*12* + *4* * (*log 2* (C_4 / ε^2) + *1*) + *log 2* (*real n*) * *2*)
  by (*intro powr-mono add-mono order.refl mult-left-mono*) *simp-all*
also have ... = *2 powr* (*2* * *log 2 n* + *4* * *log 2* (C_4 / ε^2) + *16*)
  by (*simp add*:*ac-simps*)
finally have *6*:*real* (*pro-size* Ψ_2) ≤ *2 powr* (*2* * *log 2 n* + *4* * *log 2* (C_4 / ε^2) + *16*)
  by *simp*
have (*2*, *b-exp*) = *split-power* (*2* ^ *b-exp*)
  using *b-exp-ge-26* **by** (*intro split-power-prime*[*symmetric*]) *auto*
also have ... = *split-power* (*pro-size* ($\mathcal{N}$ *b*))
  unfolding *b-def* **by** (*subst nat-pro-size*) *auto*
finally have (*2*, *b-exp*) = *split-power* (*pro-size* ($\mathcal{N}$ *b*)) **by** *simp*
hence *real* (*pro-size* Ψ_3) = *real* (*2* ^ (*k* * *max b-exp* (*nat* ⌈*log* (*real 2*) (*real* (C_7*b^2))⌉)))
  by (*intro arg-cong*[**where** *f*=*real*] *hash-pro-size'*[*OF* Ψ_3]) (*simp-all add*:C_7*-def b-def*)
also have ... = *2* ^ (*k* * *max b-exp* (*nat* ⌈*log 2* (*real* C_7 * (*2* ^ (*b-exp*2*)))⌉))
  unfolding *b-def power-mult* **by** *simp*
also have ... = *2* ^ (*max b-exp* (*nat* ⌈*log 2* C_7 + *log 2* (*2* ^ (*b-exp*2*))⌉) * *k*)
  unfolding C_7*-def* **by** (*subst log-mult*) *simp-all*
also have ... = *2* ^ (*max b-exp* (*nat* ⌈*log 2* C_7 + (*b-exp*2*)⌉) * *k*)
  by (*subst log-nat-power*) *simp-all*
also have ... = *2 powr* (*max* (*real b-exp*) (*real* (*nat* ⌈*log 2* C_7 + (*b-exp*2*)⌉)) * *real k*)
  by (*subst powr-realpow*[*symmetric*]) *simp-all*
also have ... = *2 powr* (*max* (*real b-exp*) (*of-int* ⌈*log 2* C_7 + (*b-exp*2*)⌉) * *real k*)
  using *4* **by** (*subst of-nat-nat*) *simp-all*
also have ... ≤ *2 powr* (*max* (*real b-exp*) (*log 2* C_7 + *real b-exp*2 +1*) * *real k*)
  by (*intro powr-mono mult-right-mono max-mono-2*) *simp-all*
also have ... = *2 powr* ((*log 2* (*2*^*5*) + *real b-exp*2 +1*) * *real k*)
  unfolding C_7*-def* **by** (*subst max-absorb2*) *simp-all*
also have ... = *2 powr* ((*real b-exp*2 +6*) * *real k*)
  unfolding C_7*-def* **by** (*subst log-nat-power*) (*simp-all add*:*ac-simps*)
also have ... = *2 powr* ((*of-int* ⌈*log 2* (C_4 / ε^2)⌉ * *2* + *6*) * *real k*)
  using *2* **unfolding** *b-exp-def* **by** (*subst of-nat-nat*) *simp-all*
also have ... ≤ *2 powr* (((*log 2* (C_4 / ε^*2*)+*1*) * *2* + *6*) * *real k*)
  by (*intro powr-mono mult-right-mono add-mono*) *simp-all*
also have ... = *2 powr* ((*log 2* (C_4 / ε^2) * *2* + *8*) * *real k*)
  by (*simp add*:*ac-simps*)
finally have *7*:*real* (*pro-size* Ψ_3) ≤ *2 powr* ((*log 2* (C_4 / ε^2) * *2* + *8*) * *real k*)
  by *simp*

have *ln* (*real b*) ≥ *0*
  using *b-min* **by** *simp*

hence *real k = of-int ⌈7.5 * ln (real b) + 16⌉*
unfolding *k-def* C_2*-def* C_3*-def* **by** (*subst of-nat-nat*) *simp-all*
also have *... ≤ (7.5 * ln (real b) + 16) + 1*
unfolding *b-def* **by** (*intro of-int-ceiling-le-add-one*)
also have *... = 7.5 * ln (2 powr b-exp) + 17*
unfolding *b-def* **using** *powr-realpow* **by** *simp*
also have *... = real b-exp * (7.5 * ln 2) + 17*
unfolding *powr-def* **by** *simp*
also have *... ≤ real b-exp * 6 + 17*
by (*intro add-mono mult-left-mono order.refl of-nat-0-le-iff*) (*approximation 5*)
also have *... = of-int ⌈log 2 (*C_4 */ ε²)⌉ * 6 + 17*
using *2* **unfolding** *b-exp-def* **by** (*subst of-nat-nat*) *simp-all*
also have *... ≤ (log 2 (*C_4 */ ε^2) + 1) * 6 + 17*
by (*intro add-mono mult-right-mono*) *simp-all*
also have *... = 6 * log 2 (*C_4 */ ε^2) + 23*
by *simp*
finally have *8:real k ≤ 6 * log 2 (*C_4 */ ε^2) + 23*
by *simp*

have *real (pro-size* Ψ*) = real (pro-size* Ψ_1*) * real (pro-size* Ψ_2*) * real (pro-size* Ψ_3*)*
unfolding Ψ*-def prod-pro-size* **by** *simp*
also have *... ≤*
*2 powr(2*log 2 n+2)*2 powr (2*log 2 n+4*log 2 (*C_4*/ε²)+16)*2 powr((log 2 (*C_4*/ε²)*2+8)*real k)*
by (*intro mult-mono 5 6 7 mult-nonneg-nonneg*) *simp-all*
also have *... = 2 powr (2*log 2 n + 2 + 2 * log 2 n+4*log 2 (*C_4*/ε²)+16+(log 2 (*C_4*/ε²)*2+8)*real k)*
unfolding *powr-add* **by** *simp*
also have *... = 2 powr (4*log 2 n + 4*log 2 (*C_4*/ε²) + 18 + (2*log 2 (*C_4*/ε²)+8)*real k)*
by (*simp add:ac-simps*)
also have *... ≤*
2 powr (4 log 2 n + 4* log 2 (*C_4*/ ε^2) + 18 + (2*log 2 (*C_4*/ε²)+8)*(6 * log 2 (*C_4 */ ε^2) + 23))*
using *9* **by** (*intro powr-mono add-mono order.refl mult-left-mono 8 add-nonneg-nonneg*) *simp-all*
also have *... = 2 powr (4 * log 2 n+12 * log 2 (*C_4 */ ε^2)^2 + 98 * log 2 (*C_4 */ ε^2)+202)*
by (*simp add:algebra-simps power2-eq-square*)
also have *... ≤ 2 powr (4 * log 2 n+12 * log 2 (*C_4 */ ε^2)^2 + 120 * log 2 (*C_4 */ ε^2)+300)*
using *9* **by** (*intro powr-mono add-mono order.refl mult-right-mono*) *simp-all*
also have *... = 2 powr (4 * log 2 n+12 * (log 2 (*C_4** (1/ ε)^2) + 5)^2)*
by (*simp add:power2-eq-square algebra-simps*)
also have *... = 2 powr (4 * log 2 n + 12 * (log 2* C_4 *+ log 2 ((1 / ε)^2) + 5)^2)*
unfolding C_4*-def* **using** *ε-gt-0* **by** (*subst log-mult*) *auto*
also have *... ≤ 2 powr (4 * log 2 n + 12 * (27 + log 2 ((1/ ε)^2) + 5)^2)*
using *ε-gt-0 ε-lt-1*
by (*intro powr-mono add-mono order.refl mult-left-mono power-mono add-nonneg-nonneg 10*) (*simp-all add:*C_4*-def*)
also have *... = 2 powr (4 * log 2 n + 12 * (2 * (log 2 (1 / ε) + 16))^2)*
using *ε-gt-0* **by** (*subst log-nat-power*) (*simp-all add:ac-simps*)
also have *... = 2 powr (4 * log 2 n + 48 * (log 2 (1 / ε) + 16)^2)*
unfolding *power-mult-distrib* **by** *simp*
finally have *19:real (pro-size* Ψ*) ≤ 2 powr (4 * log 2 n + 48 * (log 2 (1 / ε) + 16)^2)*
by *simp*

have *0 ≤ ln* Λ */ ln (19 / 20)*
using Λ*-gt-0* Λ*-le-1* **by** (*intro divide-nonpos-neg*) *simp-all*
hence *11: −1 < ln* Λ */ ln (19 / 20)* **by** *simp*

have *12*: *ln (19 / 20) ≤ −(0.05::real) − ln (1 / 16) ≤ (2.8::real)* **by** (*approximation 10*)+

have *13*: *ln l ≥ 0* **using** *l-gt-0* **by** *auto*

have *ln l^3 = 27 ∗ (0 + ln l/3)^3* **by** (*simp add*:*power3-eq-cube*)
also have *... ≤ 27 ∗ (1 + ln l/real 3)^3*
  using *l-gt-0* **by** (*intro mult-left-mono add-mono power-mono*) *auto*
also have *... ≤ 27 ∗ (exp (ln l))*
  using *l-gt-0 13* **by** (*intro mult-left-mono exp-ge-one-plus-x-over-n-power-n*) *linarith*+
also have *... = 27 ∗ real l* **using** *l-gt-0* **by** (*subst exp-ln*) *auto*
finally have *14*:*ln l^3 ≤ 27 ∗ real l* **by** *simp*

have *15*:C_6 *∗ ln (2 / δ) > 0*
  using *δ-lt-1 δ-gt-0* **unfolding** C_6*-def*
  by (*intro Rings.mult-pos-pos ln-gt-zero*) *auto*
hence *1 ≤ real-of-int ⌈*C_6 *∗ ln (2 / δ)⌉* **by** *simp*
hence *16*: *1 ≤ 3 ∗ real-of-int ⌈*C_6 *∗ ln (2 / δ)⌉* **by** *argo*

have *17*: *12 ∗ ln 2 ≤ (9::real)* **by** (*approximation 5*)

have *16 ^ ((l − 1) ∗ nat⌈ln Λ / ln 0.95⌉) = 16 powr (real (l−1)∗real(nat ⌈ln Λ / ln (19 / 20)⌉))*
  by (*subst powr-realpow*[*symmetric*]) *auto*
also have *... = 16 powr (real (l−1)∗ of-int ⌈ln Λ / ln (19 / 20)⌉)*
  using *11* **by** (*subst of-nat-nat*) *simp-all*
also have *... ≤ 16 powr (real (l−1)∗ (ln Λ / ln (19/20)+1))*
  by (*intro powr-mono mult-left-mono*) *auto*
also have *... = 16 powr ((real l − 1)∗(ln Λ / ln (19/20)+1))*
  using *l-gt-0* **by** (*subst of-nat-diff*) *auto*
also have *... ≤ 16 powr ((real l − 1)∗(ln Λ / (−0.05)+1))*
  using *l-gt-0 Λ-gt-0 Λ-le-1*
  by (*intro powr-mono mult-left-mono add-mono divide-left-mono-neg 12*) *auto*
also have *... = 16 powr ((real l − 1)∗(20 ∗ (−ln Λ)+1))*
  by (*simp add*:*algebra-simps*)
also have *... = 16 powr ((real l − 1)∗(20 ∗ −(min (ln (1/16)) (−l∗ln l^3))+1))*
  unfolding *Λ-def* **by** (*subst ln-min-swap*) *auto*
also have *... = 16 powr ((real l − 1)∗(20 ∗ max (−ln (1/16)) (l∗ln l^3)+1))*
  by (*intro-cong* [σ_2 (*powr*), σ_2(+), σ_2 (∗)]) *simp*
also have *... ≤ 16 powr ((real l − 1)∗(20 ∗ max (2.8) (l∗ln l^3)+1))*
  using *l-gt-0* **by** (*intro powr-mono mult-left-mono add-mono max-mono 12*) *auto*
also have *... ≤ 16 powr ((real l − 1)∗(20 ∗ (2.8+l∗ln l^3)+1))*
  using *l-gt-0* **by** (*intro powr-mono mult-left-mono add-mono*) *auto*
also have *... = 16 powr ((real l − 1)∗(20 ∗ (l∗ln l^3)+57))*
  by (*simp add*:*algebra-simps*)
also have *... ≤ 16 powr ((real l − 1)∗(20 ∗ (real l∗(27∗real l))+57))*
  using *l-gt-0* **by** (*intro powr-mono mult-left-mono add-mono 14*) *auto*
also have *... = 16 powr (540 ∗ real l^3 − 540 ∗ real l^2 + 57∗ real l − 57)*
  by (*simp add*:*algebra-simps numeral-eq-Suc*)
also have *... ≤ 16 powr (540 ∗ real l^3 − 540 ∗ real l^2 + 180∗ real l − 20)*
  by (*intro powr-mono add-mono diff-mono order.refl mult-right-mono*) *auto*
also have *... = 16 powr (20 ∗ (3∗real l − 1)^3)*
  by (*simp add*: *algebra-simps power3-eq-cube power2-eq-square*)
also have *... = 16 powr (20 ∗ (3 ∗ of-int ⌈*C_6 *∗ ln (2 / δ)⌉ − 1) ^ 3)*
  using *15* **unfolding** *l-def* **by** (*subst of-nat-nat*) *auto*
also have *... ≤ 16 powr (20 ∗ (3 ∗ (*C_6 *∗ ln (2 / δ) + 1) − 1) ^ 3)*
  using *16* **by** (*intro powr-mono mult-left-mono power-mono diff-mono*) *auto*
also have *... = 16 powr (20 ∗ (2 + 12 ∗ ln (2 ∗ (1 / δ))) ^ 3)*
  by (*simp add*:*algebra-simps* C_6*-def*)

```
  also have ... = (2 powr 4) powr (20 * (2+ 12 * (ln 2 + ln(1/ δ)))^3)
    using δ-gt-0 by (subst ln-mult) auto
  also have ... = 2 powr (80 * (2 + 12 * ln 2 + 12 * ln (1 / δ)) ^ 3)
    unfolding powr-powr by (simp add:ac-simps)
  also have ... ≤ 2 powr (80 * (2 + 9 + 12 * ln (1 / δ)) ^ 3)
    using δ-gt-0 δ-lt-1
    by (intro powr-mono mult-left-mono power-mono add-mono 17 add-nonneg-nonneg) auto
  also have ... = 2 powr (80 * (11 + 12 * ln (1 / δ)) ^ 3) by simp
  also have ... ≤ 2 powr (5^3 * (11 + 12 * ln (1 / δ)) ^ 3)
    using δ-gt-0 δ-lt-1 by (intro powr-mono mult-right-mono) (auto intro!:add-nonneg-nonneg)
  also have ... = 2 powr ((55 + 60 * ln (1 / δ))^3)
    unfolding power-mult-distrib[symmetric] by simp
  finally have 18:16^((l − 1) * nat⌈ln Λ / ln (19/20)⌉) ≤ 2 powr ((55 + 60 * ln (1 / δ))^3)
    by simp

  have ?L = real (pro-size Ψ) * 16 ^((l − 1) * nat ⌈ln Λ / ln (19 / 20)⌉)
    unfolding expander-pro-size[OF Ω] by simp
  also have ... ≤ 2 powr (4*log 2 n+48*(log 2 (1/ε)+16)^2)*2 powr ((55 + 60 * ln (1 / δ))^3)
    by (intro mult-mono 18 19) simp-all
  also have ... = 2 powr (4 * log 2 n + 48 * (log 2 (1 / ε) + 16)^2 + (55 + 60 * ln (1 / δ))^3)
    unfolding powr-add[symmetric] by simp
  finally show ?thesis by simp
qed

end

unbundle no intro-cong-syntax

end
```

7 Accuracy without cutoff

This section verifies that each of the l estimate have the required accuracy with high probability assuming that there was no cut-off, i.e., that $s = 0$. Section 9 will then show that this remains true as long as the cut-off is below $t\ f$ the subsampling threshold.

```
theory Distributed-Distinct-Elements-Accuracy-Without-Cutoff
  imports
    Concentration-Inequalities.Bienaymes-Identity
    Distributed-Distinct-Elements-Inner-Algorithm
    Distributed-Distinct-Elements-Balls-and-Bins
begin

no-notation Polynomials.var (‹X›)

locale inner-algorithm-fix-A = inner-algorithm +
  fixes A
  assumes A-range: A ⊆ {..<n}
  assumes A-nonempty: {} ≠ A
begin

definition X :: nat where X = card A

definition q-max where q-max = nat (⌈log 2 X⌉ − b-exp)

definition t :: (nat ⇒ nat) ⇒ int
  where t f = int (Max (f ‘ A)) − b-exp + 9
```

definition $s :: (nat \Rightarrow nat) \Rightarrow nat$
where $s\ f = nat\ (t\ f)$

definition $R :: (nat \Rightarrow nat) \Rightarrow nat\ set$
where $R\ f = \{a.\ a \in A \wedge f\ a \geq s\ f\}$

definition $r :: nat \Rightarrow (nat \Rightarrow nat) \Rightarrow nat$
where $r\ x\ f = card\ \{a.\ a \in A \wedge f\ a \geq x\}$

definition p **where** $p = (\lambda(f,g,h).\ card\ \{j \in \{..<b\}.\ \tau_1\ (f,g,h)\ A\ 0\ j \geq s\ f\})$

definition Y **where** $Y = (\lambda(f,g,h).\ 2\ \widehat{}\ s\ f * \varrho\text{-}inv\ (p\ (f,g,h)))$

lemma *fin-A*: *finite A*
using *A-range finite-nat-iff-bounded* **by** *auto*

lemma *X-le-n*: $X \leq n$
proof −
have *card A* ≤ *card* $\{..<n\}$
by (*intro card-mono A-range*) *simp*
thus *?thesis*
unfolding *X-def* **by** *simp*
qed

lemma *X-ge-1*: $X \geq 1$
unfolding *X-def*
using *fin-A A-nonempty* **by** (*simp add*: *leI*)

lemma *of-bool-square*: $(of\text{-}bool\ x)^2 = ((of\text{-}bool\ x)::real)$
by (*cases x*, *auto*)

lemma *r-eq*: $r\ x\ f = (\sum\ a \in A.(\ of\text{-}bool(\ x \leq f\ a) :: real))$
unfolding *r-def of-bool-def sum.If-cases*[*OF fin-A*]
by (*simp add*: *Collect-conj-eq*)

lemma
shows
r-exp: $(\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1) = real\ X * (of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)\ /\ 2\widehat{}x)$ **and**
r-var: *measure-pmf.variance* $\Psi_1\ (\lambda\omega.\ real\ (r\ x\ \omega)) \leq (\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1)$
proof −
define $V :: nat \Rightarrow (nat \Rightarrow nat) \Rightarrow real$ **where** $V = (\lambda a\ f.\ of\text{-}bool\ (x \leq f\ a))$

have *V-exp*: $(\int \omega.\ V\ a\ \omega\ \partial\Psi_1) = of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)/2\widehat{}x$
(**is** *?L* = *?R*) **if** $a \in A$ **for** *a*
proof −
have *a-le-n*: $a < n$
using *that A-range* **by** *auto*

have *?L* = $(\int \omega.\ indicator\ \{f.\ x \leq f\ a\}\ \omega\ \partial\ \Psi_1)$
unfolding *V-def* **by** (*intro integral-cong-AE*) *auto*
also have ... = *measure* (*map-pmf* $(\lambda\omega.\ \omega\ a)$ (*sample-pro* Ψ_1)) $\{f.\ x \leq f\}$
by *simp*
also have ... = *measure* ($\mathcal{G}$ *n-exp*) $\{f.\ x \leq f\}$
by (*subst hash-pro-component*[*OF* Ψ_1 *a-le-n*]) *auto*
also have ... = *of-bool* $(x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)/2\widehat{}x$
unfolding *geom-pro-prob n-exp-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

have b:$(\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1) = (\sum\ a \in A.\ (\int \omega.\ V\ a\ \omega\ \partial\Psi_1))$
unfolding *r-eq V-def* **by** (*intro Bochner-Integration.integral-sum*) *auto*
also have ... $= (\sum\ a \in A.\ \ of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)/2\hat{}x)$
using *V-exp* **by** (*intro sum.cong*) *auto*
also have ... $= X * (of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)\ /\ 2\hat{}x)$
using *X-def* **by** *simp*
finally show $(\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1) = real\ X * (of\text{-}bool\ (x \leq max\ (nat\ \lceil log\ 2\ n\rceil)\ 1)/\ 2\hat{}x)$
by *simp*

have $(\int \omega.\ (V\ a\ \omega)\hat{}2\ \partial\ \Psi_1) = (\int \omega.\ V\ a\ \omega\ \partial\ \Psi_1)$ **for** a
unfolding *V-def of-bool-square* **by** *simp*

hence a:*measure-pmf.variance* $\Psi_1\ (V\ a) \leq$ *measure-pmf.expectation* $\Psi_1\ (V\ a)$ **for** a
by (*subst measure-pmf.variance-eq*) *auto*

have $J \subseteq A \Longrightarrow card\ J = 2 \Longrightarrow$ *prob-space.indep-vars* Ψ_1 (λ-. *borel*) $V\ J$ **for** J
unfolding *V-def* **using** *A-range finite-subset*[*OF - fin-A*]
by (*intro prob-space.indep-vars-compose2*[**where** Y=$\lambda i\ y.\ of\text{-}bool(x \leq y)$ **and** M'=λ-. *discrete*]
hash-pro-indep[*OF* Ψ_1]) (*auto simp:prob-space-measure-pmf*)
hence *measure-pmf.variance* Ψ_1 ($\lambda\omega.\ real\ (r\ x\ \omega)$) = ($\sum\ a \in A.$ *measure-pmf.variance* $\Psi_1\ (V\ a)$)
unfolding *r-eq V-def* **by** (*intro measure-pmf.bienaymes-identity-pairwise-indep-2 fin-A*) *simp-all*
also have ... $\leq (\sum\ a \in A.\ (\int \omega.\ V\ a\ \omega\ \partial\ \Psi_1))$
by (*intro sum-mono a*)
also have ... $= (\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1)$
unfolding b **by** *simp*
finally show *measure-pmf.variance* Ψ_1 ($\lambda\omega.\ real\ (r\ x\ \omega)$) $\leq (\int \omega.\ real\ (r\ x\ \omega)\ \partial\ \Psi_1)$ **by** *simp*
qed

definition E_1 **where** $E_1 = (\lambda(f,g,h).\ 2\ powr\ (-t\ f) * X \in \{b/2\hat{}16..b/2\})$

lemma *t-low*:
measure Ψ_1 $\{f.\ of\text{-}int\ (t\ f) < log\ 2\ (real\ X) + 1 - b\text{-}exp\} \leq 1/2\hat{}7$ (**is** $?L \leq ?R$)
proof (*cases log 2* (*real X*) $\geq$ *8*)
case *True*
define Z :: (*nat* $\Rightarrow$ *nat*) $\Rightarrow$ *real* **where** $Z = r\ (nat\ \lceil log\ 2\ (real\ X) - 8\rceil)$

have *log 2* (*real X*) $\leq$ *log 2* (*real n*)
using *X-le-n X-ge-1* **by** (*intro log-mono*) *auto*
hence *nat* $\lceil log\ 2\ (real\ X) - 8\rceil \leq nat\ \lceil log\ 2\ (real\ n)\rceil$
by (*intro nat-mono ceiling-mono*) *simp*
hence a:(*nat* $\lceil log\ 2\ (real\ X) - 8\rceil \leq max\ (nat\ \lceil log\ 2\ (real\ n)\rceil)\ 1$)
by *simp*

have b:*real* (*nat* ($\lceil log\ 2\ (real\ X)\rceil - 8$)) $\leq log\ 2\ (real\ X) - 7$
using *True* **by** *linarith*

have $2\ \hat{}\ 7 = real\ X\ /\ (2\ powr\ (log\ 2\ X) * 2\ powr\ (-7))$
using *X-ge-1* **by** *simp*
also have ... $= real\ X\ /\ (2\ powr\ (log\ 2\ X - 7))$
by (*subst powr-add*[*symmetric*]) *simp*
also have ... $\leq real\ X\ /\ (2\ powr\ (real\ (nat\ \lceil log\ 2\ (real\ X) - 8\rceil)))$
using b **by** (*intro divide-left-mono powr-mono*) *auto*
also have ... $= real\ X\ /\ 2\ \hat{}\ nat\ \lceil log\ 2\ (real\ X) - 8\rceil$
by (*subst powr-realpow*) *auto*
finally have $2\ \hat{}\ 7 \leq real\ X\ /\ 2\ \hat{}\ nat\ \lceil log\ 2\ (real\ X) - 8\rceil$
by *simp*

```
  hence exp-Z-gt-2-7: (∫ ω. Z ω ∂Ψ1) ≥ 2^7
    using a unfolding Z-def r-exp by simp

  have var-Z-le-exp-Z: measure-pmf.variance Ψ1 Z ≤ (∫ ω. Z ω ∂Ψ1)
    unfolding Z-def by (intro r-var)

  have ?L ≤ measure Ψ1 {f. of-nat (Max (f ' A)) < log 2 (real X) − 8}
    unfolding t-def by (intro pmf-mono) (auto simp add:int-of-nat-def)
  also have ... ≤ measure Ψ1 {f ∈ space Ψ1. (∫ ω. Z ω ∂Ψ1) ≤ |Z f − (∫ ω. Z ω ∂Ψ1)|}
  proof (rule pmf-mono)
    fix f assume f ∈ set-pmf (sample-pro Ψ1)
    have fin-f-A: finite (f ' A) using fin-A finite-imageI by blast
    assume f ∈ {f. real (Max (f ' A)) < log 2 (real X) − 8}
    hence real (Max (f ' A)) < log 2 (real X) − 8 by auto
    hence real (f a) < log 2 (real X) − 8 if a ∈ A for a
      using Max-ge[OF fin-f-A] imageI[OF that] order-less-le-trans by fastforce
    hence of-nat (f a) < ⌈log 2 (real X) − 8⌉ if a ∈ A for a
      using that by (subst less-ceiling-iff) auto
    hence f a < nat ⌈log 2 (real X) − 8⌉ if a ∈ A for a
      using that True by fastforce
    hence r (nat ⌈log 2 (real X) − 8⌉) f = 0
      unfolding r-def card-eq-0-iff using not-less by auto
    hence Z f = 0
      unfolding Z-def by simp
    thus f ∈ {f ∈ space Ψ1. (∫ ω. Z ω ∂Ψ1) ≤ |Z f − (∫ ω. Z ω ∂Ψ1)|}
      by auto
  qed
  also have ... ≤ measure-pmf.variance Ψ1 Z / (∫ ω. Z ω ∂Ψ1)^2
    using exp-Z-gt-2-7 by (intro measure-pmf.second-moment-method) simp-all
  also have ... ≤ (∫ ω. Z ω ∂Ψ1) / (∫ ω. Z ω ∂Ψ1)^2
    by (intro divide-right-mono var-Z-le-exp-Z) simp
  also have ... = 1 / (∫ ω. Z ω ∂Ψ1)
    using exp-Z-gt-2-7 by (simp add:power2-eq-square)
  also have ... ≤ ?R
    using exp-Z-gt-2-7 by (intro divide-left-mono) auto
  finally show ?thesis by simp
next
  case False
  have ?L ≤ measure Ψ1 {f. of-nat (Max (f ' A)) < log 2 (real X) − 8}
    unfolding t-def by (intro pmf-mono) (auto simp add:int-of-nat-def)
  also have ... ≤ measure Ψ1 {}
    using False by (intro pmf-mono) simp
  also have ... = 0
    by simp
  also have ... ≤ ?R by simp
  finally show ?thesis by simp
qed

lemma t-high:
  measure Ψ1 {f. of-int (t f) > log 2 (real X) + 16 − b-exp} ≤ 1/2^7 (is ?L ≤ ?R)
proof −
  define Z :: (nat ⇒ nat) ⇒ real where Z = r (nat ⌊log 2 (real X) + 8⌋)

  have Z-nonneg: Z f ≥ 0 for f
    unfolding Z-def r-def by simp

  have (∫ ω. Z ω ∂Ψ1) ≤ real X / (2 ^ nat ⌊log 2 (real X) + 8⌋)
    unfolding Z-def r-exp by simp
```

```
  also have ... ≤ real X / (2 powr (real (nat ⌊log 2 (real X) + 8⌋)))
    by (subst powr-realpow) auto
  also have ... ≤ real X / (2 powr ⌊log 2 (real X) + 8⌋)
    by (intro divide-left-mono powr-mono) auto
  also have ... ≤ real X / (2 powr (log 2 (real X) + 7))
    by (intro divide-left-mono powr-mono, linarith) auto
  also have ... = real X / 2 powr (log 2 (real X)) / 2 powr 7
    by (subst powr-add) simp
  also have ... ≤ 1/2 powr 7
    using X-ge-1 by (subst powr-log-cancel) auto
  finally have Z-exp: (∫ ω. Z ω ∂Ψ₁) ≤ 1/2^7
    by simp

  have ?L ≤ measure Ψ₁ {f. of-nat (Max (f ' A)) > log 2 (real X) + 7}
    unfolding t-def by (intro pmf-mono) (auto simp add:int-of-nat-def)
  also have ... ≤ measure Ψ₁ {f. Z f ≥ 1}
  proof (rule pmf-mono)
    fix f assume f ∈ set-pmf (sample-pro Ψ₁)
    assume  f ∈ {f. real (Max (f ' A)) > log 2 (real X) + 7}
    hence real (Max (f ' A)) > log 2 (real X) + 7 by simp
    hence int (Max (f ' A)) ≥ ⌊log 2 (real X) + 8⌋
      by linarith
    hence Max (f ' A) ≥ nat ⌊log 2 (real X) + 8⌋
      by simp
    moreover have f ' A ≠ {} finite (f ' A)
      using fin-A finite-imageI A-nonempty by auto
    ultimately obtain fa where fa ∈ f ' A  fa ≥  nat ⌊log 2 (real X) + 8⌋
      using Max-in by auto
    then obtain ae where ae-def: ae ∈ A nat ⌊log 2 (real X) + 8⌋ ≤ f ae
      by auto
    hence r (nat ⌊log 2 (real X) + 8⌋) f > 0
      unfolding r-def card-gt-0-iff using fin-A by auto
    hence Z f ≥ 1
      unfolding Z-def by simp
    thus f ∈ {f. 1 ≤ Z f} by simp
  qed
  also have ... ≤ (∫ ω. Z ω ∂Ψ₁) / 1
    using Z-nonneg by (intro pmf-markov) auto
  also have ... ≤ ?R
    using Z-exp by simp
  finally show ?thesis by simp
qed

lemma e-1: measure Ψ {ψ. ¬E₁ ψ} ≤ 1/2^6
proof −
  have measure Ψ₁ {f. 2 powr (of-int (−t f)) * real X ∉ {real b/2^16..real b/2}} ≤
    measure Ψ₁ {f. 2 powr (of-int (−t f)) * real X < real b/2^16} +
    measure Ψ₁ {f. 2 powr (of-int (−t f)) * real X > real b/2}
    by (intro pmf-add) auto
  also have ... ≤ measure Ψ₁ {f. of-int (t f) > log 2 X + 16 − b-exp} +
                  measure Ψ₁ {f. of-int (t f) < log 2 X + 1 − b-exp}
  proof (rule add-mono)
    show measure Ψ₁ {f. 2 powr (of-int (−t f)) * real X < real b/2^16} ≤
    measure Ψ₁ {f. of-int (t f) > log 2 X + 16 − b-exp}
    proof (rule pmf-mono)
      fix f assume f ∈ {f. 2 powr real-of-int (−t f) * real X < real b / 2 ^ 16}
      hence 2 powr real-of-int (−t f) * real X < real b / 2 ^ 16
        by simp
```

```
    hence log 2 (2 powr of-int (−t f) * real X) < log 2 (real b / 2^16)
      using b-min X-ge-1 by (intro iffD2[OF log-less-cancel-iff]) auto
    hence of-int (−t f) + log  2 (real X) < log 2 (real b / 2^16)
      using X-ge-1 by (subst (asm) log-mult) auto
    also have  ... = real b-exp − log 2 (2 powr 16)
      unfolding b-def by (subst log-divide) auto
    also have ... = real b-exp − 16
      by (subst log-powr-cancel) auto
    finally have of-int (−t f) + log 2 (real X) < real b-exp − 16 by simp
    thus f ∈ {f. of-int (t f) > log 2 (real X) + 16 − b-exp}
      by simp
  qed
 next
  show measure Ψ1 {f. 2 powr of-int (−t f) * real X > real b/2} ≤
    measure Ψ1 {f. of-int (t f) < log 2 X + 1 − b-exp}
  proof (rule pmf-mono)
    fix f assume f ∈ {f. 2 powr real-of-int (−t f) * real X > real b / 2}
    hence 2 powr real-of-int (−t f) * real X > real b / 2
      by simp
    hence log 2 (2 powr of-int (−t f) * real X) > log 2 (real b / 2)
      using b-min X-ge-1 by (intro iffD2[OF log-less-cancel-iff]) auto
    hence of-int (−t f) + log  2 (real X) > log 2 (real b / 2)
      using X-ge-1 by (subst (asm) log-mult) auto
    hence  of-int (−t f) + log  2 (real X) > real b-exp − 1
      unfolding b-def by (subst (asm) log-divide) auto
    hence of-int (t f) < log 2 (real X) + 1 − b-exp
      by simp
    thus f ∈ {f. of-int (t f) < log 2 (real X) + 1 − b-exp}
      by simp
  qed
 qed
 also have ... ≤ 1/2^7 + 1/2^7
  by (intro add-mono t-low t-high)
 also have ... = 1/2^6 by simp
 finally have measure Ψ1 {f. 2 powr of-int (−t f) * real X ∉ {real b/2^16..real b/2}} ≤ 1/2^6
  by simp

 thus ?thesis
  unfolding sample-pro-Ψ E1-def case-prod-beta
  by (subst pair-pmf-prob-left)
qed

definition E2 where E2 = (λ(f,g,h). |card (R f) − X / 2^(s f)| ≤ ε/3 * X / 2^(s f))

lemma e-2: measure Ψ {ψ. E1 ψ ∧ ¬E2 ψ} ≤ 1/2^6 (is ?L ≤ ?R)
proof −
 define t_m :: int where t_m = ⌊log 2 (real X)⌋ + 16 − b-exp

 have t-m-bound: t_m ≤ ⌊log 2 (real X)⌋ − 10
  unfolding t_m-def using b-exp-ge-26 by simp

 have real b / 2^16 = (real X * (1/ X)) * (real b / 2^16)
  using X-ge-1 by simp
 also have ... = (real X * 2 powr (−log 2 X)) * (real b / 2^16)
  using X-ge-1 by (subst powr-minus-divide) simp
 also have ... ≤ (real X * 2 powr (− ⌊log 2 (real X)⌋)) * (2 powr b-exp / 2^16)
  unfolding b-def using powr-realpow
  by (intro mult-mono powr-mono) auto
```

also have ... = *real X* $*$ (*2 powr* ($-$ $\lfloor$*log 2* (*real X*)$\rfloor$) $*$ *2 powr*(*real b-exp*$-$*16*))
by (*subst powr-diff*) *simp*
also have ... = *real X* $*$ *2 powr* ($-$ $\lfloor$*log 2* (*real X*)$\rfloor$ + (*int b-exp* $-$ *16*))
by (*subst powr-add*[*symmetric*]) *simp*
also have ... = *real X* $*$ *2 powr* ($-t_m$)
unfolding t_m*-def* **by** (*simp add*:*algebra-simps*)
finally have *c*:*real b* / *2^16* $\leq$ *real X* $*$ *2 powr* ($-t_m$) **by** *simp*

define *T* :: *nat set* **where** *T* = {*x*. (*real X* / *2^x* $\geq$ *real b* / *2^16*)}

have *x* $\in$ *T* $\longleftrightarrow$ *int x* $\leq$ t_m **for** *x*
proof $-$
have *x* $\in$ *T* $\longleftrightarrow$ *2^ x* $\leq$ *real X* $*$ *2^16* / *b*
using *b-min* **by** (*simp add*: *field-simps T-def*)
also have ... $\longleftrightarrow$ *log 2* (*2^x*) $\leq$ *log 2* (*real X* $*$ *2^16* / *b*)
using *X-ge-1 b-min* **by** (*intro log-le-cancel-iff*[*symmetric*] *divide-pos-pos*) *auto*
also have ... $\longleftrightarrow$ *x* $\leq$ *log 2* (*real X* $*$ *2^16*) $-$ *log 2 b*
using *X-ge-1 b-min* **by** (*subst log-divide*) *auto*
also have ... $\longleftrightarrow$ *x* $\leq$ *log 2* (*real X*) + *log 2* (*2 powr 16*) $-$ *b-exp*
unfolding *b-def* **using** *X-ge-1* **by** (*subst log-mult*) *auto*
also have ... $\longleftrightarrow$ *x* $\leq$ $\lfloor$*log 2* (*real X*) + *log 2* (*2 powr 16*) $-$ *b-exp*$\rfloor$
by *linarith*
also have ... $\longleftrightarrow$ *x* $\leq$ $\lfloor$*log 2* (*real X*) + *16* $-$ *real-of-int* (*int b-exp*)$\rfloor$
by (*subst log-powr-cancel*) *auto*
also have ... $\longleftrightarrow$ *x* $\leq$ t_m
unfolding t_m*-def* **by** *linarith*
finally show *?thesis* **by** *simp*
qed
hence *T-eq*: *T* = {*x*. *int x* $\leq$ t_m} **by** *auto*

have *T* = {*x*. *int x* $<$ t_m+*1*}
unfolding *T-eq* **by** *simp*
also have ... = {*x*. *x* $<$ *nat* (t_m + *1*)}
unfolding *zless-nat-eq-int-zless* **by** *simp*
finally have *T-eq-2*: *T* = {*x*. *x* $<$ *nat* (t_m + *1*)}
by *simp*

have *inj-1*: *inj-on* ((−) (*nat* t_m)) *T*
unfolding *T-eq* **by** (*intro inj-onI*) *simp*
have *fin-T*: *finite T*
unfolding *T-eq-2* **by** *simp*

have *r-exp*: ($\int$ ω. *real* (*r t* ω) $\partial\Psi_1$) = *real X* / *2^t* **if** *t* $\in$ *T* **for** *t*
proof $-$
have *t* $\leq$ t_m
using *that* **unfolding** *T-eq* **by** *simp*
also have ... $\leq$ $\lfloor$*log 2* (*real X*)$\rfloor$ $-$ *10*
using *t-m-bound* **by** *simp*
also have ... $\leq$ $\lfloor$*log 2* (*real X*)$\rfloor$
by *simp*
also have ... $\leq$ $\lfloor$*log 2* (*real n*)$\rfloor$
using *X-le-n X-ge-1* **by** (*intro floor-mono log-mono*) *auto*
also have ... $\leq$ $\lceil$*log 2* (*real n*)$\rceil$
by *simp*
finally have *t* $\leq$ $\lceil$*log 2* (*real n*)$\rceil$ **by** *simp*
hence *t* $\leq$ *max* (*nat* $\lceil$*log 2* (*real n*)$\rceil$) *1***by** *simp*
thus *?thesis*
unfolding *r-exp* **by** *simp*

qed

have *r-var*: *measure-pmf.variance* Ψ_1 ($\lambda\omega$. *real* (*r t* ω)) $\leq$ *real X / 2^t* **if** $t \in T$ **for** t
using *r-exp*[*OF that*] *r-var* **by** *metis*

have *9* = C_4 / ε^2 * *ε^2/2^23*
using *ε-gt-0* **by** (*simp add*:C_4-*def*)
also have ... = *2 powr* (*log 2* (C_4 / ε^2)) * *ε^2/2^23*
using *ε-gt-0* C_4-*def* **by** (*subst powr-log-cancel*) *auto*
also have ... $\leq$ *2 powr b-exp* * *ε^2/2^23*
unfolding *b-exp-def*
by (*intro divide-right-mono mult-right-mono powr-mono*, *linarith*) *auto*
also have ... = *b* * *ε^2/2^23*
using *powr-realpow* **unfolding** *b-def* **by** *simp*
also have ... = (*b/2^16*) * (*ε^2/2^7*)
by *simp*
also have ... $\leq$ (*X* * *2 powr* ($-t_m$)) * (*ε^2/2^7*)
by (*intro mult-mono c*) *auto*
also have ... = *X* * (*2 powr* ($-t_m$) * *2 powr* (-7)) * *ε^2*
using *powr-realpow* **by** *simp*
also have ... = *2 powr* ($-t_m-7$) * (*ε^2* * *X*)
by (*subst powr-add*[*symmetric*]) (*simp*)
finally have *9* $\leq$ *2 powr* ($-t_m-7$) * (*ε^2* * *X*) **by** *simp*
hence *b*: *9/* (*ε^2* * *X*) $\leq$ *2 powr* ($-t_m$ -7)
using *ε-gt-0 X-ge-1*
by (*subst pos-divide-le-eq*) *auto*

have *a*: *measure* Ψ_1 {*f*.|*real* (*r t f*)−*real X/2^t*|> *ε/3* ∗*real X/2^t*} $\leq$ *2 powr* (*real t*−t_m−*7*)
(**is***?L1* $\leq$ *?R1*) **if** $t \in T$ **for** t
proof −
have *?L1* $\leq$ $\mathcal{P}$(*f in* Ψ_1. |*real* (*r t f*) − *real X / 2^t*| $\geq$ *ε/3* * *real X / 2^t*)
by (*intro pmf-mono*) *auto*
also have ... = $\mathcal{P}$(*f in* Ψ_1. |*real* (*r t f*)−($\int \omega$. *real* (*r t* ω) ∂ Ψ_1)| $\geq$ *ε/3* * *real X/2^t*)
by (*simp add*: *r-exp*[*OF that*])
also have ... $\leq$ *measure-pmf.variance* Ψ_1 ($\lambda\omega$. *real* (*r t* ω)) / (*ε/3* * *real X / 2^t*)^2
using *X-ge-1 ε-gt-0*
by (*intro measure-pmf.Chebyshev-inequality divide-pos-pos mult-pos-pos*) *auto*
also have ... $\leq$ (*X / 2^t*) / (*ε/3* * *X / 2^t*)^2
by (*intro divide-right-mono r-var*[*OF that*]) *simp*
also have ... = *2^t*∗(*9/* (*ε^2* * *X*))
by (*simp add*:*power2-eq-square algebra-simps*)
also have ... $\leq$ *2^t*∗(*2 powr* ($-t_m-7$))
by (*intro mult-left-mono b*) *simp*
also have ... = *2 powr t* * *2 powr* ($-t_m-7$)
by (*subst powr-realpow*[*symmetric*]) *auto*
also have ... = *?R1*
by (*subst powr-add*[*symmetric*]) (*simp add*:*algebra-simps*)
finally show *?L1* $\leq$ *?R1* **by** *simp*
qed

have $\exists$ *y*<*nat* (t_m + *1*). *x* = *nat* t_m − *y* **if** *x* < *nat* (t_m+*1*) **for** *x*
using *that* **by** (*intro exI*[**where** *x*=*nat* t_m − *x*]) *simp*
hence *T-reindex*: (−) (*nat* t_m) ‘ {*x*. *x* < *nat* (t_m + *1*)} = {..<*nat* (t_m + *1*)}
by (*auto simp add*: *set-eq-iff image-iff*)

have *?L* $\leq$ *measure* Ψ {ψ. ($\exists$ *t* $\in$ *T*. |*real* (*r t* (*fst* ψ))−*real X/2^t*| > *ε/3* * *real X / 2^t*)}
proof (*rule pmf-mono*)
fix ψ

assume $\psi \in$ *set-pmf* (*sample-pro* Ψ)
obtain $f\ g\ h$ **where** ψ*-def*: $\psi = (f,g,h)$ **by** (*metis prod-cases3*)
assume $\psi \in \{\psi.\ E_1\ \psi \wedge \neg\ E_2\ \psi\}$
hence *a*:*2 powr* ($-$*real-of-int* (*t f*)) $*$ *real X* $\in$ {*real b*/*2*^*16*..*real b*/*2*} **and**
b:|*card* (*R f*) $-$ *real X* / *2*^(*s f*)| $>$ ε/*3* $*$ *X* / *2*^(*s f*)
unfolding E_1*-def* E_2*-def* **by** (*auto simp add*:ψ*-def*)
have |*card* (*R f*) $-$ *X* / *2*^(*s f*)| = *0* **if** *s f*= *0*
using *that* **by** (*simp add*:*R-def X-def*)
moreover have (ε/*3*) $*$ (*X* / *2*^*s f*) $\geq$ *0*
using ε*-gt-0 X-ge-1* **by** (*intro mult-nonneg-nonneg*) *auto*
ultimately have *False* **if** *s f* = *0*
using *b that* **by** *simp*
hence *s f* $>$ *0* **by** *auto*
hence *t f* = *s f* **unfolding** *s-def* **by** *simp*
hence *2 powr* ($-$*real* (*s f*)) $*$ *X* $\geq$ *b* / *2*^*16*
using *a* **by** *simp*
hence *X* / *2 powr* (*real* (*s f*)) $\geq$ *b* / *2*^*16*
by (*simp add*: *divide-powr-uminus mult.commute*)
hence *real X* / *2* ^ (*s f*) $\geq$ *b* / *2*^*16*
by (*subst* (*asm*) *powr-realpow*, *auto*)
hence *s f* $\in$ *T* **unfolding** *T-def* **by** *simp*
moreover have |*r* (*s f*) *f* $-$ *X* / *2*^*s f*| $>$ ε/*3* $*$ *X* / *2*^*s f*
using *R-def r-def b* **by** *simp*
ultimately have $\exists$ *t* $\in$ *T*. |*r t* (*fst* ψ) $-$ *X* / *2*^*t*| $>$ ε/*3* $*$ *X* / *2*^*t*
using ψ*-def* **by** (*intro bexI*[**where** *x*=*s f*]) *simp*
thus $\psi \in \{\psi.\ (\exists\, t \in T.\ |r\ t\ (fst\ \psi) - X\ /\ 2\hat{}t| > \varepsilon/3 * X\ /\ 2\hat{}t)\}$ **by** *simp*
qed
also have ... = *measure* Ψ_1 {*f*. ($\exists$ *t* $\in$ *T*. |*real* (*r t f*)$-$*real X* / *2*^*t*| $>$ ε/*3* $*$ *real X*/*2*^*t*)}
unfolding *sample-pro*-Ψ **by** (*intro pair-pmf-prob-left*)
also have ... = *measure* Ψ_1 ($\bigcup$ *t* $\in$ *T*. {*f*. |*real* (*r t f*)$-$*real X* / *2*^*t*| $>$ ε/*3* $*$ *real X*/*2*^*t*})
by (*intro measure-pmf-cong*) *auto*
also have ... $\leq$ ($\sum$ *t* $\in$ *T*. *measure* Ψ_1 {*f*.|*real* (*r t f*)$-$*real X* / *2*^*t*| $>$ ε/*3* $*$ *real X*/*2*^*t*})
by (*intro measure-UNION-le fin-T*) (*simp*)
also have ... $\leq$ ($\sum$ *t* $\in$ *T*. *2 powr* (*real t* $-$ *of-int* t_m $-$ *7*))
by (*intro sum-mono a*)
also have ... = ($\sum$ *t* $\in$ *T*. *2 powr* ($-$*int* (*nat* $t_m$$-$*t*) $-$ *7*))
unfolding *T-eq*
by (*intro sum.cong refl arg-cong2*[**where** *f*=(*powr*)]) *simp*
also have ... = ($\sum$ *x* $\in$ (λ*x*. *nat* t_m $-$ *x*) ‘ *T*. *2 powr* ($-$*real x* $-$ *7*))
by (*subst sum.reindex*[*OF inj-1*]) *simp*
also have ... = ($\sum$ *x* $\in$ (λ*x*. *nat* t_m $-$ *x*) ‘ *T*. *2 powr* ($-$*7*) $*$ *2 powr* ($-$*real x*))
by (*subst powr-add*[*symmetric*]) (*simp add*:*algebra-simps*)
also have ... = *1*/*2*^*7* $*$ ($\sum$ *x* $\in$ (λ*x*. *nat* t_m $-$ *x*) ‘ *T*. *2 powr* ($-$*real x*))
by (*subst sum-distrib-left*) *simp*
also have ... = *1*/*2*^*7* $*$ ($\sum$ *x* $<$*nat* (t_m+*1*). *2 powr* ($-$*real x*))
unfolding *T-eq-2 T-reindex*
by (*intro arg-cong2*[**where** *f*=($*$)] *sum.cong*) *auto*
also have ... = *1*/*2*^*7* $*$ ($\sum$ *x* $<$*nat* (t_m+*1*). (*2 powr* ($-$*1*)) *powr* (*real x*))
by (*subst powr-powr*) *simp*
also have ... = *1*/*2*^*7* $*$ ($\sum$ *x* $<$*nat* (t_m+*1*). (*1*/*2*) ^ *x*)
using *powr-realpow* **by** *simp*
also have ... $\leq$ *1*/*2*^*7* $*$ *2*
by(*subst geometric-sum*) *auto*
also have ... = *1*/*2*^*6* **by** *simp*
finally show *?thesis* **by** *simp*
qed

definition E_3 **where** E_3 = (λ(*f*,*g*,*h*). *inj-on g* (*R f*))

lemma *R-bound*:
fixes *f g h*
assumes E_1 (*f*,*g*,*h*)
assumes E_2 (*f*,*g*,*h*)
shows *card* (*R f*) ≤ *2*/*3* ∗ *b*
proof −
have *real* (*card* (*R f*)) ≤ (*ε* / *3*) ∗ (*real X* / *2* ^ *s f*) + *real X* / *2* ^ *s f*
using *assms*(*2*) **unfolding** E_2*-def* **by** *simp*
also have ... ≤ (*1*/*3*) ∗ (*real X* / *2* ^ *s f*) + *real X* / *2* ^ *s f*
using *ε-lt-1* **by** (*intro add-mono mult-right-mono*) *auto*
also have ... = (*4*/*3*) ∗ (*real X* / *2 powr s f*)
using *powr-realpow* **by** *simp*
also have ... ≤ (*4*/*3*) ∗ (*real X* / *2 powr t f*)
unfolding *s-def*
by (*intro mult-left-mono divide-left-mono powr-mono*) *auto*
also have ... = (*4*/*3*) ∗ (*2 powr* (−(*of-int* (*t f*))) ∗ *real X*)
by (*subst powr-minus-divide*) *simp*
also have ... = (*4*/*3*) ∗ (*2 powr* (− *t f*) ∗ *real X*)
by *simp*
also have ... ≤ (*4*/*3*) ∗ (*b*/*2*)
using *assms*(*1*) **unfolding** E_1*-def*
by (*intro mult-left-mono*) *auto*
also have ... ≤ (*2*/*3*) ∗ *b* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *e-3*: *measure* Ψ {*ψ*. E_1 *ψ* ∧ E_2 *ψ* ∧ ¬E_3 *ψ*} ≤ *1*/*2*^*6* (**is** *?L* ≤ *?R*)
proof −
let *?α* = (λ(*z*,*x*,*y*) *f*. *z* < C_7∗*b*^*2* ∧ *x* ∈ *R f* ∧ *y* ∈ *R f* ∧ *x* < *y*)
let *?β* = (λ(*z*,*x*,*y*) *g*. *g x* = *z* ∧ *g y* = *z*)

have *β-prob*: *measure* Ψ_2 {*g*. *?β ω g*} ≤ (*1*/*real* (C_7∗*b*^*2*)^*2*)
if *?α ω f* **for** *ω f*
proof −
obtain *x y z* **where** *ω-def*: *ω* = (*z*,*x*,*y*) **by** (*metis prod-cases3*)
have *a*:*prob-space.indep-vars* Ψ_2 (λ*i*. *discrete*) (λ*x ω*. *ω x* = *z*) *I*
if *I* ⊆ {..<*n*} *card I* ≤ *2* **for** *I*
by (*intro prob-space.indep-vars-compose2*[*OF - hash-pro-indep*[*OF* Ψ_2]] *that*)
(*simp-all add*:*prob-space-measure-pmf*)

have *u* ∈ *R f* ⟹ *u* < *n* **for** *u*
unfolding *R-def* **using** *A-range* **by** *auto*
hence *b*: *x* < *n y* < *n card* {*x*, *y*} = *2*
using *that ω-def* **by** *auto*
have *c*: *z* < C_7∗*b*^*2* **using** *ω-def that* **by** *simp*

have *measure* Ψ_2 {*g*. *?β ω g*} = *measure* Ψ_2 {*g*. (∀ *ξ* ∈ {*x*,*y*}. *g ξ* = *z*)}
by (*simp add*:*ω-def*)
also have ... = (∏ *ξ* ∈ {*x*,*y*}. *measure* Ψ_2 {*g*. *g ξ* = *z*})
using *b* **by** (*intro measure-pmf.split-indep-events*[*OF refl*, **where** *I*={*x*,*y*}] *a*)
(*simp-all add*:*prob-space-measure-pmf*)
also have ... = (∏ *ξ* ∈ {*x*,*y*}. *measure* (*map-pmf* (λ*ω*. *ω ξ*) (*sample-pro* Ψ_2)) {*g*. *g* = *z*})
by (*simp add*:*vimage-def*)
also have ... = (∏ *ξ* ∈ {*x*,*y*}. *measure* (*𝒩* (C_7 ∗ *b*^*2*)) {*g*. *g*=*z*})
using *b hash-pro-component*[*OF* Ψ_2] **by** (*intro prod.cong*) *fastforce*+
also have ... = (∏ *ξ* ∈ {*x*,*y*}. *measure* (*pmf-of-set* {..<C_7 ∗ *b*^*2*}) {*z*})
by (*subst nat-pro*) (*simp-all add*:C_7*-def b-def*)

also have ... = (*measure* (*pmf-of-set* {..<C_7 * b^2}) {z})^2
using b **by** *simp*
also have ... ≤ (*1* /(C_7*b^2))^2
using c **by** (*subst measure-pmf-of-set*) *auto*
also have ... = (*1* /(C_7*b^2)^2)
by (*simp add*:*algebra-simps power2-eq-square*)
finally show *?thesis* **by** *simp*
qed

have α-*card*: *card* {ω. *?*α ω f} ≤ (C_7*b^2) * (*card* (*R* f) * (*card* (*R* f)−*1*)/*2*)
(**is** *?TL* ≤ *?TR*) **and** *fin-*α: *finite* {ω. *?*α ω f} (**is** *?T2*) **for** f
proof −
have *t1*: {ω. *?*α ω f} ⊆ {..<C_7*b^2} × {(x,y) ∈ *R* f × *R* f. x < y}
by (*intro subsetI*) *auto*
moreover have *card* ({..<C_7*b^2} × {(x,y) ∈ *R* f × *R* f. x < y}) = *?TR*
using *card-ordered-pairs'*[**where** *M*=*R* f]
by (*simp add*: *card-cartesian-product*)
moreover have *finite* (*R* f)
unfolding *R-def* **using** *fin-A finite-subset* **by** *simp*
hence *finite* {(x, y). (x, y) ∈ *R* f × *R* f ∧ x < y}
by (*intro finite-subset*[**where** *B*=*R* f × *R* f, *OF - finite-cartesian-product*]) *auto*
hence *t2*: *finite* ({..<C_7*b^2} × {(x,y) ∈ *R* f × *R* f. x < y})
by (*intro finite-cartesian-product*) *auto*
ultimately show *?TL* ≤ *?TR*
using *card-mono of-nat-le-iff* **by** (*metis* (*no-types*, *lifting*))
show *?T2*
using *finite-subset*[*OF t1 t2*] **by** *simp*
qed

have *?L* ≤ *measure* Ψ {(f,g,h). *card* (*R* f) ≤ b ∧ (∃ x y z. *?*α (x,y,z) f ∧ *?*β (x,y,z) g)}
proof (*rule pmf-mono*)
fix ψ **assume** b:ψ ∈ *set-pmf* (*sample-pro* Ψ)
obtain f g h **where** ψ-*def*:ψ = (f,g,h) **by** (*metis prod-cases3*)
have (f,g,h) ∈ *pro-set* Ψ **using** b ψ-*def* **by** *simp*
hence c:g x < C_7*b^2 **for** x
using *g-range* **by** *simp*

assume a:ψ ∈ {ψ. E_1 ψ ∧ E_2 ψ ∧ ¬ E_3 ψ}
hence *card* (*R* f) ≤ *2*/*3* * b
using *R-bound* ψ-*def* **by** *force*
moreover have ∃ a b. a ∈ *R* f ∧ b ∈ *R* f ∧ a ≠ b ∧ g a = g b
using a **unfolding** ψ-*def* E_3-*def inj-on-def* **by** *auto*
hence ∃ x y. x ∈ *R* f ∧ y ∈ *R* f ∧ x < y ∧ g x = g y
by (*metis not-less-iff-gr-or-eq*)
hence ∃ x y z. *?*α (x,y,z) f ∧ *?*β (x,y,z) g
using c **by** *blast*
ultimately show ψ ∈ {(f, g, h). *card* (*R* f) ≤ b ∧ (∃ x y z. *?*α (x,y,z) f ∧ *?*β (x,y,z) g)}
unfolding ψ-*def* **by** *auto*
qed
also have ... = (∫ f. *measure* (*pair-pmf* Ψ_2 Ψ_3)
{g. *card* (*R* f) ≤ b ∧ (∃ x y z. *?*α (x,y,z) f ∧ *?*β (x,y,z) (*fst* g))} ∂Ψ_1)
unfolding *sample-pro-*Ψ *split-pair-pmf* **by** (*simp add*: *case-prod-beta*)
also have
... = (∫ f. *measure* Ψ_2 {g. *card* (*R* f) ≤ b ∧ (∃ x y z. *?*α (x,y,z) f ∧ *?*β (x,y,z) g)} ∂Ψ_1)
by (*subst pair-pmf-prob-left*) *simp*
also have ... ≤ (∫ f. *1*/*real* (*2**C_7) ∂Ψ_1)
proof (*rule pmf-exp-mono*[*OF integrable-sample-pro integrable-sample-pro*])
fix f **assume** f ∈ *set-pmf* (*sample-pro* Ψ_1)

```
    show measure Ψ2 {g. card (R f) ≤ b ∧ (∃ x y z. ?α (x,y,z) f ∧ ?β (x,y,z) g)} ≤ 1 / real (2
* C7)
      (is ?L1 ≤ ?R1)
    proof (cases card (R f) ≤ b)
      case True
      have ?L1 ≤ measure Ψ2 (⋃ ω ∈ {ω. ?α ω f}. {g. ?β ω g})
        by (intro pmf-mono) auto
      also have ... ≤ (∑ω ∈ {ω. ?α ω f}. measure Ψ2 {g. ?β ω g})
        by (intro measure-UNION-le fin-α) auto
      also have ... ≤ (∑ω ∈ {ω. ?α ω f}. (1/real (C7*b^2)^2))
        by (intro sum-mono β-prob) auto
      also have ... = card {ω. ?α ω f} /(C7*b^2)^2
        by simp
      also have ... ≤ (C7*b^2) * (card (R f) * (card (R f)−1)/2) / (C7*b^2)^2
        by (intro α-card divide-right-mono) simp
      also have ... ≤ (C7*b^2) * (b * b / 2)  / (C7*b^2)^2
        unfolding C7-def using True
        by (intro divide-right-mono Nat.of-nat-mono mult-mono) auto
      also have ... = 1/(2*C7)
        using b-min by (simp add:algebra-simps power2-eq-square)
      finally show ?thesis by simp
    next
      case False
      then show ?thesis by simp
    qed
  qed
  also have ... ≤ 1/2^6
    unfolding C7-def by simp
  finally show ?thesis by simp
qed

definition E4 where E4 = (λ(f,g,h). |p (f,g,h) − ϱ (card (R f))| ≤  ε/12 * card (R f))

lemma e-4-h: 9 / sqrt b ≤ ε / 12
proof −
  have 108 ≤ sqrt (C4)
    unfolding C4-def by (approximation 5)
  also have ... ≤ sqrt( ε^2 * real b)
    using b-lower-bound ε-gt-0
    by (intro real-sqrt-le-mono) (simp add: pos-divide-le-eq algebra-simps)
  also have ... =  ε * sqrt b
    using ε-gt-0 by (simp add:real-sqrt-mult)
  finally have 108 ≤  ε * sqrt b  by simp
  thus ?thesis
    using b-min by (simp add:pos-divide-le-eq)
qed

lemma e-4: measure Ψ {ψ. E1 ψ ∧ E2 ψ ∧ E3 ψ ∧ ¬E4 ψ} ≤ 1/2^6 (is ?L ≤ ?R)
proof −
  have a: measure Ψ3 {h. E1 (f,g,h) ∧ E2 (f,g,h) ∧ E3 (f,g,h) ∧ ¬E4 (f,g,h)} ≤ 1/2^6
    (is ?L1 ≤ ?R1) if f ∈ set-pmf (sample-pro Ψ1) g ∈ set-pmf(sample-pro Ψ2)
    for f g
  proof (cases card (R f) ≤ b ∧ inj-on g (R f))
    case True

    have g-inj: inj-on g (R f)
      using True by simp
```

have *fin-R*: *finite* (*g* ‘ *R f*)
unfolding *R-def* **using** *fin-A*
by (*intro finite-imageI*) *simp*

interpret *B*:*balls-and-bins-abs g* ‘ *R f* {..<*b*}
using *fin-R b-ne* **by** *unfold-locales auto*

have *range g* ⊆ {..<$C_7 * b^2$}
using *g-range-1 that*(*2*) **by** *auto*
hence *g-ran*: *g* ‘ *R f* ⊆ {..<$C_7 * b^2$}
by *auto*

have *sample-pro* (*𝒩 b*) = *pmf-of-set* {..<*b*}
by (*intro nat-pro*) (*simp add*:*b-def*)
hence *map-pmf* (λω. ω *x*) (*sample-pro* (*ℋ k* ($C_7 * b^2$) (*𝒩 b*))) = *pmf-of-set* {..<*b*}
if *x* ∈ *g* ‘ *R f* **for** *x*
using *g-ran hash-pro-component*[*OF* Ψ_3 - *k-gt-0*] *that* **by** *auto*
moreover have *prob-space.k-wise-indep-vars* Ψ_3 *k* (λ-. *discrete*) (λ*x* ω. ω *x*) (*g* ‘ *R f*)
by (*intro prob-space.k-wise-indep-subset*[*OF* - - *hash-pro-k-indep*[*OF* Ψ_3]] *g-ran*
prob-space-measure-pmf)
ultimately have *lim-balls-and-bins*: *B.lim-balls-and-bins k* (*sample-pro* (*ℋ k* ($C_7 * b^2$) (*𝒩 b*)))
unfolding *B.lim-balls-and-bins-def* **by** *auto*

have *card-g-R*: *card* (*g* ‘ *R f*) = *card* (*R f*)
using *True card-image* **by** *auto*
hence *b-mu*: *ϱ* (*card* (*R f*)) = *B.μ*
unfolding *B.μ-def ϱ-def* **using** *b-min* **by** (*simp add*:*powr-realpow*)

have *card-g-le-b*: *card* (*g* ‘ *R f*) ≤ *card* {..<*b*}
unfolding *card-g-R* **using** *True* **by** *simp*

have *?L1* ≤ *measure* Ψ_3 {*h*. |*B.Y h* − *B.μ*| > *9* * *real* (*card* (*g* ‘ *R f*)) / *sqrt* (*card* {..<*b*})}
proof (*rule pmf-mono*)
fix *h* **assume** *h* ∈ {*h*. E_1 (*f*,*g*,*h*) ∧ E_2 (*f*,*g*,*h*) ∧ E_3 (*f*,*g*,*h*) ∧ ¬E_4 (*f*,*g*,*h*)}
hence *b*: |*p* (*f*,*g*,*h*) −*ϱ* (*card* (*R f*))| > *ε*/*12* * *card* (*R f*)
unfolding E_4*-def* **by** *simp*
assume *h* ∈ *set-pmf* (*sample-pro* Ψ_3)
hence *h-range*: *h x* < *b* **for** *x* **using** *h-range-1* **by** *simp*

have {*j* ∈ {..<*b*}. *int* (*s f*) ≤ τ_1 (*f*, *g*, *h*) *A* *0* *j*} =
{*j* ∈ {..<*b*}. *int* (*s f*) ≤ *max* (*Max* ({*int* (*f a*) |*a*. *a* ∈ *A* ∧ *h* (*g a*) = *j*} ∪ {−*1*})) (− *1*)}
unfolding τ_1*-def* **by** *simp*
also have ... = {*j* ∈ {..<*b*}. *int* (*s f*) ≤ *Max* ({*int* (*f a*) |*a*. *a* ∈ *A* ∧ *h* (*g a*) = *j*} ∪ {−*1*})}
using *fin-A* **by** (*subst max-absorb1*) (*auto intro*: *Max-ge*)
also have ... = {*j* ∈ {..<*b*}. (∃ *a* ∈ *R f*. *h* (*g a*) = *j*)}
unfolding *R-def* **using** *fin-A* **by** (*subst Max-ge-iff*) *auto*
also have ... = {*j*. ∃ *a* ∈ *R f*. *h* (*g a*) = *j*}
using *h-range* **by** *auto*
also have ... = (*h* ∘ *g*) ‘ (*R f*)
by (*auto simp add*:*set-eq-iff image-iff*)
also have ... = *h* ‘ (*g* ‘ (*R f*))
by (*simp add*:*image-image*)
finally have *c*:{*j* ∈ {..<*b*}. *int* (*s f*) ≤ τ_1 (*f*, *g*, *h*) *A* *0* *j*} = *h* ‘ (*g* ‘ *R f*)
by *simp*
have *9* * *real* (*card* (*g* ‘ *R f*)) / *sqrt* (*card* {..<*b*}) = *9*/ *sqrt b* * *real* (*card* (*R f*))
using *card-image*[*OF g-inj*] **by** *simp*
also have ... ≤ *ε*/*12* * *card* (*R f*)

```
      by (intro mult-right-mono e-4-h) simp
    also have ... < |B.Y h − B.μ|
      using b c unfolding B.Y-def p-def b-mu by simp
    finally show h ∈ {h. |B.Y h − B.μ| >  9 * real (card (g ' R f)) / sqrt (card {..<b})}
      by simp
  qed
  also have ... ≤ 1/2^6
    using k-min
    by (intro B.devitation-bound[OF card-g-le-b lim-balls-and-bins]) auto
  finally show ?thesis by simp
next
  case False
  have ?L1 ≤ measure Ψ3 {}
  proof (rule pmf-mono)
    fix h assume b:h ∈ {h. E1 (f, g, h) ∧ E2 (f, g, h) ∧ E3 (f, g, h) ∧ ¬ E4 (f, g, h)}
    hence card (R f) ≤ (2/3)*b
      by (auto intro!: R-bound[simplified])
    hence card (R f) ≤ b
      by simp
    moreover have inj-on g (R f)
      using b by (simp add:E3-def)
    ultimately have False using False by simp
    thus h ∈ {} by simp
  qed
  also have ... = 0 by simp
  finally show ?thesis by simp
qed

have ?L = (∫ f. (∫ g.
  measure Ψ3 {h. E1 (f,g,h) ∧ E2 (f,g,h) ∧ E3 (f,g,h) ∧ ¬E4 (f,g,h)} ∂Ψ2) ∂Ψ1)
  unfolding sample-pro-Ψ split-pair-pmf by simp
also have ... ≤ (∫ f. (∫ g.  1/2^6  ∂Ψ2) ∂Ψ1)
  using a by (intro integral-mono-AE AE-pmfI) simp-all
also have ... = 1/2^6
  by simp
finally show ?thesis by simp
qed

lemma ϱ-inverse: ϱ-inv (ϱ x) = x
proof −
  have a:1−1/b ≠ 0
    using b-min by simp

  have ϱ x = b * (1−(1−1/b) powr x)
    unfolding ϱ-def by simp
  hence ϱ x / real b = 1−(1−1/b) powr x by simp
  hence ln (1 − ϱ x / real b) = ln ((1−1/b) powr x) by simp
  also have ... = x * ln (1 − 1/ b)
    using a by (intro ln-powr)
  finally have ln (1 − ϱ x / real b) = x * ln (1− 1/ b)
    by simp
  moreover have ln (1−1/b) < 0
    using b-min by (subst ln-less-zero-iff) auto
  ultimately show ?thesis
    using ϱ-inv-def by simp
qed

lemma rho-mono:
```

```
  assumes x ≤ y
  shows ϱ x ≤ ϱ y
proof−
  have (1 − 1 / real b) powr y ≤ (1 − 1 / real b) powr x
    using b-min
    by (intro powr-mono-rev assms) auto
  thus ?thesis
    unfolding ϱ-def by (intro mult-left-mono) auto
qed

lemma rho-two-thirds: ϱ (2/3 ∗ b) ≤ 3/5 ∗b
proof −
  have 1/3 ≤ exp ( − 13 / 12::real )
    by (approximation 8)
  also have ... ≤ exp ( − 1 − 2 / real b )
    using b-min by (intro iffD2[OF exp-le-cancel-iff]) (simp add:algebra-simps)
  also have ... ≤ exp ( b ∗ (−(1/real b)−2∗(1/real b)^2))
    using b-min by (simp add:algebra-simps power2-eq-square)
  also have  ... ≤ exp ( b ∗ ln (1−1/real b))
    using b-min
    by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono ln-one-minus-pos-lower-bound) auto
  also have ... = exp ( ln ( (1−1/real b) powr b))
    using b-min by (subst ln-powr) auto
  also have ... = (1−1/real b) powr b
    using b-min by (subst exp-ln) auto
  finally have a:1/3 ≤ (1−1/real b) powr b by simp

  have 2/5 ≤ (1/3) powr (2/3::real)
    by (approximation 5)
  also have ... ≤ ((1−1/real b) powr b) powr (2/3)
    by (intro powr-mono2 a) auto
  also have ... = (1−1/real b) powr (2/3 ∗ real b)
    by (subst powr-powr) (simp add:algebra-simps)
  finally have 2/5 ≤ (1 − 1 / real b) powr (2 / 3 ∗ real b) by simp
  hence 1 − (1 − 1 / real b) powr (2 / 3 ∗ real b) ≤ 3/5
    by simp
  hence ϱ (2/3 ∗ b) ≤ b ∗ (3/5)
    unfolding ϱ-def by (intro mult-left-mono) auto
  thus ?thesis
    by simp
qed

definition ϱ-inv' :: real ⇒ real
  where ϱ-inv' x = −1 / (real b ∗ (1−x / real b) ∗ ln (1 − 1 / real b))

lemma ϱ-inv'-bound:
  assumes x ≥ 0
  assumes x ≤ 59/90∗b
  shows |ϱ-inv' x| ≤ 4
proof −
  have c:ln (1 − 1 / real b) < 0
    using b-min
    by (subst ln-less-zero-iff) auto
  hence d:real b ∗ (1 − x / real b) ∗ ln (1 − 1 / real b) < 0
    using b-min assms by (intro Rings.mult-pos-neg) auto

  have (1::real) ≤ 31/30 by simp
  also have ... ≤ (31/30) ∗ (b ∗ −(− 1 / real b))
```

```
    using b-min by simp
  also have ... ≤ (31/30) * (b * -ln (1 + (- 1 / real b)))
    using b-min
    by (intro mult-left-mono le-imp-neg-le  ln-add-one-self-le-self2) auto
  also have ... ≤ 3 * (31/90) * (- b * ln (1 - 1 / real b))
    by simp
  also have ... ≤ 3 * (1 - x / real b) * (- b * ln (1 - 1 / real b))
    using assms b-min pos-divide-le-eq[where c=b] c
    by (intro mult-right-mono mult-left-mono mult-nonpos-nonpos) auto
  also have ... ≤ 3 * (real b * (1 - x / real b) * (-ln (1 - 1 / real b)))
    by (simp add:algebra-simps)
  finally have 3 * (real b * (1 - x / real b) * (-ln (1 - 1 / real b))) ≥ 1 by simp
  hence 3 * (real b * (1 - x / real b) * ln (1 - 1 / real b)) ≤ -1 by simp
  hence ϱ-inv' x ≤ 3
    unfolding ϱ-inv'-def using d
    by (subst neg-divide-le-eq) auto
  moreover have ϱ-inv' x > 0
    unfolding ϱ-inv'-def using d by (intro divide-neg-neg) auto
  ultimately show ?thesis by simp
qed

lemma ϱ-inv':
  fixes x :: real
  assumes x < b
  shows DERIV ϱ-inv x :> ϱ-inv' x
proof -
  have DERIV (ln ∘ (λx. (1 - x / real b))) x :> 1 / (1-x / real b) * (0 -1/b)
    using assms b-min
    by (intro DERIV-chain DERIV-ln-divide DERIV-cdivide derivative-intros) auto
  hence DERIV ϱ-inv x :> (1 / (1-x / real b) * (-1/b)) / ln (1-1/real b)
    unfolding comp-def ϱ-inv-def by (intro DERIV-cdivide) auto
  thus ?thesis
    by (simp add:ϱ-inv'-def algebra-simps)
qed

lemma accuracy-without-cutoff:
  measure Ψ {(f,g,h). |Y (f,g,h) - real X| > ε * X ∨ s f < q-max} ≤ 1/2^4
  (is ?L ≤ ?R)
proof -
  have ?L ≤ measure Ψ {ψ. ¬E1 ψ ∨  ¬E2 ψ ∨  ¬E3 ψ ∨  ¬E4 ψ}
  proof (rule pmf-rev-mono)
    fix ψ assume ψ ∈ set-pmf (sample-pro Ψ)
    obtain f g h where ψ-def: ψ = (f,g,h) by (metis prod-cases3)

    assume ψ ∉ {ψ. ¬ E1 ψ ∨ ¬ E2 ψ ∨ ¬ E3 ψ ∨ ¬ E4 ψ}
    hence assms: E1 (f,g,h) E2 (f,g,h) E3 (f,g,h) E4 (f,g,h)
      unfolding ψ-def by auto

    define I :: real set where I = {0..59/90*b}

    have p (f,g,h) ≤ ϱ (card (R f)) + ε/12 * card (R f)
      using assms(4) E4-def unfolding abs-le-iff by simp
    also have ... ≤ ϱ(2/3*b) + 1/12* (2/3*b)
      using ε-lt-1 R-bound[OF assms(1,2)]
      by (intro add-mono rho-mono mult-mono) auto
    also have ... ≤ 3/5 * b + 1/18*b
      by (intro add-mono rho-two-thirds) auto
    also have ... ≤ 59/90 * b
```

by *simp*
finally have $p\ (f,g,h) \leq 59/90 * b$ **by** *simp*
hence *p-in-I*: $p\ (f,g,h) \in I$
unfolding *I-def* **by** *simp*

have $\varrho\ (card\ (R\ f)) \leq \varrho(2/3 * b)$
using *R-bound*[*OF assms*(1,2)]
by (*intro rho-mono*) *auto*
also have ... $\leq 3/5 * b$
using *rho-two-thirds* **by** *simp*
also have ... $\leq b * 59/90$ **by** *simp*
finally have $\varrho\ (card\ (R\ f)) \leq b * 59/90$ **by** *simp*
moreover have $(1 - 1\ /\ real\ b)\ powr\ (real\ (card\ (R\ f))) \leq 1\ powr\ (real\ (card\ (R\ f)))$
using *b-min* **by** (*intro powr-mono2*) *auto*
hence $\varrho\ (card\ (R\ f)) \geq 0$
unfolding *ϱ-def* **by** (*intro mult-nonneg-nonneg*) *auto*
ultimately have $\varrho\ (card\ (R\ f)) \in I$
unfolding *I-def* **by** *simp*

moreover have *interval I*
unfolding *I-def interval-def* **by** *simp*
moreover have $59\ /\ 90 * b < b$
using *b-min* **by** *simp*
hence *DERIV ϱ-inv x :> ϱ-inv' x* **if** $x \in I$ **for** x
using *that I-def* **by** (*intro ϱ-inv'*) *simp*
ultimately obtain ξ :: *real* **where** *ξ-def*: $\xi \in I$
ϱ-inv $(p(f,g,h)) - ϱ\text{-}inv\ (\varrho\ (card\ (R\ f))) = (p\ (f,g,h) - \varrho(card\ (R\ f))) * ϱ\text{-}inv'\ \xi$
using *p-in-I MVT-interval* **by** *blast*

have $|ϱ\text{-}inv(p\ (f,g,h)) - card\ (R\ f)| = |ϱ\text{-}inv(p\ (f,g,h)) - ϱ\text{-}inv(\varrho(card\ (R\ f)))|$
by (*subst ϱ-inverse*) *simp*
also have ... $= |(p\ (f,g,h) - \varrho\ (card\ (R\ f)))| * |ϱ\text{-}inv'\ \xi\ |$
using *ξ-def*(2) *abs-mult* **by** *simp*
also have ... $\leq |p\ (f,g,h) - \varrho\ (card\ (R\ f))| * 4$
using *ξ-def*(1) *I-def*
by (*intro mult-left-mono ϱ-inv'-bound*) *auto*
also have ... $\leq (\ \varepsilon/12 * card\ (R\ f)) * 4$
using *assms*(4) E_4*-def* **by** (*intro mult-right-mono*) *auto*
also have ... $= \varepsilon/3 * card\ (R\ f)$ **by** *simp*
finally have *b*: $|ϱ\text{-}inv(p\ (f,g,h)) - card\ (R\ f)| \leq \varepsilon/3 * card\ (R\ f)$ **by** *simp*

have $|ϱ\text{-}inv(p\ (f,g,h)) - X\ /\ 2\ \hat{}\ (s\ f)| \leq$
$|ϱ\text{-}inv(p\ (f,g,h)) - card\ (R\ f)| + |card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|$
by *simp*
also have ... $\leq \varepsilon/3 * card\ (R\ f) + |card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|$
by (*intro add-mono b*) *auto*
also have ... $= \varepsilon/3 * |X\ /\ 2\ \hat{}\ (s\ f) + (card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f))| +$
$|card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|$ **by** *simp*
also have ... $\leq \varepsilon/3 * (|X\ /\ 2\ \hat{}\ (s\ f)| + |card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|) +$
$|card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|$
using *ε-gt-0* **by** (*intro mult-left-mono add-mono abs-triangle-ineq*) *auto*
also have ... $\leq \varepsilon/3 * |X\ /\ 2\ \hat{}\ (s\ f)| + (1+ \varepsilon/3) * |card\ (R\ f) - X\ /\ 2\ \hat{}\ (s\ f)|$
using *ε-gt-0 ε-lt-1* **by** (*simp add:algebra-simps*)
also have ... $\leq \varepsilon/3 * |X\ /\ 2\ \hat{}\ s\ f| + (4/3) * (\ \varepsilon\ /\ 3 * real\ X\ /\ 2\ \hat{}\ s\ f)$
using *assms*(2) *ε-gt-0 ε-lt-1*
unfolding E_2*-def* **by** (*intro add-mono mult-mono*) *auto*
also have ... $= (7/9) * \varepsilon * real\ X\ /\ 2\hat{}s\ f$
using *X-ge-1* **by** (*subst abs-of-nonneg*) *auto*

```
    also have ... ≤ 1 * ε * real X / 2^s f
      using ε-gt-0 by (intro mult-mono divide-right-mono) auto
    also have ... =  ε * real X / 2^s f by simp
    finally have a:|ϱ-inv(p (f,g,h)) − X / 2 ^ (s f)| ≤ ε * X / 2 ^ (s f)
      by simp

    have |Y (f, g, h) − real X| = |2 ^ (s f)| * |ϱ-inv(p (f,g,h)) − real X / 2 ^ (s f)|
     unfolding Y-def by (subst abs-mult[symmetric]) (simp add:algebra-simps powr-add[symmetric])
    also have ... ≤ 2 ^ (s f) * (ε * X / 2 ^ (s f))
      by (intro mult-mono a) auto
    also have ... = ε * X
      by (simp add:algebra-simps powr-add[symmetric])
    finally have |Y (f, g, h) − real X| ≤ ε * X by simp
    moreover have 2 powr (⌈log 2 (real X)⌉ − t f) ≤ 2 powr b-exp (is ?L1 ≤ ?R1)
    proof −
      have ?L1 ≤ 2 powr (1 + log 2 (real X)− t f)
        by (intro powr-mono, linarith) auto
      also have ... = 2 powr 1 * 2 powr (log 2 (real X)) * 2 powr (− t f)
        unfolding powr-add[symmetric] by simp
      also have ... = 2 * (2 powr (−t f) * X)
        using X-ge-1 by simp
      also have ... ≤ 2 * (b/2)
        using assms(1) unfolding E1-def by (intro mult-left-mono) auto
      also have ... = b by simp
      also have ... = ?R1
        unfolding b-def by (simp add: powr-realpow)
      finally show ?thesis by simp
    qed
    hence ⌈log 2 (real X)⌉ − t f ≤ real b-exp
      unfolding not-less[symmetric] using powr-less-mono[where x=2] by simp
    hence s f ≥ q-max unfolding s-def q-max-def by (intro nat-mono) auto
    ultimately show ψ ∉ {(f, g, h). ε * X < |Y (f, g, h) − real X| ∨ s f < q-max}
      unfolding ψ-def by auto
  qed
  also have ... ≤
    measure Ψ {ψ. ¬E1 ψ ∨ ¬E2 ψ ∨ ¬E3 ψ} + measure Ψ {ψ. E1 ψ ∧ E2 ψ ∧ E3 ψ ∧ ¬E4 ψ}
    by (intro pmf-add) auto
  also have ... ≤ (measure Ψ {ψ. ¬E1 ψ ∨ ¬E2 ψ} + measure Ψ {ψ. E1 ψ ∧ E2 ψ ∧ ¬E3 ψ})
+ 1/2^6
    by (intro add-mono e-4 pmf-add) auto
  also have ... ≤ ((measure Ψ {ψ. ¬E1 ψ} + measure Ψ {ψ. E1 ψ ∧ ¬E2 ψ}) + 1/2^6) + 1/2^6
    by (intro add-mono e-3 pmf-add) auto
  also have ... ≤ ((1/2^6 + 1/2^6) + 1/2^6) + 1/2^6
    by (intro add-mono e-2 e-1) auto
  also have ... = ?R by simp
  finally show ?thesis by simp
qed

end

end
```

8 Cutoff Level

This section verifies that the cutoff will be below *q-max* with high probability. The result will be needed in Section 9, where it is shown that the estimates will be accurate for any cutoff below *q-max*.

theory *Distributed-Distinct-Elements-Cutoff-Level*
imports
Distributed-Distinct-Elements-Accuracy-Without-Cutoff
Distributed-Distinct-Elements-Tail-Bounds
begin

hide-const (**open**) *Quantum.Z*

unbundle *intro-cong-syntax*

lemma *mono-real-of-int*: *mono real-of-int*
unfolding *mono-def* **by** *auto*

lemma *Max-le-Sum*:
fixes $f :: 'a \Rightarrow int$
assumes *finite A*
assumes $\bigwedge a.\ a \in A \Longrightarrow f\ a \geq 0$
shows $Max\ (insert\ 0\ (f\ `\ A)) \leq (\sum a \in A\ .f\ a)$ (**is** *?L* $\leq$ *?R*)
proof (*cases A*$\neq${}})
case *True*

have *0*: $f\ a \leq (\sum a \in A\ .f\ a)$ **if** $a \in A$ **for** *a*
using *that assms* **by** (*intro member-le-sum*) *auto*

have $?L = max\ 0\ (Max\ (f\ `\ A))$
using *True assms*(*1*) **by** (*subst Max-insert*) *auto*
also have $... = Max\ (max\ 0\ `\ f\ `\ A)$
using *assms True* **by** (*intro mono-Max-commute monoI*) *auto*
also have $... = Max\ (f\ `\ A)$
unfolding *image-image* **using** *assms*
by (*intro arg-cong*[**where** *f*=*Max*] *image-cong*) *auto*
also have $... \leq ?R$
using *0 True assms*(*1*)
by (*intro iffD2*[*OF Max-le-iff*]) *auto*
finally show *?thesis* **by** *simp*
next
case *False*
hence $A = \{\}$ **by** *simp*
then show *?thesis* **by** *simp*
qed

context *inner-algorithm-fix-A*
begin

The following inequality is true for base e on the entire domain ($x > 0$). It is shown in *ln-add-one-self-le-self*. In the following it is established for base 2, where it holds for $x \geq 1$.

lemma *log-2-estimate*:
assumes $x \geq (1::real)$
shows $log\ 2\ (1+x) \leq x$
proof −
define *f* **where** $f\ x = x - log\ 2\ (1+\ x)$ **for** $x :: real$
define f' **where** $f'\ x = 1\ -\ 1/((x+1)*ln\ 2)$ **for** $x :: real$

have *0*:(*f has-real-derivative* $(f'\ x))\ (at\ x)$ **if** $x > 0$ **for** *x*
unfolding *f-def f'-def* **using** *that*
by (*auto intro*!: *derivative-eq-intros*)

have $f'\ x \geq 0$ **if** $1 \leq x$ **for** $x :: real$

```
  proof −
    have (1::real) ≤ 2∗ln 2
      by (approximation 5)
    also have ... ≤ (x+1)∗ln 2
      using that by (intro mult-right-mono) auto
    finally have 1 ≤ (x+1)∗ln 2 by simp
    hence 1/((x+1)∗ln 2) ≤ 1
      by simp
    thus ?thesis
      unfolding f'-def by simp
  qed

  hence ∃ y. (f has-real-derivative y) (at x) ∧ 0 ≤ y if x ≥ 1 for x :: real
    using that order-less-le-trans[OF exp-gt-zero]
    by (intro exI[where x=f' x] conjI 0) auto
  hence f 1 ≤ f x
    by (intro DERIV-nonneg-imp-nondecreasing[OF assms]) auto
  thus ?thesis
    unfolding f-def by simp
qed

lemma cutoff-eq-7:
  real X ∗ 2 powr (−real q-max) / b ≤ 1
proof −
  have real X = 2 powr (log 2 X)
    using X-ge-1 by (intro powr-log-cancel[symmetric]) auto
  also have ... ≤ 2 powr (nat ⌈log 2 X⌉)
    by (intro powr-mono) linarith+
  also have ... = 2 ^ (nat ⌈log 2 X⌉)
    by (subst powr-realpow) auto
  also have ... = real (2 ^ (nat ⌈log 2 (real X)⌉))
    by simp
  also have ... ≤ real (2 ^ (b-exp + nat (⌈log 2 (real X)⌉ − int b-exp)))
    by (intro Nat.of-nat-mono power-increasing) linarith+
  also have ... = b ∗ 2^q-max
    unfolding q-max-def b-def by (simp add: power-add)

  finally have real X ≤ b ∗ 2 ^ q-max by simp
  thus ?thesis
    using b-min
    unfolding powr-minus inverse-eq-divide
    by (simp add:field-simps powr-realpow)
qed

lemma cutoff-eq-6:
  fixes k
  assumes a ∈ A
  shows (∫ f. real-of-int (max 0 (int (f a) − int k)) ∂Ψ1) ≤ 2 powr (−real k) (is ?L ≤ ?R)
proof (cases k ≤ n-exp − 1)
  case True
  have a-le-n: a < n
    using assms A-range by auto

  have ?L = (∫ x. real-of-int (max 0 (int x − k)) ∂map-pmf (λx. x a) Ψ1)
    by simp
  also have ... = (∫ x. real-of-int (max 0 (int x − k)) ∂(G n-exp))
    by (subst hash-pro-component[OF Ψ1 a-le-n]) auto
  also have ... = (∫ x. max 0 (real x − real k) ∂(G n-exp))
```

unfolding *max-of-mono*[*OF mono-real-of-int*,*symmetric*] **by** *simp*
also have ... = ($\sum$ *x*≤*n-exp*. *max 0* (*real x* − *real k*) ∗ *pmf* (*G n-exp*) *x*)
using *geom-pro-range* **by** (*intro integral-measure-pmf-real*) *auto*
also have ... = ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x* − *real k*) ∗ *pmf* (*G n-exp*) *x*)
by (*intro sum.mono-neutral-cong-right*) *auto*
also have ... = ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x* − *real k*) ∗ *measure* (*G n-exp*) {*x*})
unfolding *measure-pmf-single* **by** *simp*
also have ... = ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x*−*real k*)∗(*measure* (*G n-exp*) ({ω. ω≥*x*}−{ω. ω≥(*x*+*1*)})))
by (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *measure-pmf-cong*) *auto*
also have ... = ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x*−*real k*)∗
(*measure* (*G n-exp*) {ω. ω≥*x*} − *measure* (*G n-exp*) {ω. ω≥(*x*+*1*)}))
by (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *measure-Diff*) *auto*
also have ... = ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x* − *real k*) ∗ (*1*/*2*^*x* − *of-bool*(*x*+*1*≤*n-exp*)/*2*^(*x*+*1*)))
unfolding *geom-pro-prob* **by** (*intro-cong* [σ_2 (∗), σ_2 (−), σ_2 (/)] *more*:*sum.cong*) *auto*
also have ... =
($\sum$ *x*=*k*+*1*..*n-exp*. (*real x*−*k*)/*2*^*x*) − ($\sum$ *x*=*k*+*1*..*n-exp*. (*real x*−*k*)∗ *of-bool*(*x*+*1*≤*n-exp*)/*2*^(*x*+*1*))
by (*simp add*:*algebra-simps sum-subtractf*)
also have ...=($\sum$ *x*=*k*+*1*..*n-exp*. (*real x*−*k*)/*2*^*x*)−($\sum$ *x*=*k*+*1*..*n-exp*−*1*. (*real x*−*k*)/*2*^(*x*+*1*))
by (*intro arg-cong2*[**where** *f*=(−)] *refl sum.mono-neutral-cong-right*) *auto*
also have ...=($\sum$ *x*=*k*+*1*..(*n-exp*−*1*)+*1*. (*real x*−*k*)/*2*^*x*)−($\sum$ *x*=*k*+*1*..*n-exp*−*1*. (*real x*−*k*)/*2*^(*x*+*1*))
using *n-exp-gt-0* **by** (*intro arg-cong2*[**where** *f*=(−)] *refl sum.cong*) *auto*
also have ...= ($\sum$ *x*∈*insert k* {*k*+*1*..*n-exp*−*1*}. (*real* (*x*+*1*)−*k*)/*2*^(*x*+*1*))−
($\sum$ *x*=*k*+*1*..*n-exp*−*1*. (*real x*−*k*)/*2*^(*x*+*1*))
unfolding *sum.shift-bounds-cl-nat-ivl* **using** *True*
by (*intro arg-cong2*[**where** *f*=(−)] *sum.cong*) *auto*
also have ... = *1*/*2*^(*k*+*1*)+($\sum$ *x*=*k*+*1*..*n-exp*−*1*. (*real* (*x*+*1*)−*k*)/*2*^(*x*+*1*)− (*real x*−*k*)/*2*^(*x*+*1*))
by (*subst sum.insert*) (*auto simp add*:*sum-subtractf*)
also have ... = *1*/*2*^(*k*+*1*)+($\sum$ *x*=*k*+*1*..*n-exp*−*1*. (*1*/*2*^(*x*+*1*)))
by (*intro arg-cong2*[**where** *f*=(+)] *sum.cong refl*) (*simp add*:*field-simps*)
also have ... = ($\sum$ *x*∈*insert k* {*k*+*1*..*n-exp*−*1*}. (*1*/*2*^(*x*+*1*)))
by (*subst sum.insert*) *auto*
also have ... = ($\sum$ *x*=*0*+*k*..(*n-exp*−*1*−*k*)+*k*. *1*/*2*^(*x*+*1*))
using *True* **by** (*intro sum.cong*) *auto*
also have ... = ($\sum$ *x*<*n-exp*−*k*. *1*/*2*^(*x*+*k*+*1*))
unfolding *sum.shift-bounds-cl-nat-ivl* **using** *True n-exp-gt-0* **by** (*intro sum.cong*) *auto*
also have ... = (*1*/*2*)^(*k*+*1*) ∗ ($\sum$ *x*<*n-exp*−*k*. (*1*/*2*)^*x*)
unfolding *sum-distrib-left power-add*[*symmetric*] **by** (*simp add*:*power-divide ac-simps*)
also have ... = (*1*/*2*)^(*k*+*1*) ∗ *2* ∗ (*1*−(*1* / *2*) ^(*n-exp* − *k*))
by (*subst geometric-sum*) *auto*
also have ... ≤ (*1*/*2*)^(*k*+*1*) ∗ *2* ∗ (*1*−*0*)
by (*intro mult-left-mono diff-mono*) *auto*
also have ... = (*1*/*2*)^*k*
unfolding *power-add* **by** *simp*
also have ... = *?R*
unfolding *powr-minus* **by** (*simp add*:*powr-realpow inverse-eq-divide power-divide*)
finally show *?thesis*
by *simp*
next
case *False*
hence *k-ge-n-exp*: *k* ≥ *n-exp*
by *simp*
have *a-lt-n*: *a* < *n*
using *assms A-range* **by** *auto*

have *?L* = ($\int$ *x*. *real-of-int* (*max 0* (*int x* − *k*)) ∂*map-pmf* (λ*x*. *x a*) Ψ_1)
by *simp*
also have ... = ($\int$ *x*. *real-of-int* (*max 0* (*int x* − *k*)) ∂(*G n-exp*))
by (*subst hash-pro-component*[*OF* Ψ_1 *a-lt-n*]) *auto*

```
  also have ... = (∫ x. real-of-int 0 ∂(𝒢 n-exp))
    using geom-pro-range k-ge-n-exp
    by (intro integral-cong-AE AE-pmfI iffD2[OF of-int-eq-iff] max-absorb1) force+
  also have ... = 0 by simp
  finally show ?thesis by simp
qed

lemma cutoff-eq-5:
  assumes x ≥ (−1 :: real)
  shows real-of-int ⌊log 2 (x+2)⌋ ≤ (real c+2) + max (x − 2^c) 0 (is ?L ≤ ?R)
proof −
  have 0: 1 ≤ 2 ^ 1 ∗ ln (2::real)
    by (approximation 5)

  consider (a) c = 0 ∧ x ≥ 2^c+1 | (b) c > 0 ∧ x ≥ 2^c+1 | (c) x ≤ 2^c+1
    by linarith
  hence log 2 (x+2) ≤ ?R
  proof (cases)
    case a
    have log 2 (x+2) = log 2 (1+(x+1))
      by (simp add:algebra-simps)
    also have ... ≤ x+1
      using a by (intro log-2-estimate) auto
    also have ... = ?R
      using a by auto
    finally show ?thesis by simp
  next
    case b
    have 0 < 0 + (1::real)
      by simp
    also have ... ≤ 2^c+(1::real)
      by (intro add-mono) auto
    also have ... ≤ x
      using b by simp
    finally have x-gt-0: x > 0
      by simp

    have log 2 (x+2) = log 2 ((x+2)/2^c) + c
      using x-gt-0 by (subst log-divide) auto
    also have ... = log 2 (1+(x+2−2^c)/2^c) + c
      by (simp add:divide-simps)
    also have ... ≤ (x+2−2^c)/2^c / ln 2 + c
      using b unfolding log-def
      by (intro add-mono divide-right-mono ln-add-one-self-le-self divide-nonneg-pos) auto
    also have ... = (x+2−2^c)/(2^c∗ln 2) + c
      by simp
    also have ... ≤ (x+2−2^c)/(2^1∗ln 2)+c
      using b by (intro add-mono divide-left-mono mult-right-mono power-increasing) simp-all
    also have ... ≤ (x+2−2^c)/1 + c
      using b by (intro add-mono divide-left-mono 0) auto
    also have ... ≤ (c+2) + max (x − 2^c) 0
      using b by simp
    finally show ?thesis by simp
  next
    case c
    hence log 2 (x+2) ≤ log 2 ((2^c+1)+2)
      using assms by (intro log-mono add-mono) auto
    also have ... = log 2 (2^c∗(1+3/2^c))
```

```
    by (simp add:algebra-simps)
  also have ... = c + log 2 (1+3/2^c)
    by (subst log-mult-pos) (auto intro:add-pos-nonneg)
  also have ... ≤ c + log 2 (1+3/2^0)
    by (intro add-mono log-mono divide-left-mono power-increasing add-pos-nonneg) auto
  also have ... = c + log 2 (2*2)
    by simp
  also have ... = real c + 2
    by (subst log-mult) auto
  also have ... ≤ (c+2) + max (x - 2^c) 0
    by simp
  finally show ?thesis
    by simp
 qed
 moreover have ⌊log 2 (x+2)⌋ ≤ log 2 (x+2)
    by simp
 ultimately show ?thesis using order-trans by blast
qed

lemma cutoff-level:
 measure Ω {ω. q ω A > q-max} ≤ δ/2 (is ?L ≤ ?R)
proof −
 have C1-est: C1 * l ≤ 30 * real l
   unfolding C1-def
   by (intro mult-right-mono of-nat-0-le-iff) (approximation 10)

 define Z where Z ω = (∑j<b. real-of-int ⌊log 2 (of-int (max (τ1 ω A q-max j) (−1)) + 2)⌋)
for ω
 define V where V ω = Z ω / real b − 3 for ω

 have 2:Z ψ ≤ real b*(real c+2) + of-int (∑a∈A. max 0 (int (fst ψ a) − q-max −2^c))
   (is ?L1 ≤ ?R1) if ψ ∈ sample-pro Ψ for c ψ
 proof −
   obtain f g h where ψ-def: ψ = (f,g,h)
     using prod-cases3 by blast

   have ψ-range: (f,g,h) ∈ sample-pro Ψ
     using that unfolding ψ-def by simp

   have − 1 − 2^c ≤ −1−(1::real)
     by (intro diff-mono) auto
   also have ... ≤ 0 by simp
   finally have − 1 − 2 ^ c ≤ (0::real) by simp
   hence aux3: max (−1−2^c) 0 = (0::real)
     by (intro max-absorb2)

   have − 1 − int q-max − 2 ^ c ≤ −1 − 0 − 1
     by (intro diff-mono) auto
   also have ... ≤ 0 by simp
   finally have − 1 − int q-max − 2 ^ c ≤ 0 by simp
   hence aux3-2: max 0 (− 1 − int q-max − 2 ^ c) = 0
     by (intro max-absorb1)

   have ?L1 ≤ (∑j<b. (real c+2) + max (real-of-int (max (τ1 ψ A q-max j) (− 1)) − 2^c) 0)
     unfolding Z-def by (intro sum-mono cutoff-eq-5) auto
   also have ... = (∑j<b. (real c+2)+max (τ0 ψ A j − q-max − 2^c) 0)
     unfolding τ1-def max-of-mono[OF mono-real-of-int,symmetric]
     by (intro-cong [σ2 (+)] more:sum.cong) (simp add:max-diff-distrib-left max.assoc aux3)
```

```
  also have ... = real b * (real c + 2) +
    of-int (∑j<b. (max 0 (Max (insert (− 1) {int (f a) |a. a ∈ A ∧ h (g a) = j})−q-max −
2^c)))
    unfolding ψ-def by (simp add:max.commute)
  also have ... = real b * (real c + 2) +
     of-int (∑j<b. max 0 (Max ((λx. x−q-max−2^c)‘(insert(−1){int (f a) |a. a ∈ A∧h(g
a)=j})))))
    using fin-A
     by (intro-cong [σ2 (+), σ1 of-int, σ2 max] more:sum.cong mono-Max-commute) (auto
simp:monoI)
  also have ... = real b * (real c + 2) +
    of-int(∑j<b. max 0(Max(insert(−1−q-max−2^c){int (f a)−q-max−2^c |a. a ∈ A ∧ h (g
a) = j})))
    by (intro-cong [σ2 (+), σ1 of-int, σ2 max, σ1 Max] more:sum.cong)  auto
  also have ... = real b * (real c + 2) + of-int
    (∑j<b. Max ((max 0) ‘(insert(−1−q-max−2^c){int (f a)−q-max−2^c |a. a ∈ A ∧ h (g a)
= j})))
    using fin-A by (intro-cong [σ2 (+), σ1 of-int] more:sum.cong mono-Max-commute)
     (auto simp add:monoI setcompr-eq-image)
  also have ... = real b * (real c + 2) +
    of-int (∑j<b. Max (insert 0 {max 0 (int (f a)−q-max−2^c) |a. a ∈ A ∧ h (g a) = j}))
    using aux3-2 by (intro-cong [σ2 (+), σ1 of-int, σ1 Max] more:sum.cong)
     (simp add:setcompr-eq-image image-image)
 also have ... ≤ b∗(real c+2)+ of-int(∑j<b. (∑ a|a∈A∧h(g(a))=j. max 0(int(f a)−q-max−2^c)))
    using fin-A Max-le-Sum unfolding setcompr-eq-image
    by (intro add-mono iffD2[OF of-int-le-iff] sum-mono Max-le-Sum) (simp-all)
  also have ... = real b∗(real c+2)+
    of-int(∑ a∈(⋃j∈{..<b}. {a. a∈A∧ h(g(a)) = j}). max 0(int(f a)−q-max−2^c))
    using fin-A
    by (intro-cong [σ2 (+), σ1 of-int] more:sum.UNION-disjoint[symmetric]) auto
  also have ... = real b∗(real c+2) + of-int(∑ a∈A. max 0(int(f a)−q-max−2^c))
    using h-range[OF ψ-range] by (intro-cong [σ2 (+), σ1 of-int] more:sum.cong) auto
  also have ... = ?R1
    unfolding ψ-def by simp
  finally show ?thesis
    by simp
 qed

 have 1: measure Ψ {ψ. real c ≤ V ψ} ≤ 2 powr (− (2^c)) (is ?L1 ≤ ?R1) for c
 proof −
   have ?L1 = measure Ψ {ψ. real b * (real c + 3) ≤ Z ψ}
    unfolding V-def using b-min by (intro measure-pmf-cong) (simp add:field-simps)
   also have ... ≤ measure Ψ
     {ψ. real b∗(real c+3)≤ real b∗(real c+2)+ of-int (∑ a∈A. max 0 (int (fst ψ a)−q-max
−2^c))}
     using 2 order-trans by (intro pmf-mono) blast
  also have ... = measure Ψ {ψ. real b ≤ (∑ a∈A. of-int (max 0 (int (fst ψ a) −q-max −2^c)))}
    by (intro measure-pmf-cong) (simp add:algebra-simps)
   also have ... ≤ (∫ ψ. (∑ a∈A. of-int (max 0 (int (fst ψ a) −q-max −2^c))) ∂Ψ)/real b
    using b-min by (intro pmf-markov sum-nonneg) simp-all
   also have ... = (∑ a∈A. (∫ ψ.  of-int (max 0 (int (fst ψ a) −q-max −2^c)) ∂Ψ))/real b
    by (intro-cong [σ2(/)] more:Bochner-Integration.integral-sum) simp
  also have ... = (∑ a∈A. (∫ f.  of-int (max 0 (int (f a)−q-max −2^c)) ∂(map-pmf fst Ψ)))/real
b
    by simp
   also have ... = (∑ a∈A. (∫ f. of-int (max 0 (int (f a) − (q-max +2^c))) ∂Ψ1))/real b
    unfolding sample-pro-Ψ map-fst-pair-pmf by (simp add:algebra-simps)
   also have ... ≤ (∑ a∈A. 2 powr −real (q-max + 2^c))/real b
```

```
    using b-min by (intro sum-mono divide-right-mono cutoff-eq-6) auto
  also have ... = real X * 2 powr (- real q-max + (- (2 ^ c))) / real b
    unfolding X-def by simp
  also have ... = (real X * 2 powr (-real q-max) / b) * 2 powr (-(2^c))
    unfolding powr-add by (simp add:algebra-simps)
  also have ... ≤ 1 * 2 powr (-(2^c))
    using cutoff-eq-7 by (intro mult-right-mono) auto
  finally show ?thesis
    by simp
qed

have 0: measure Ψ {ψ. x ≤ V ψ} ≤ exp (- x * ln x ^ 3)  (is ?L1 ≤ ?R1) if x ≥ 20 for x
proof -
  define c where c = nat ⌊x⌋

  have x * ln x^3 ≤ exp (x * ln 2) * ln 2/2 if x ≥ 150 for x::real
  proof -
    have aux-aux-0: x^4 ≥ 0
      by simp

    have x * ln x^3 ≤ x * x^3
      using that by (intro mult-left-mono power-mono ln-bound) auto
    also have ... = x^4 * 1
      by (simp add:numeral-eq-Suc)
    also have ... ≤ x^4 * ((ln 2 / 10)^4 * (150 * (ln 2 / 10))^6 * (ln 2/2))
      by (intro mult-left-mono aux-aux-0) (approximation 8)
    also have ... = (x * (ln 2 / 10))^4 * (150 * (ln 2 / 10))^6 * (ln 2/2)
      unfolding power-mult-distrib by (simp add:algebra-simps)
    also have ... ≤ (x * (ln 2 / 10))^4 * (x * (ln 2 / 10))^6 * (ln 2/2)
      by (intro mult-right-mono mult-left-mono power-mono that) auto
    also have ... = (0+x * (ln 2 / 10))^10 * (ln 2/2)
      unfolding power-add[symmetric] by simp
    also have ... ≤ (1+x * ln 2 / 10)^10 * (ln 2/2)
      using that by (intro mult-right-mono power-mono add-mono) auto
    also have ... ≤ exp (x * ln 2 / 10)^10 * (ln 2/2)
      using that by (intro mult-right-mono power-mono exp-ge-add-one-self) auto
    also have ... = exp (x * ln 2) * (ln 2/2)
      unfolding exp-of-nat-mult[symmetric] by simp
    finally show ?thesis by simp
  qed
  moreover have x * ln x^3 ≤ exp (x * ln 2) * ln 2/2 if x ∈ {20..150}
    using that by (approximation 10 splitting: x=1)

  ultimately have x * ln x^3 ≤ exp (x * ln 2) * ln 2/2
    using that by fastforce
  also have ... = 2 powr (x-1) * ln 2
    unfolding powr-diff unfolding powr-def by simp
  also have ... ≤ 2 powr c * ln 2
    unfolding c-def using that
    by (intro mult-right-mono powr-mono) auto
  also have ... = 2^c * ln 2
    using powr-realpow by simp
  finally have aux0: x * ln x^3 ≤ 2^c * ln 2
    by simp
  have real c ≤ x
    using that unfolding c-def by linarith
  hence ?L1 ≤ measure Ψ {ψ. real c ≤ V ψ}
    by (intro pmf-mono) auto
```

```
    also have ... ≤ 2 powr (−(2^c))
      by (intro 1)
    also have ... = exp (− (2 ^ c * ln 2))
      by (simp add:powr-def)
    also have ... ≤ exp (− (x *ln x^3))
      using aux0 by (intro iffD2[OF exp-le-cancel-iff]) auto
    also have ... = ?R1
      by simp
    finally show ?thesis
      by simp
  qed

  have ?L ≤ measure Ω {ω. is-too-large (τ2 ω A q-max)}
    using lt-s-too-large
    by (intro pmf-mono) (simp del:is-too-large.simps)
  also have ... = measure Ω
    {ω. (∑(i,j)∈{..<l}×{..<b}. ⌊log 2 (of-int (max (τ2 ω A q-max i j) (−1)) + 2)⌋) > C5 * b
*l}
    by simp
  also have ... = measure Ω {ω. real-of-int (∑(i,j)∈{..<l}×{..<b}.
    ⌊log 2 (of-int (max (τ2 ω A q-max i j) (−1)) + 2)⌋) > of-int (C5 * b * l)}
    unfolding of-int-less-iff by simp
  also have ... = measure Ω {ω. real-of-int C5 * real b * real l < of-int (∑x∈{..<l} × {..<b}.
    ⌊log 2 (real-of-int (τ1 (ω (fst x)) A q-max (snd x)) + 2)⌋)}
    by (intro-cong [σ2 measure, σ1 Collect, σ1 of-int, σ2 (<)] more:ext sum.cong)
     (auto simp add:case-prod-beta τ2-def τ1-def)

  also have ... = measure Ω {ω. (∑i<l. Z (ω i)) > of-int C5 * real b * real l}
    unfolding Z-def sum.cartesian-product τ1-def by (simp add:case-prod-beta)
  also have ... = measure Ω {ω. (∑i<l. V (ω i) + 3) > of-int C5 * real l}
    unfolding V-def using b-min
    by (intro measure-pmf-cong) (simp add:sum-divide-distrib[symmetric] field-simps sum.distrib)
  also have ... = measure Ω {ω. (∑i<l. V (ω i)) > of-int (C5−3) * real l}
    by (simp add:sum.distrib algebra-simps)
  also have ... ≤ measure Ω {ω. (∑i<l. V (ω i)) ≥ C1 * real l}
    unfolding C5-def using C1-est by (intro pmf-mono) auto
  also have ... ≤ exp (− real l)
    by (intro deviation-bound l-gt-0 0) (simp-all add: Λ-def)
  also have ... ≤ exp (− (C6 * ln (2 / δ)))
    using l-lbound by (intro iffD2[OF exp-le-cancel-iff]) auto
  also have ... ≤ exp (− (1 * ln (2 / δ)))
    unfolding C6-def using δ-gt-0 δ-lt-1
    by (intro iffD2[OF exp-le-cancel-iff] le-imp-neg-le mult-right-mono ln-ge-zero) auto
  also have ... = exp ( ln ( δ / 2))
    using δ-gt-0 by (simp add: ln-div)
  also have ... = δ/2
    using δ-gt-0 by simp
  finally show ?thesis
    by simp
qed

end

unbundle no intro-cong-syntax

end
```

9 Accuracy with cutoff

This section verifies that each of the l estimate have the required accuracy with high probability assuming as long as the cutoff is below *q-max*, generalizing the result from Section 7.

theory *Distributed-Distinct-Elements-Accuracy*
imports
Distributed-Distinct-Elements-Accuracy-Without-Cutoff
Distributed-Distinct-Elements-Cutoff-Level
begin

unbundle *intro-cong-syntax*

lemma (**in** *semilattice-set*) *Union*:
assumes *finite I* $I \neq \{\}$
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *finite* $(Z\ i)$
assumes $\bigwedge i.\ i \in I \Longrightarrow Z\ i \neq \{\}$
shows $F\ (\bigcup\ (Z\ `\ I)) = F\ ((\lambda i.\ (F\ (Z\ i)))\ `\ I)$
using *assms*(*1*,*2*,*3*,*4*)
proof (*induction I rule*:*finite-ne-induct*)
case (*singleton x*)
then show *?case* **by** *simp*
next
case (*insert x I*)
have $F\ (\bigcup\ (Z\ `\ insert\ x\ I)) = F\ ((Z\ x) \cup (\bigcup\ (Z\ `\ I)))$
by *simp*
also have $... = f\ (F\ (Z\ x))\ (F\ (\bigcup\ (Z\ `\ I)))$
using *insert* **by** (*intro union finite-UN-I*) *auto*
also have $... = f\ (F\ \{F\ (Z\ x)\})\ (F\ ((\lambda i.\ F\ (Z\ i))\ `\ I))$
using *insert*(*5*,*6*) **by** (*subst insert*(*4*)) *auto*
also have $... = F\ (\{F\ (Z\ x)\} \cup (\lambda i.\ F\ (Z\ i))\ `\ I)$
using *insert*(*1*,*2*) **by** (*intro union*[*symmetric*] *finite-imageI*) *auto*
also have $... = F\ ((\lambda i.\ F\ (Z\ i))\ `\ insert\ x\ I)$
by *simp*
finally show *?case* **by** *simp*
qed

This is similar to the existing *hom-Max-commute* with the crucial difference that it works even if the function is a homomorphism between distinct lattices. An example application is $Max\ (int\ `\ A) = int\ (Max\ A)$.

lemma *hom-Max-commute′*:
assumes *finite A* $A \neq \{\}$
assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow max\ (f\ x)\ (f\ y) = f\ (max\ x\ y)$
shows $Max\ (f\ `\ A) = f\ (Max\ A)$
using *assms* **by** (*induction A rule*:*finite-ne-induct*) *auto*

context *inner-algorithm-fix-A*
begin

definition t_c
where $t_c\ \psi\ \sigma = (Max\ ((\lambda j.\ \tau_1\ \psi\ A\ \sigma\ j + \sigma)\ `\ \{..<b\})) - b\text{-}exp + 9$

definition s_c
where $s_c\ \psi\ \sigma = nat\ (t_c\ \psi\ \sigma)$

definition p_c
where $p_c\ \psi\ \sigma = card\ \{j \in \{..<b\}.\ \tau_1\ \psi\ A\ \sigma\ j + \sigma \geq s_c\ \psi\ \sigma\}$

definition Y_c
where $Y_c\ \psi\ \sigma = 2\ \widehat{}\ s_c\ \psi\ \sigma * \varrho\text{-}inv\ (p_c\ \psi\ \sigma)$

lemma s_c*-eq-s*:
assumes $(f,g,h) \in$ *sample-pro* Ψ
assumes $\sigma \leq s\ f$
shows $s_c\ (f,g,h)\ \sigma = s\ f$
proof −
have *int* $(Max\ (f\ `\ A)) - int\ b\text{-}exp + 9 \leq int\ (Max\ (f\ `\ A)) - 26 + 9$
using *b-exp-ge-26* **by** (*intro add-mono diff-left-mono*) *auto*
also have ... $\leq int\ (Max\ (f\ `\ A))$ **by** *simp*
finally have *1*:*int* $(Max\ (f\ `\ A)) - int\ b\text{-}exp + 9 \leq int\ (Max\ (f\ `\ A))$
by *simp*
have $\sigma \leq int\ (s\ f)$ **using** *assms*(*2*) **by** *simp*
also have ... $= max\ 0\ (t\ f)$
unfolding *s-def* **by** *simp*
also have ... $\leq max\ 0\ (int\ (Max\ (f\ `\ A)))$
unfolding *t-def* **using** *1* **by** *simp*
also have ... $= int\ (Max\ (f\ `\ A))$
by *simp*
finally have $\sigma \leq int\ (Max\ (f\ `\ A))$
by *simp*
hence *0*: $int\ \sigma - 1 \leq int\ (Max\ (f\ `\ A))$
by *simp*

have *c*:$h \in$ *sample-pro* $(\mathcal{H}\ k\ (C_7 * b^2)\ (\mathcal{N}\ b))$
using *assms*(*1*) *sample-set-*Ψ **by** *auto*
hence *h-range*: $h\ x < b$ **for** x
using *h-range-1* **by** *simp*

have $(MAX\ j{\in}\{..{<}b\}.\ \tau_1\ (f,\ g,\ h)\ A\ \sigma\ j + int\ \sigma) =$
$(MAX\ x{\in}\{..{<}b\}.\ Max\ (\{int\ (f\ a)\ |a.\ a \in A \wedge h\ (g\ a) = x\} \cup \{-1\} \cup \{int\ \sigma\ -1\}))$
using *fin-f*[*OF assms*(*1*)] **by** (*simp add*:*max-add-distrib-left max.commute* τ_1*-def*)
also have ... $= Max\ (\bigcup x{<}b.\ \{int\ (f\ a)\ |a.\ a \in A \wedge h\ (g\ a) = x\} \cup \{-\ 1\} \cup \{int\ \sigma\ -\ 1\})$
using *fin-f*[*OF assms*(*1*)] *b-ne* **by** (*intro Max.Union*[*symmetric*]) *auto*
also have ... $= Max\ (\{int\ (f\ a)\ |a.\ a \in A\} \cup \{-\ 1,\ int\ \sigma\ -\ 1\})$
using *h-range* **by** (*intro arg-cong*[**where** *f*=*Max*]) *auto*
also have ... $= max\ (Max\ (int\ `\ f\ `\ A))\ (int\ \sigma\ -\ 1)$
using *A-nonempty fin-A* **unfolding** *Setcompr-eq-image image-image*
by (*subst Max.union*) *auto*
also have ... $= max\ (int\ (Max\ (f\ `\ A)))\ (int\ \sigma\ -\ 1)$
using *fin-A A-nonempty* **by** (*subst hom-Max-commute*′) *auto*
also have ... $= int\ (Max\ (f\ `\ A))$
by (*intro max-absorb1 0*)
finally have $(MAX\ j{\in}\{..{<}b\}.\ \tau_1\ (f,\ g,\ h)\ A\ \sigma\ j + int\ \sigma) = Max\ (f\ `\ A)$ **by** *simp*

thus *?thesis*
unfolding s_c*-def* t_c*-def s-def t-def* **by** *simp*
qed

lemma p_c*-eq-p*:
assumes $(f,g,h) \in$ *sample-pro* Ψ
assumes $\sigma \leq s\ f$
shows $p_c\ (f,g,h)\ \sigma = p\ (f,g,h)$
proof −
have $\{j \in \{..{<}b\}.\ int\ (s\ f) \leq max\ (\tau_0\ (f,\ g,\ h)\ A\ j)\ (int\ \sigma\ -\ 1)\} =$
$\{j \in \{..{<}b\}.\ int\ (s\ f) \leq max\ (\tau_0\ (f,\ g,\ h)\ A\ j)\ (-\ 1)\}$

```
    using assms(2) unfolding le-max-iff-disj by simp
  thus ?thesis
    unfolding p_c-def p-def s_c-eq-s[OF assms]
    by (simp add:max-add-distrib-left τ_1-def del:τ_0.simps)
qed

lemma Y_c-eq-Y:
  assumes (f,g,h) ∈ sample-pro Ψ
  assumes σ ≤ s f
  shows Y_c (f,g,h) σ = Y (f,g,h)
  unfolding Y_c-def Y-def s_c-eq-s[OF assms] p_c-eq-p[OF assms] by simp

lemma accuracy-single: measure Ψ {ψ. ∃σ ≤ q-max. |Y_c ψ σ − real X| > ε * X} ≤ 1/2^4
  (is ?L ≤ ?R)
proof −
  have measure Ψ {ψ. ∃σ ≤ q-max. |Y_c ψ σ − real X| > ε * real X} ≤
    measure Ψ {(f,g,h). |Y (f,g,h) − real X| >  ε * real X ∨ s f < q-max}
  proof (rule pmf-mono)
    fix ψ
    assume a:ψ ∈ {ψ. ∃σ≤q-max. ε * real X < |Y_c ψ σ − real X|}
    assume d:ψ ∈ set-pmf (sample-pro Ψ)
    obtain σ where b:σ ≤ q-max and c: ε * real X < |Y_c ψ σ − real X|
      using a by auto
    obtain f g h where ψ-def: ψ = (f,g,h) by (metis prod-cases3)
    hence e:(f,g,h) ∈ sample-pro Ψ using d by simp

    show ψ ∈ {(f, g, h). ε * real X < |Y (f, g, h) − real X| ∨ s f < q-max}
    proof (cases s f ≥ q-max)
      case True
      hence f:σ ≤ s f using b by simp
      have ε * real X < |Y ψ − real X|
        using Y_c-eq-Y[OF e f] c unfolding ψ-def by simp
      then show ?thesis unfolding ψ-def by simp
    next
      case False
      then show ?thesis unfolding ψ-def by simp
    qed
  qed
  also have ... ≤ 1/2^4
    using accuracy-without-cutoff by simp
  finally show ?thesis by simp
qed

lemma estimate1-eq:
  assumes j < l
  shows estimate1 (τ_2 ω A σ, σ) j = Y_c (ω j) σ (is ?L = ?R)
proof −
  define t where t = max 0 (Max ((τ_2 ω A σ j) ` {..<b}) + σ − ⌊log 2 b⌋ + 9)
  define p where p = card { k. k ∈ {..<b} ∧ (τ_2 ω A σ j k) + σ ≥ t }

  have 0: int (nat x) = max 0 x for x
    by simp
  have 1: ⌊log 2 b⌋ = b-exp
    unfolding b-def by simp

  have b > 0
    using b-min by simp
  hence 2:  {..<b} ≠ {} by auto
```

```
  have t = int (nat (Max ((τ2 ω A σ j) ‘ {..<b}) + σ − b-exp + 9))
    unfolding t-def 0 1 by (rule refl)
  also have ... = int (nat (Max ((λx. x + σ) ‘ (τ2 ω A σ j) ‘ {..<b}) − b-exp + 9))
    by (intro-cong [σ1 int,σ1 nat,σ2(+),σ2(−)] more:hom-Max-commute) (simp-all add:2)
  also have ... = int (sc (ω j) σ)
    using assms
    unfolding sc-def tc-def τ2-def image-image by simp
  finally have 3:t = int (sc (ω j) σ)
    by simp

  have 4: p = pc (ω j) σ
    using assms unfolding p-def pc-def 3 τ2-def by simp

  have ?L = 2 powr t ∗ ln (1−p/b) / ln(1−1/b)
    unfolding estimate1.simps τ-def τ3-def
    by (simp only:t-def p-def Let-def)
  also have ... = 2 powr (sc (ω j) σ) ∗ ϱ-inv p
    unfolding 3 ϱ-inv-def by (simp)
  also have ... = ?R
    unfolding Yc-def 3 4 by (simp add:powr-realpow)
  finally show ?thesis
    by blast
qed

lemma estimate-result-1:
  measure Ω {ω. (∃σ≤q-max. ε∗X < |estimate (τ2 ω A σ,σ)−X|) } ≤ δ/2 (is ?L ≤ ?R)
proof −
  define I :: real set where I = {x. |x − real X| ≤ ε∗X}

  define μ where μ = measure Ψ {ψ. ∃σ≤q-max. Yc ψ σ∉I}

  have int-I: interval I
    unfolding interval-def I-def by auto

  have μ = measure Ψ {ψ. ∃σ ≤ q-max. |Yc ψ σ − real X| > ε ∗ X}
    unfolding μ-def I-def by (simp add:not-le)
  also have ... ≤  1 / 2 ^ 4
    by (intro accuracy-single)
  also have ... = 1/ 16
    by simp
  finally have 1:μ ≤ 1 / 16 by simp

  have (μ + Λ) ≤ 1/16 + 1/16
    unfolding Λ-def by (intro add-mono 1) auto
  also have ... ≤ 1/8
    by simp
  finally have 2:(μ + Λ) ≤ 1/8
    by simp

  hence 0: (μ + Λ) ≤ 1/2
    by simp

  have μ ≥ 0
    unfolding μ-def by simp
  hence 3: μ + Λ > 0
    by (intro add-nonneg-pos Λ-gt-0)
```

have *?L = measure Ω {ω. (∃σ≤q-max. ε∗X < |median l (estimate1 (τ₂ ω A σ,σ))−X|) }*
by *simp*
also have *... = measure Ω {ω. (∃σ≤q-max. median l (estimate1 (τ₂ ω A σ,σ)) ∉ I)}*
unfolding *I-def* **by** (*intro measure-pmf-cong*) *auto*
also have *... ≤ measure Ω {ω. real(card{i∈{..<l}.(∃σ≤q-max. Y_c (ω i) σ∉I)})≥ real l/2}*
proof (*rule pmf-mono*)
fix *ω*
assume *ω ∈ set-pmf Ω ω ∈ {ω. ∃σ≤q-max. median l (estimate1 (τ₂ ω A σ, σ)) ∉ I}*
then obtain *σ* **where** *σ-def*: *median l (estimate1 (τ₂ ω A σ, σ)) ∉ I σ≤q-max*
by *auto*
hence *real l ≤ real (2 ∗ card {i. i <l ∧ estimate1 (τ₂ ω A σ, σ) i ∉ I})*
by (*intro of-nat-mono median-est-rev*[*OF int-I*])
also have *... = 2 ∗ real (card {i∈{..<l}. estimate1 (τ₂ ω A σ, σ) i ∉ I})*
by *simp*
also have *... = 2 ∗ real (card {i ∈ {..<l}. Y_c (ω i) σ ∉ I})*
using *estimate1-eq* **by** (*intro-cong* [*σ₂* (∗), *σ₁ of-nat*, *σ₁ card*] *more:restr-Collect-cong*) *auto*
also have *... ≤ 2 ∗ real (card {i ∈ {..<l}. (∃σ≤q-max. Y_c (ω i) σ ∉ I)})*
using *σ-def*(*2*) **by** (*intro mult-left-mono Nat.of-nat-mono card-mono*) *auto*
finally have *real l ≤ 2 ∗ real (card {i ∈ {..<l}. (∃σ≤q-max. Y_c (ω i) σ ∉ I)})*
by *simp*
thus *ω ∈ {ω. real l/2 ≤ real (card {i ∈ {..<l}. ∃σ≤q-max. Y_c (ω i) σ ∉ I})}*
by *simp*
qed
also have *... = measure Ω {ω. real (card{i∈{..<l}. (∃σ≤q-max. Y_c (ω i) σ∉I)}) ≥ (1/2)∗real l}*
unfolding *p-def* **by** *simp*
also have *... ≤ exp (− real l ∗ ((1/2) ∗ ln (1 / (μ + Λ)) − 2 ∗ exp (− 1)))*
using *0* **unfolding** *μ-def* **by** (*intro walk-tail-bound l-gt-0 Λ-gt-0*) *auto*
also have *... = exp (− (real l ∗ ((1/2) ∗ ln (1 / (μ + Λ)) − 2 ∗ exp (− 1))))*
by *simp*
also have *... ≤ exp (− (real l ∗ ((1/2) ∗ ln 8 − 2 ∗ exp (− 1))))*
using *2 3 l-gt-0* **by** (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le mult-left-mono diff-mono*) (*auto simp add:divide-simps*)
also have *... ≤ exp (− (real l ∗ (1/4)))*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le mult-left-mono of-nat-0-le-iff*) (*approximation 5*)
also have *... ≤ exp (− (C₆ ∗ ln (2/ δ)∗(1/4)))*
by (*intro iffD2*[*OF exp-le-cancel-iff*] *le-imp-neg-le mult-right-mono l-lbound*) *auto*
also have *... = exp (− ln (2/ δ))*
unfolding *C₆-def* **by** *simp*
also have *... = ?R*
using *δ-gt-0* **by** (*subst ln-inverse*[*symmetric*]) *auto*
finally show *?thesis*
by *simp*
qed

theorem *estimate-result*:
measure Ω {ω. |estimate (τ ω A)− X| > ε ∗ X} ≤ δ
(**is** *?L ≤ ?R*)
proof −
let *?P = measure Ω*
have *?L ≤ ?P {ω. (∃σ≤q-max. ε∗real X<|estimate (τ₂ ω A σ, σ)−real X|)∨q ω A> q-max}*
unfolding *τ-def τ₃-def not-le*[*symmetric*]
by (*intro pmf-mono*) *auto*
also have *...≤ ?P {ω. (∃σ≤q-max. ε∗real X<|estimate (τ₂ ω A σ,σ)−X|)} + ?P {ω. q ω A> q-max}*
by (*intro pmf-add*) *auto*
also have *...≤ δ/2 + δ/2*

by (*intro add-mono cutoff-level estimate-result-1*)
also have ... = δ
by *simp*
finally show *?thesis*
by *simp*
qed

end

lemma (**in** *inner-algorithm*) *estimate-result*:
assumes $A \subseteq \{..<n\}$ $A \neq \{\}$
shows *measure* Ω $\{\omega.$ $|estimate\ (\tau\ \omega\ A) -\ real\ (card\ A)| > \varepsilon * real\ (card\ A)\} \leq \delta$ (**is** *?L* $\leq$ *?R*)
proof −
interpret *inner-algorithm-fix-A*
using *assms* **by** *unfold-locales auto*
have *?L* = *measure* Ω $\{\omega.$ $|estimate\ (\tau\ \omega\ A) -\ X| > \varepsilon * X\}$
unfolding *X-def* **by** *simp*
also have ... $\leq$ *?R*
by (*intro estimate-result*)
finally show *?thesis*
by *simp*
qed

unbundle *no intro-cong-syntax*

end

10 Outer Algorithm

This section introduces the final solution with optimal size space usage. Internally it relies on the inner algorithm described in Section 6, dependending on the paramaters n, ε and δ it either uses the inner algorithm directly or if ε^{-1} is larger than $\ln n$ it runs $\frac{\varepsilon^{-1}}{\ln \ln n}$ copies of the inner algorithm (with the modified failure probability $\frac{1}{\ln n}$) using an expander to select its seeds. The theorems below verify that the probability that the relative accuracy of the median of the copies is too large is below ε.

theory *Distributed-Distinct-Elements-Outer-Algorithm*
imports
Distributed-Distinct-Elements-Accuracy
Prefix-Free-Code-Combinators.Prefix-Free-Code-Combinators
Frequency-Moments.Landau-Ext
Landau-Symbols.Landau-More
begin

unbundle *intro-cong-syntax*

The following are non-asymptotic hard bounds on the space usage for the sketches and seeds repsectively. The end of this section contains a proof that the sum is asymptotically in $\mathcal{O}(\ln(\varepsilon^{-1})\delta^{-1} + \ln n)$.

definition *state-space-usage* = (λ(*n*,ε,δ). *2^40 * (ln(1/δ)+1)/ ε^2 + log 2 (log 2 n + 3)*)
definition *seed-space-usage* = (λ(*n*,ε,δ). *2^30+2^23*ln n+48*(log 2(1/ε)+16)^2+336*ln (1/δ)*)

locale *outer-algorithm* =
fixes *n* :: *nat*
fixes δ :: *real*
fixes ε :: *real*
assumes *n-gt-0*: *n > 0*

```
  assumes δ-gt-0: δ > 0 and δ-lt-1: δ < 1
  assumes ε-gt-0: ε > 0 and ε-lt-1: ε < 1
begin

definition n₀ where n₀ = max (real n) (exp (exp 5))
definition stage-two where stage-two = (δ < (1/ln n₀))
definition δᵢ :: real where δᵢ = (if stage-two then (1/ln n₀) else δ)
definition m :: nat where m = (if stage-two then nat ⌈4 * ln (1/ δ)/ln (ln n₀)⌉ else 1)
definition α where α = (if stage-two then (1/ln n₀) else 1)

lemma m-lbound:
  assumes stage-two
  shows m ≥ 4 * ln (1/ δ)/ln(ln n₀)
proof −
  have m = real (nat ⌈4 * ln (1 / δ) / ln (ln n₀)⌉)
    using assms unfolding m-def by simp
  also have ... ≥ 4 * ln (1 / δ) / ln (ln n₀)
    by linarith
  finally show ?thesis by simp
qed

lemma n-lbound:
  n₀ ≥ exp (exp 5) ln n₀ ≥ exp 5 5 ≤ ln (ln n₀) ln n₀ > 1 n₀ > 1
proof −
  show 0:n₀ ≥ exp (exp 5)
    unfolding n₀-def by simp
  have (1::real) ≤ exp (exp 5)
    by (approximation 5)
  hence n₀ ≥ 1
    using 0 by argo
  thus 1:ln n₀ ≥ exp 5
    using 0 by (intro iffD2[OF ln-ge-iff]) auto
  moreover have 1 < exp (5::real)
    by (approximation 5)
  ultimately show 2:ln n₀ > 1
    by argo
  show 5 ≤ ln (ln n₀)
    using 1 2 by (subst ln-ge-iff) simp
  have (1::real) < exp (exp 5)
    by (approximation 5)
  thus n₀ > 1
    using 0 by argo
qed

lemma δ1-gt-0: 0 < δᵢ
  using n-lbound(4) δ-gt-0 unfolding δᵢ-def
  by (cases stage-two) simp-all

lemma δ1-lt-1: δᵢ < 1
  using n-lbound(4) δ-lt-1 unfolding δᵢ-def
  by (cases stage-two) simp-all

lemma m-gt-0-aux:
  assumes stage-two
  shows 1 ≤ ln (1 / δ) / ln (ln n₀)
proof −
  have ln n₀ ≤ 1 / δ
    using n-lbound(4)
```

```
    using assms unfolding pos-le-divide-eq[OF δ-gt-0] stage-two-def
    by (simp add:divide-simps ac-simps)
  hence ln (ln n₀) ≤ ln (1 / δ)
    using n-lbound(4) δ-gt-0 by (intro iffD2[OF ln-le-cancel-iff] divide-pos-pos) auto
  thus 1 ≤ ln (1 / δ) / ln (ln n₀)
    using n-lbound(3)
    by (subst pos-le-divide-eq) auto
qed

lemma m-gt-0: m > 0
proof (cases stage-two)
  case True
  have 0 < 4 * ln (1/ δ)/ln(ln n₀)
    using m-gt-0-aux[OF True] by simp
  also have ... ≤ m
    using m-lbound[OF True] by simp
  finally have 0 < real m
    by simp
  then show ?thesis by simp
next
  case False
  then show ?thesis unfolding m-def by simp
qed

lemma α-gt-0: α > 0
  using n-lbound(4) unfolding α-def
  by (cases stage-two) auto

lemma α-le-1: α ≤ 1
  using n-lbound(4) unfolding α-def
  by (cases stage-two) simp-all

sublocale I: inner-algorithm n δ_i ε
  unfolding inner-algorithm-def using n-gt-0 ε-gt-0 ε-lt-1 δ1-gt-0 δ1-lt-1 by auto

abbreviation Θ where Θ ≡ E m α I.Ω

lemma Θ: m > 0 α > 0 using α-gt-0 m-gt-0 by auto

type-synonym state = inner-algorithm.state list

fun single :: nat ⇒ nat ⇒ state where
  single ϑ x = map (λj. I.single (pro-select Θ ϑ j) x) [0..<m]

fun merge :: state ⇒ state ⇒ state where
  merge x y = map (λ(x,y). I.merge x y) (zip x y)

fun estimate :: state ⇒ real where
  estimate x = median m (λi. I.estimate (x ! i))

definition ν :: nat ⇒ nat set ⇒ state
  where ν ϑ A = map (λi. I.τ (pro-select Θ ϑ i) A) [0..<m]
```

The following three theorems verify the correctness of the algorithm. The term τ is a mathematical description of the sketch for a given subset, while *local.single*, *local.merge* are the actual functions that compute the sketches.

```
theorem merge-result: merge (ν ω A) (ν ω B) = ν ω (A ∪ B) (is ?L = ?R)
proof −
```

```
  have 0: zip [0..<m] [0..<m] = map (λx. (x,x)) [0..<m] for m
    by (induction m, auto)

  have ?L = map (λx. I.merge (I.τ (pro-select Θ ω x) A) (I.τ (pro-select Θ ω x) B)) [0..<m]
    unfolding ν-def
    by (simp add:zip-map-map 0 comp-def case-prod-beta)
  also have ... = map (λx.  I.τ (pro-select Θ ω x) (A ∪ B)) [0..<m]
    by (intro map-cong refl I.merge-result expander-pro-range[OF Θ])
  also have ... = ?R
    unfolding ν-def by simp
  finally show ?thesis by simp
qed

theorem single-result: single ω x = ν ω {x} (is ?L = ?R)
proof −
  have ?L = map (λj. I.single (pro-select Θ ω j) x) [0..<m]
    by (simp del:I.single.simps)
  also have ... = ?R
    unfolding ν-def by (intro map-cong I.single-result expander-pro-range[OF Θ]) auto
  finally show ?thesis by simp
qed

theorem estimate-result:
  assumes A ⊆ {..<n} A ≠ {}
  defines p ≡ (pmf-of-set {..<pro-size Θ})
  shows measure p {ω. |estimate (ν ω A)− real (card A)| > ε ∗ real (card A)} ≤ δ (is ?L ≤ ?R)
proof (cases stage-two)
  case True
  define I where I = {x. |x − real (card A)| ≤ ε ∗ real (card A)}
  have int-I: interval I
    unfolding interval-def I-def by auto

  define μ where μ = measure I.Ω {ω. I.estimate (I.τ ω A) ∉ I}

  have 0:μ + α > 0
    unfolding μ-def
    by (intro add-nonneg-pos α-gt-0) auto

  have μ ≤ δ_i
    unfolding μ-def I-def using I.estimate-result[OF assms(1,2)]
    by (simp add: not-le del:I.estimate.simps)
  also have ... = 1/ln n_0
    using True unfolding δ_i-def by simp
  finally have μ ≤ 1/ln n_0 by simp
  hence μ + α ≤ 1/ln n_0 + 1/ln n_0
    unfolding α-def using True by (intro add-mono) auto
  also have ... = 2/ln n_0
    by simp
  finally have 1:μ + α ≤ 2 / ln n_0
    by simp
  hence 2:ln n_0 ≤ 2 / (μ + α)
    using 0 n-lbound by (simp add:field-simps)

  have μ + α ≤ 2/ln n_0
    by (intro 1)
  also have ... ≤ 2/exp 5
    using n-lbound by (intro divide-left-mono) simp-all
  also have ... ≤ 1/2
```

```
    by (approximation 5)
  finally have 3:μ + α ≤ 1/2 by simp

  have 4: 2 * ln 2 + 8 * exp (− 1) ≤ (5::real)
    by (approximation 5)

  have ?L = measure p {ω. median m (λi. I.estimate (ν ω A  ! i)) ∉ I}
    unfolding I-def by (simp add:not-le)
  also have ... ≤
    measure p {ϑ. real (card {i ∈ {..<m}. I.estimate (I.τ (pro-select Θ ϑ i) A) ∉ I})≥ real m/2}
  proof (rule pmf-mono)
    fix ϑ assume ϑ ∈ set-pmf p
    assume a:ϑ ∈ {ω. median m (λi. I.estimate (ν ω A ! i)) ∉ I}
    hence real m ≤ real (2∗card {i. i < m ∧ I.estimate (ν ϑ A ! i) ∉ I})
      by (intro of-nat-mono median-est-rev int-I) auto
    also have ... = 2 * real (card {i∈{..<m}. I.estimate (ν ϑ A ! i) ∉ I})
      by simp
    also have ... = 2 * real (card {i ∈ {..<m}. I.estimate (I.τ (pro-select Θ ϑ i) A) ∉ I})
      unfolding ν-def by (intro-cong [σ2 (∗), σ1 of-nat, σ1 card] more:restr-Collect-cong)
       (simp del:I.estimate.simps)
    finally have real m ≤ 2 * real (card {i ∈ {..<m}. I.estimate (I.τ (pro-select Θ ϑ i) A) ∉ I})
      by simp
    thus ϑ ∈ {ϑ. real m / 2 ≤ real (card {i ∈ {..<m}. I.estimate (I.τ (pro-select Θ ϑ i) A) ∉
I})}
      by simp
  qed
  also have ...=measure Θ{ϑ. real(card {i ∈ {..<m}. I.estimate (I.τ (ϑ i) A) ∉ I})≥(1/2)∗real
m}
    unfolding sample-pro-alt p-def by (simp del:I.estimate.simps)
  also have ... ≤ exp (−real m ∗ ((1/2) ∗ ln (1/ (μ + α)) − 2∗exp (−1)))
    using 3 m-gt-0 α-gt-0 unfolding μ-def by (intro walk-tail-bound) force+
  also have ... ≤ exp (−real m ∗ ((1/2) ∗ ln (ln n0 / 2) − 2∗exp (−1)))
    using 0 2 3 n-lbound
    by (intro iffD2[OF exp-le-cancel-iff] mult-right-mono mult-left-mono-neg[where c=−real m]
        diff-mono mult-left-mono iffD2[OF ln-le-cancel-iff]) (simp-all)
  also have ... = exp (−real m ∗ (ln (ln n0) / 2 − (ln 2/2 + 2∗exp (−1))))
    using n-lbound by (subst ln-div) (simp-all add:algebra-simps)
  also have ... ≤ exp (−real m ∗ (ln (ln n0) / 2 − (ln (ln (exp(exp 5))) / 4)))
    using 4
   by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono-neg[where c=−real m] diff-mono) simp-all
  also have ... ≤ exp (−real m ∗ (ln (ln n0) / 2 − (ln (ln n0) / 4)))
    using n-lbound
   by (intro iffD2[OF exp-le-cancel-iff] mult-left-mono-neg[where c=−real m] diff-mono) simp-all
  also have ... = exp (− real m ∗ (ln (ln n0)/ 4) )
    by (simp add:algebra-simps)
  also have ... ≤ exp (− (4 ∗ ln (1/ δ)/ln(ln n0)) ∗ (ln (ln n0)/4))
    using m-lbound[OF True] n-lbound
    by (intro iffD2[OF exp-le-cancel-iff] mult-right-mono divide-nonneg-pos) simp-all
  also have ... = exp (− ln (1/ δ))
    using n-lbound by simp
  also have ... = δ
    using δ-gt-0 by (subst ln-inverse[symmetric]) auto
  finally show ?thesis by simp
next
  case False
  have m-eq: m = 1
    unfolding m-def using False by simp
  hence ?L = measure p {ω. ε ∗ real (card A) < |I.estimate (ν ω A ! 0) − real (card A)|}
```

```
    unfolding estimate.simps m-eq median-def by simp
  also have ... =  measure p {ω. ε*card A<|I.estimate (I.τ (pro-select Θ ω 0) A)−real(card A)|}
    unfolding ν-def m-eq by (simp del: I.estimate.simps)
  also have ... = measure Θ {ω. ε*real(card A) < |I.estimate (I.τ (ω 0) A)−real(card A)|}
    unfolding sample-pro-alt p-def by (simp del:I.estimate.simps)
  also have ...=
    measure (map-pmf (λϑ. ϑ 0) Θ) {ω. ε*real(card A) < |I.estimate (I.τ ω A)−real(card A)|}
    by simp
  also have ... = measure I.Ω {ω. ε*real(card A) < |I.estimate (I.τ ω A)−real(card A)|}
    using m-eq by (subst expander-uniform-property[OF Θ]) auto
  also have ... ≤ δ_i
    by (intro I.estimate-result[OF assms(1,2)])
  also have ... = ?R
    unfolding δ_i-def using False by simp
  finally show ?thesis
    by simp
qed
```

The function *encode-state* can represent states as bit strings. This enables verification of the space usage.

```
definition encode-state
  where encode-state = Lf_e I.encode-state m

lemma encode-state: is-encoding encode-state
  unfolding encode-state-def
  by (intro fixed-list-encoding I.encode-state)

lemma state-bit-count:
  bit-count (encode-state (ν ω A)) ≤ state-space-usage (real n, ε, δ)
    (is ?L ≤ ?R)
proof −
  have 0: length (ν ω A) = m
    unfolding ν-def by simp
  have ?L = (∑ x←ν ω A. bit-count (I.encode-state x))
    using 0 unfolding encode-state-def fixed-list-bit-count by simp
  also have ... = (∑ x←[0..<m]. bit-count (I.encode-state (I.τ (pro-select Θ ω x) A)))
    unfolding ν-def by (simp add:comp-def)
  also have ... ≤ (∑ x←[0..<m]. ereal (2^36 *(ln (1/δ_i)+ 1)/ε² + log 2 (log 2 (real n) + 3)))
    using I.state-bit-count by (intro sum-list-mono I.state-bit-count expander-pro-range[OF Θ])
  also have ... = ereal ( real m * (2^36 *(ln (1/δ_i)+ 1)/ε² + log 2 (log 2 (real n) + 3)))
    unfolding sum-list-triv-ereal by simp
  also have ... ≤ 2^40 * (ln(1/δ)+1)/ ε^2 + log 2 (log 2 n + 3) (is ?L1 ≤ ?R1)
  proof (cases stage-two)
    case True
    have ⌈4*ln (1/δ)/ln(ln n_0)⌉ ≤ 4*ln (1/δ)/ln(ln n_0) + 1
      by simp
    also have ... ≤ 4*ln (1/δ)/ln(ln n_0) + ln (1/δ)/ln(ln n_0)
      using m-gt-0-aux[OF True] by (intro add-mono) auto
    also have ... = 5 * ln (1/δ)/ln(ln n_0) by simp
    finally have 3: ⌈4*ln (1/δ)/ln(ln n_0)⌉ ≤ 5 * ln (1/δ)/ln(ln n_0)
      by simp

    have 4: 0 ≤ log 2 (log 2 (real n) + 3)
      using n-gt-0
      by (intro iffD2[OF zero-le-log-cancel-iff] add-nonneg-pos) auto

    have 5: 1 / ln 2 + 3 / exp 5 ≤ exp (1::real)  1.2 / ln 2 ≤ (2::real)
      by (approximation 5)+
```

```
  have log 2(log 2 (real n)+3) ≤ log 2 (log 2 n₀ + 3)
    using n-gt-0 by (intro iffD2[OF log-le-cancel-iff] add-mono add-nonneg-pos
        iffD2[OF zero-le-log-cancel-iff]) (simp-all add:n₀-def)
  also have ... = ln (ln n₀ / ln 2 + 3) / ln 2
    unfolding log-def by simp
  also have ... ≤ ln (ln n₀/ln 2 + (3 / exp 5) * ln n₀) / ln 2
   using n-lbound by (intro divide-right-mono iffD2[OF ln-le-cancel-iff] add-mono add-nonneg-pos)
     (simp-all add:divide-simps)
  also have ... = ln ( ln n₀ * (1 /ln 2 + 3/exp 5)) / ln 2
    by (simp add:algebra-simps)
  also have ... ≤ ln ( ln n₀ * exp 1) / ln 2
    using n-lbound by (intro divide-right-mono iffD2[OF ln-le-cancel-iff] add-mono
        mult-left-mono 5 Rings.mult-pos-pos add-pos-nonneg) auto
  also have ... = (ln (ln n₀) + 1) / ln 2
    using n-lbound by (subst ln-mult) simp-all
  also have ... ≤ (ln (ln n₀) + 0.2 * ln (ln n₀)) / ln 2
    using n-lbound by (intro divide-right-mono add-mono) auto
  also have ... = (1.2/ ln 2) * ln (ln n₀)
    by simp
  also have ... ≤ 2 * ln (ln n₀)
    using n-lbound by (intro mult-right-mono 5) simp
  finally have log 2(log 2 (real n)+3) ≤ 2 * ln (ln n₀)
    by simp
  hence 6: log 2(log 2 (real n)+3)/ln(ln n₀) ≤ 2
    using n-lbound by (subst pos-divide-le-eq) simp-all

   have ?L1 = real(nat ⌈4*ln (1/δ)/ln(ln n₀)⌉)*(2^36*(ln (ln n₀)+1)/ε^2+log 2(log 2 (real
n)+3))
    using True unfolding m-def δᵢ-def by simp
  also have ... = ⌈4*ln (1/δ)/ln(ln n₀)⌉*(2^36*(ln (ln n₀)+1)/ε^2+log 2(log 2 (real n)+3))
    using m-gt-0-aux[OF True] by (subst of-nat-nat) simp-all
  also have ... ≤ (5*ln (1/δ)/ln(ln n₀)) *(2^36*(ln (ln n₀)+1)/ε^2+log 2(log 2 (real n)+3))
    using n-lbound(3) ε-gt-0  4 by (intro ereal-mono mult-right-mono
        add-nonneg-nonneg divide-nonneg-pos mult-nonneg-nonneg 3) simp-all
   also have ... ≤ (5 * ln (1/δ)/ln(ln n₀))*((2^36+2^36)*ln (ln n₀)/ε^2+log 2(log 2 (real
n)+3))
    using n-lbound δ-gt-0 δ-lt-1
    by (intro ereal-mono mult-left-mono add-mono divide-right-mono divide-nonneg-pos) auto
  also have ... = 5*(2^37)* ln (1/δ)/ ε^2 + (5*ln (1/δ)) * (log 2(log 2 (real n)+3)/ln(ln n₀))
    using n-lbound by (simp add:algebra-simps)
  also have ... ≤ 5*(2^37)* ln (1/δ)/ ε^2 + (5*ln(1/ δ)) * 2
    using δ-gt-0 δ-lt-1 by (intro add-mono ereal-mono order.refl mult-left-mono 6) auto
  also have ... = 5*(2^37)* ln (1/δ)/ ε^2 + 5*2*ln(1/ δ) / 1
    by simp
  also have ... ≤ 5*(2^37)* ln (1/δ)/ ε^2 + 5*2*ln(1/ δ) / ε^2
    using ε-gt-0 ε-lt-1 δ-gt-0 δ-lt-1
    by (intro add-mono ereal-mono divide-left-mono Rings.mult-pos-pos power-le-one) auto
  also have ... = (5*(2^37+2))* (ln (1/δ)+0)/ ε^2 + 0
    by (simp add:algebra-simps)
  also have ... ≤ 2^40 * (ln (1 / δ)+1) / ε^2 +  log 2 (log 2 (real n) + 3)
    using ε-gt-0 ε-lt-1 δ-gt-0 δ-lt-1 n-gt-0 by (intro add-mono ereal-mono divide-right-mono
        mult-right-mono iffD2[OF zero-le-log-cancel-iff] add-nonneg-pos) auto
  finally show ?thesis by simp
 next
  case False
  have ?L1 = 2^36 *(ln (1/δ)+ 1)/ε² + log 2 (log 2 (real n) + 3)
    using False unfolding δᵢ-def m-def by simp
```

```
    also have ... ≤ ?R1
      using ε-gt-0 ε-lt-1 δ-gt-0 δ-lt-1
      by (intro ereal-mono add-mono divide-right-mono mult-right-mono add-nonneg-nonneg) auto
    finally show ?thesis by simp
  qed
  finally show ?thesis
    unfolding state-space-usage-def by simp
qed
```

Encoding function for the seeds which are just natural numbers smaller than *pro-size* Θ.

```
definition encode-seed
  where encode-seed = Nb_e (pro-size Θ)

lemma encode-seed:
  is-encoding encode-seed
  unfolding encode-seed-def by (intro bounded-nat-encoding)

lemma random-bit-count:
  assumes ω < pro-size Θ
  shows bit-count (encode-seed ω) ≤ seed-space-usage (real n, ε, δ)
    (is ?L ≤ ?R)
proof −
  have 0: pro-size Θ > 0 by (intro pro-size-gt-0)
  have 1: pro-size I.Ω > 0 by (intro pro-size-gt-0)

  have (55+60∗ln (ln n_0))^3 ≤ (180+60∗ln (ln n_0))^3
    using n-lbound by (intro power-mono add-mono) auto
  also have ... = 180^3 ∗ (1+ln (ln n_0)/real 3)^3
    unfolding power-mult-distrib[symmetric] by simp
  also have ... ≤ 180^3 ∗ exp (ln (ln n_0))
    using n-lbound by (intro mult-left-mono exp-ge-one-plus-x-over-n-power-n) auto
  also have ... = 180^3 ∗ ln n_0
    using n-lbound by (subst exp-ln) auto
  also have ... ≤ 180^3 ∗ max (ln n) (ln (exp (exp 5)))
    using n-gt-0 unfolding n_0-def by (subst ln-max-swap) auto
  also have ... ≤ 180^3 ∗ (ln n + exp 5)
    using n-gt-0 unfolding ln-exp by (intro mult-left-mono) auto
  finally have 2:(55+60∗ln (ln n_0))^3 ≤ 180^3 ∗ ln n + 180^3∗exp 5
    by simp

  have 3:(1::real)+180^3∗exp 5 ≤ 2^30 (4::real)/ln 2 + 180^3 ≤ 2^23
    by (approximation 10)+

  have ?L = ereal (real (floorlog 2 (pro-size Θ − 1)))
    using assms unfolding encode-seed-def bounded-nat-bit-count by simp
  also have ... ≤ ereal (real (floorlog 2 (pro-size Θ)))
    by (intro ereal-mono Nat.of-nat-mono floorlog-mono) auto
  also have ... = ereal (1 + of-int ⌊log 2 (real (pro-size Θ))⌋)
    using 0 unfolding floorlog-def by simp
  also have ... ≤ ereal (1 + log 2 (real (pro-size Θ)))
    by (intro add-mono ereal-mono) auto
  also have ... = 1 + log 2 (real (pro-size I.Ω)) ∗ (2^4) ^ ((m − 1) ∗ nat ⌈ln α / ln 0.95⌉))
    unfolding expander-pro-size[OF Θ] by simp
  also have ... = 1 + log 2 (real (pro-size I.Ω)) ∗ 2^ (4 ∗ (m − 1) ∗ nat ⌈ln α / ln 0.95⌉))
    unfolding power-mult by simp
  also have ... = 1 + log 2 (real (pro-size I.Ω)) + (4∗(m−1)∗ nat⌈ln α / ln 0.95⌉)
    using 1 by (subst log-mult) simp-all
 also have ... ≤ 1+log 2(2 powr (4∗log 2 n + 48 ∗ (log 2 (1/ε)+16)^2+ (55+60∗ln (1/δ_i))^3))+
```

```
  (4*(m−1)* nat⌈ln α / ln 0.95⌉)
   using 1 by (intro ereal-mono add-mono iffD2[OF log-le-cancel-iff] I.random-bit-count) auto
  also have ...=1+4*log 2 n+48*(log 2(1/ε)+16)²+(55+60*ln (1/δ_i))^3+(4*(m−1)*nat⌈ln
α/ln 0.95⌉)
   by (subst log-powr-cancel) auto
  also have ... ≤ 2^30 + 2^23*ln n+48*(log 2(1/ε)+16)² + 336*ln (1/δ) (is ?L1 ≤ ?R1)
  proof (cases stage-two)
   case True

   have −1 < (0::real) by simp
   also have ... ≤ ln α / ln 0.95
    using α-gt-0 α-le-1 by (intro divide-nonpos-neg) auto
   finally have 4: − 1 < ln α / ln 0.95 by simp

   have 5: − 1 / ln 0.95 ≤ (20::real)
    by (approximation 10)

   have (4*(m−1)*nat⌈ln α/ln 0.95⌉) = 4 * (real m−1) * of-int ⌈ln α/ln 0.95⌉
    using 4 m-gt-0 unfolding of-nat-mult by (subst of-nat-nat) auto
   also have ... ≤ 4 * (real m−1) * (ln α/ln 0.95 + 1)
    using m-gt-0 by (intro mult-left-mono) auto
   also have ... = 4 * (real m−1) * (−ln (ln n_0)/ln 0.95 + 1)
    using n-lbound True unfolding α-def
    by (subst ln-inverse[symmetric]) (simp-all add:inverse-eq-divide)
   also have ... = 4 * (real m − 1 ) * (ln (ln n_0) * (−1/ln 0.95) + 1)
    by simp
   also have ... ≤ 4 * (real m − 1 ) * (ln (ln n_0) * 20 + 1)
    using n-lbound m-gt-0 by (intro mult-left-mono add-mono 5) auto
   also have ... = 4 * (real (nat ⌈4 * ln (1 / δ) / ln (ln n_0)⌉)−1) * (ln (ln n_0) * 20 + 1)
    using True unfolding m-def by simp
   also have ... = 4 * (real-of-int ⌈4 * ln (1 / δ) / ln (ln n_0)⌉−1) * (ln (ln n_0) * 20 + 1)
    using m-gt-0-aux[OF True] by (subst of-nat-nat) simp-all
   also have ... ≤ 4 * (4 * ln (1 / δ) / ln (ln n_0)) * (ln (ln n_0) * 20 + 1)
    using n-lbound by (intro mult-left-mono mult-right-mono) auto
   also have ... ≤ 4 * (4 * ln (1 / δ) / ln (ln n_0)) * (ln (ln n_0) * 20 + ln (ln n_0))
    using δ-gt-0 δ-lt-1 n-lbound
   by (intro mult-left-mono mult-right-mono add-mono divide-nonneg-pos Rings.mult-nonneg-nonneg)
     simp-all
   also have ... = 336 * ln (1 / δ)
    using n-lbound by simp
   finally have 6: 4 * (m−1) * nat ⌈ln α/ln 0.95⌉ ≤ 336 * ln (1/δ)
    by simp

    have ?L1 =1+4*log 2 n+48*(log 2(1/ε)+16)²+(55+60*ln (ln n_0))^3+(4*(m−1)*nat⌈ln
α/ln 0.95⌉)
    using True unfolding δ_i-def by simp
   also have ... ≤ 1+4*log 2 n+48*(log 2(1/ε)+16)²+(180^3 * ln n + 180^3*exp 5) + 336 *
ln (1/δ)
    by (intro add-mono 6 2 ereal-mono order.refl)
   also have ... = (1+180^3*exp 5)+ (4/ln 2 + 180^3)*ln n+48*(log 2(1/ε)+16)²+ 336 * ln
(1/δ)
    by (simp add:log-def algebra-simps)
   also have ... ≤ 2^30 + 2^23*ln n+48*(log 2(1/ε)+16)²+ 336 * ln (1/δ)
    using n-gt-0 by (intro add-mono ereal-mono 3 order.refl mult-right-mono) auto
   finally show ?thesis by simp
  next
   case False
   hence 1 / δ ≤ ln n_0
```

```
    using δ-gt-0 n-lbound
    unfolding stage-two-def not-less by (simp add:divide-simps ac-simps)
  hence 7: ln (1 / δ) ≤ ln (ln n₀)
    using n-lbound δ-gt-0 δ-lt-1
    by (intro iffD2[OF ln-le-cancel-iff]) auto

  have 8: 0 ≤ 336*ln (1/δ)
    using δ-gt-0 δ-lt-1 by auto

  have ?L1 = 1 + 4 * log 2 (real n) + 48 * (log 2 (1 / ε) + 16)² + (55 + 60 * ln (1 / δ)) ^ 3
    using False unfolding δᵢ-def m-def by simp
  also have ... ≤ 1 + 4 * log 2 (real n) + 48 * (log 2 (1 / ε) + 16)² + (55 + 60 * ln (ln
n₀)) ^3
    using δ-gt-0 δ-lt-1
    by (intro add-mono order.refl ereal-mono power-mono mult-left-mono add-nonneg-nonneg 7)
auto
  also have ... ≤ 1+4*log 2(real n)+48*(log 2 (1 / ε)+16)²+(180^3*ln (real n) + 180 ^ 3 *
exp 5)
    by (intro add-mono ereal-mono 2 order.refl)
  also have ... = (1+180^3*exp 5)+ (4/ln 2 + 180^3)*ln n+48*(log 2(1/ε)+16)²+ 0
    by (simp add:log-def algebra-simps)
  also have ... ≤ 2^30 + 2^23*ln n+48*(log 2(1/ε)+16)² + 336*ln (1/δ)
    using n-gt-0 by (intro add-mono ereal-mono 3 order.refl mult-right-mono 8) auto
  finally show ?thesis by simp
 qed
 also have ... = seed-space-usage (real n, ε, δ)
  unfolding seed-space-usage-def by simp
 finally show ?thesis by simp
qed
```

The following is an alternative form expressing the correctness and space usage theorems. If x is expression formed by *local.single* and *local.merge* operations. Then x requires *state-space-usage* (*real* n, ε, δ) bits to encode and *estimate* x approximates the count of the distinct universe elements in the expression.

For example:

estimate (*local.merge* (*local.single* ω *1*) (*local.merge* (*local.single* ω *5*) (*local.single* ω *1*))) approximates the cardinality of $\{1, 5, 1\}$ i.e. 2.

```
datatype sketch-tree = Single nat | Merge sketch-tree sketch-tree

fun eval :: nat ⇒ sketch-tree ⇒ state
 where
  eval ω (Single x) = single ω x |
  eval ω (Merge x y) = merge (eval ω x) (eval ω y)

fun sketch-tree-set :: sketch-tree ⇒ nat set
 where
  sketch-tree-set (Single x) = {x} |
  sketch-tree-set (Merge x y) = sketch-tree-set x ∪ sketch-tree-set y

theorem correctness:
 fixes X
 assumes sketch-tree-set t ⊆ {..<n}
 defines p ≡ pmf-of-set {..<pro-size Θ}
 defines X ≡ real (card (sketch-tree-set t))
 shows measure p {ω. |estimate (eval ω t) − X| > ε * X} ≤ δ (is ?L ≤ ?R)
proof −
 define A where A = sketch-tree-set t
```

have *X-eq*: $X = real\ (card\ A)$
unfolding *X-def A-def* **by** *simp*

have *0*:$eval\ \omega\ t = \nu\ \omega\ A$ **for** ω
unfolding *A-def* **using** *single-result merge-result*
by (*induction t*) (*auto simp del*:*merge.simps single.simps*)

have *1*: $A \subseteq \{..<n\}$
using *assms*(*1*) **unfolding** *A-def* **by** *blast*

have *2*: $A \neq \{\}$
unfolding *A-def* **by** (*induction t*) *auto*

show *?thesis*
unfolding *0 X-eq p-def* **by** (*intro estimate-result 1 2*)
qed

theorem *space-usage*:
assumes $\omega < pro\text{-}size\ \Theta$
shows
$bit\text{-}count\ (encode\text{-}state\ (eval\ \omega\ t)) \leq state\text{-}space\text{-}usage\ (real\ n,\ \varepsilon,\ \delta)$ (**is** *?A*)
$bit\text{-}count\ (encode\text{-}seed\ \omega) \leq seed\text{-}space\text{-}usage\ (real\ n,\ \varepsilon,\ \delta)$ (**is** *?B*)
proof−
define A **where** $A = sketch\text{-}tree\text{-}set\ t$

have *0*:$eval\ \omega\ t = \nu\ \omega\ A$ **for** ω
unfolding *A-def* **using** *single-result merge-result*
by (*induction t*) (*auto simp del*:*merge.simps single.simps*)

show *?A*
unfolding *0* **by** (*intro state-bit-count*)
show *?B*
using *random-bit-count*[*OF assms*] **by** *simp*
qed

end

The functions *state-space-usage* and *seed-space-usage* are exact bounds on the space usage for the state and the seed. The following establishes asymptotic bounds with respect to the limit $n, \delta^{-1}, \varepsilon^{-1} \to \infty$.

context
begin

Some local notation to ease proofs about the asymptotic space usage of the algorithm:

private definition $n\text{-}of :: real \times real \times real \Rightarrow real$ **where** $n\text{-}of = (\lambda(n,\ \varepsilon,\ \delta).\ n)$
private definition $\delta\text{-}of :: real \times real \times real \Rightarrow real$ **where** $\delta\text{-}of = (\lambda(n,\ \varepsilon,\ \delta).\ \delta)$
private definition $\varepsilon\text{-}of :: real \times real \times real \Rightarrow real$ **where** $\varepsilon\text{-}of = (\lambda(n,\ \varepsilon,\ \delta).\ \varepsilon)$

private abbreviation $F :: (real \times real \times real)\ filter$
where $F \equiv (at\text{-}top \times_F at\text{-}right\ 0 \times_F at\text{-}right\ 0)$

private lemma *var-simps*:
$n\text{-}of = fst$
$\varepsilon\text{-}of = (\lambda x.\ fst\ (snd\ x))$
$\delta\text{-}of = (\lambda x.\ snd\ (snd\ x))$
unfolding *n-of-def ε-of-def δ-of-def* **by** (*auto simp add*:*case-prod-beta*)

private lemma *evt-n*: $eventually\ (\lambda x.\ n\text{-}of\ x \geq n)\ F$

unfolding *var-simps* **by** (*intro eventually-prod1' eventually-prod2' eventually-ge-at-top*)
(*simp add:prod-filter-eq-bot*)

private lemma *evt-n-1*: $\forall_F\ x\ in\ F.\ 0 \le ln\ (n\text{-}of\ x)$
by (*intro eventually-mono*[*OF evt-n*[*of 1*]] *ln-ge-zero*) *simp*

private lemma *evt-n-2*: $\forall_F\ x\ in\ F.\ 0 \le ln\ (ln\ (n\text{-}of\ x))$
using *order-less-le-trans*[*OF exp-gt-zero*]
by (*intro eventually-mono*[*OF evt-n*[*of exp 1*]] *ln-ge-zero iffD2*[*OF ln-ge-iff*]) *auto*

private lemma *evt-ε*: *eventually* $(\lambda x.\ 1/\varepsilon\text{-}of\ x \ge \varepsilon \wedge \varepsilon\text{-}of\ x > 0)\ F$
unfolding *var-simps* **by** (*intro eventually-prod1' eventually-prod2' eventually-conj*
real-inv-at-right-0-inf eventually-at-right-less) (*simp-all add:prod-filter-eq-bot*)

private lemma *evt-δ*: *eventually* $(\lambda x.\ 1/\delta\text{-}of\ x \ge \delta \wedge \delta\text{-}of\ x > 0)\ F$
unfolding *var-simps* **by** (*intro eventually-prod1' eventually-prod2' eventually-conj*
real-inv-at-right-0-inf eventually-at-right-less) (*simp-all add:prod-filter-eq-bot*)

private lemma *evt-δ-1*: $\forall_F\ x\ in\ F.\ 0 \le ln\ (1\ /\ \delta\text{-}of\ x)$
by (*intro eventually-mono*[*OF evt-δ*[*of 1*]] *ln-ge-zero*) *simp*

theorem *asymptotic-state-space-complexity*:
state-space-usage $\in O[F](\lambda(n,\ \varepsilon,\ \delta).\ ln\ (1/\delta)/\varepsilon\hat{}2 + ln\ (ln\ n))$
(**is** - $\in O[?F](?rhs)$)
proof −
have *0*:$(\lambda x.\ 1) \in O[?F](\lambda x.\ ln\ (1\ /\ \delta\text{-}of\ x))$
using *order-less-le-trans*[*OF exp-gt-zero*]
by (*intro landau-o.big-mono eventually-mono*[*OF evt-δ*[*of exp 1*]])
(*auto intro*!: *iffD2*[*OF ln-ge-iff*] *simp add:abs-ge-iff*)

have *1*:$(\lambda x.\ 1) \in O[?F](\lambda x.\ ln\ (n\text{-}of\ x))$
using *order-less-le-trans*[*OF exp-gt-zero*]
by (*intro landau-o.big-mono eventually-mono*[*OF evt-n*[*of exp 1*]])
(*auto intro*!:*iffD2*[*OF ln-ge-iff*] *simp add:abs-ge-iff*)

have $(\lambda x.\ ((ln\ (1/\delta\text{-}of\ x)+1)*\ (1/\varepsilon\text{-}of\ x)^2)) \in O[?F](\lambda x.\ ln(1/\delta\text{-}of\ x)*\ (1/\varepsilon\text{-}of\ x)^2)$
by (*intro landau-o.mult sum-in-bigo 0*) *simp-all*
hence *2*: $(\lambda x.\ 2\hat{}40*((ln\ (1/\delta\text{-}of\ x)+1)*\ (1/\varepsilon\text{-}of\ x)^2)) \in O[?F](\lambda x.\ ln(1/\delta\text{-}of\ x)*\ (1/\varepsilon\text{-}of\ x)^2)$
unfolding *cmult-in-bigo-iff* **by** *simp*

have *3*: $(1::real) \le exp\ 2$
by (*approximation 5*)

have $(\lambda x.\ ln\ (n\text{-}of\ x)\ /\ ln\ 2 + 3) \in O[?F](\lambda x.\ ln\ (n\text{-}of\ x))$
using *1* **by** (*intro sum-in-bigo*) *simp-all*
hence $(\lambda x.\ ln\ (ln\ (n\text{-}of\ x)\ /\ ln\ 2 + 3)) \in O[?F](\lambda x.\ ln\ (ln\ (n\text{-}of\ x)))$
using *order-less-le-trans*[*OF exp-gt-zero*] *order-trans*[*OF 3*]
by (*intro landau-ln-2*[**where** *a=2*] *eventually-mono*[*OF evt-n*[*of exp 2*]])
(*auto intro*!:*iffD2*[*OF ln-ge-iff*] *add-nonneg-nonneg divide-nonneg-pos*)
hence *4*: $(\lambda x.\ log\ 2\ (log\ 2\ (n\text{-}of\ x) + 3)) \in O[?F](\lambda x.\ ln(ln(n\text{-}of\ x)))$
unfolding *log-def* **by** *simp*

have *5*: $\forall_F\ x\ in\ ?F.\ 0 \le ln\ (1\ /\ \delta\text{-}of\ x) * (1\ /\ \varepsilon\text{-}of\ x)^2$
by (*intro eventually-mono*[*OF eventually-conj*[*OF evt-δ-1 evt-ε*[*of 1*]]]) *auto*

have *state-space-usage* = $(\lambda x.\ state\text{-}space\text{-}usage\ (n\text{-}of\ x,\ \varepsilon\text{-}of\ x,\ \delta\text{-}of\ x))$
by (*simp add:case-prod-beta' n-of-def δ-of-def ε-of-def*)
also have ... = $(\lambda x.\ 2\ \hat{}\ 40\ *\ ((ln\ (1\ /\ (\delta\text{-}of\ x)) + 1)*\ (1/\varepsilon\text{-}of\ x)^2) + log\ 2\ (log\ 2\ (n\text{-}of\ x)+3))$

```
    unfolding state-space-usage-def by (simp add:divide-simps)
  also have ... ∈ O[?F](λx. ln (1/δ-of x)* (1/ε-of x)^2 + ln (ln (n-of x)))
    by (intro landau-sum 2 4 5 evt-n-2)
  also have ... = O[?F](?rhs)
    by (simp add:case-prod-beta' n-of-def δ-of-def ε-of-def divide-simps)
  finally show ?thesis by simp
qed

theorem asymptotic-seed-space-complexity:
  seed-space-usage ∈ O[F](λ(n, ε, δ). ln (1/δ)+ln (1/ε)^2 + ln n)
  (is - ∈ O[?F](?rhs))
proof −
  have 0: ∀ F x in ?F. 0 ≤ (ln (1 / ε-of x))^2
    by simp

  have 1: ∀ F x in ?F. 0 ≤ ln (1 / δ-of x) + (ln (1 / ε-of x))^2
    by (intro eventually-mono[OF eventually-conj[OF evt-δ-1 0]] add-nonneg-nonneg) auto

  have 2: (λx. 1) ∈ O[?F](λx. ln (1 / ε-of x))
    using order-less-le-trans[OF exp-gt-zero]
    by (intro landau-o.big-mono eventually-mono[OF evt-ε[of exp 1]])
      (auto intro!:iffD2[OF ln-ge-iff] simp add:abs-ge-iff)

  have (λx. 1) ∈ O[at-top ×F at-right 0 ×F at-right 0](λx. ln (n-of x))
    using order-less-le-trans[OF exp-gt-zero]
    by (intro landau-o.big-mono eventually-mono[OF evt-n[of exp 1]])
      (auto intro!:iffD2[OF ln-ge-iff] simp add:abs-ge-iff)
  hence 3: (λx. 1) ∈ O[?F](λx. ln (1 / δ-of x) + (ln (1 / ε-of x))^2 + ln (n-of x))
    by (intro landau-sum-2 1 evt-n-1 0 evt-δ-1) simp
  have 4: (λx. ln (n-of x)) ∈ O[?F](λx. ln (1 / δ-of x) + (ln (1 / ε-of x))^2 + ln (n-of x))
    by (intro landau-sum-2 1 evt-n-1) simp
  have (λx. log 2 (1 / ε-of x) + 16) ∈ O[?F](λx. ln (1 / ε-of x))
    using 2 unfolding log-def by (intro sum-in-bigo) simp-all
  hence 5: (λx. (log 2 (1 / ε-of x) + 16)^2) ∈ O[?F](λx. ln (1/δ-of x)+(ln (1/ε-of x))^2)
    using 0 unfolding power2-eq-square by (intro landau-sum-2 landau-o.mult evt-δ-1) simp-all
  have 6: (λx. (log 2 (1 / ε-of x) + 16)^2) ∈ O[?F](λx. ln (1/δ-of x)+(ln (1/ε-of x))^2+ln (n-of
x))
    by (intro landau-sum-1[OF - - 5] 1 evt-n-1)
  have 7: (λx. ln (1/δ-of x)) ∈ O[?F](λx. ln (1/δ-of x)+(ln (1/ε-of x))^2+ln (n-of x))
    by (intro landau-sum-1 1 evt-δ-1 0 evt-n-1) simp

  have seed-space-usage = (λx. seed-space-usage (n-of x, ε-of x, δ-of x))
    by (simp add:case-prod-beta' n-of-def δ-of-def ε-of-def)
  also have ... = (λx. 2^30+2^23*ln (n-of x)+48*(log 2 (1/(ε-of x))+16)^2 + 336 * ln (1 / δ-of
x))
    unfolding seed-space-usage-def by (simp add:divide-simps)
  also have ... ∈ O[?F](λx. ln (1/δ-of x)+ln (1/ε-of x)^2 + ln (n-of x))
    using 3 4 6 7 by (intro sum-in-bigo) simp-all
  also have ... = O[?F](?rhs)
    by (simp add:case-prod-beta' n-of-def δ-of-def ε-of-def)
  finally show ?thesis by simp
qed

definition space-usage x = state-space-usage x + seed-space-usage x

theorem asymptotic-space-complexity:
  space-usage ∈ O[at-top ×F at-right 0 ×F at-right 0](λ(n, ε, δ). ln (1/δ)/ε^2 + ln n)
proof −
```

let *?f1* = (λx. *ln* (*1*/*δ-of* x)∗(*1*/*ε-of* x^*2*)+*ln* (*ln* (*n-of* x)))
let *?f2* = (λx. *ln*(*1*/*δ-of* x)+*ln*(*1*/*ε-of* x)^*2*+*ln* (*n-of* x))

have *0*: ∀ *F* x *in* *F*. *0* ≤ (*1* / (*ε-of* x)²)
  unfolding *var-simps* **by** (*intro eventually-prod1' eventually-prod2' eventually-inv*)
    (*simp-all add*:*prod-filter-eq-bot eventually-nonzero-simps*)

have *1*: ∀ *F* x *in* *F*. *0* ≤ *ln* (*1* / *δ-of* x) ∗ (*1* / (*ε-of* x)²)
  by (*intro eventually-mono*[*OF eventually-conj*[*OF evt-δ-1 0*]] *mult-nonneg-nonneg*) *auto*

have *2*: ∀ *F* x *in* *F*. *0* ≤ *ln* (*1* / *δ-of* x) ∗ (*1* / (*ε-of* x)²) + *ln* (*ln* (*n-of* x))
  by (*intro eventually-mono*[*OF eventually-conj*[*OF 1 evt-n-2*]] *add-nonneg-nonneg*) *auto*

have *3*: ∀ *F* x *in* *F*. *0* ≤ *ln* (*1* / (*ε-of* x)²)
  unfolding *power-one-over*[*symmetric*]
  by (*intro eventually-mono*[*OF evt-ε*[*of 1*]] *ln-ge-zero*) *simp*

have *4*: ∀ *F* x *in* *F*. *0* ≤ *ln* (*1* / *δ-of* x) + (*ln* (*1* / *ε-of* x))² + *ln* (*n-of* x)
  by (*intro eventually-mono*[*OF eventually-conj*[*OF evt-n-1 eventually-conj*[*OF evt-δ-1 3*]]]
      add-nonneg-nonneg) *auto*

have *5*: (λ-. *1*) ∈ *O*[*F*](λx. *1* / (*ε-of* x)²)
  unfolding *var-simps* **by** (*intro bigo-prod-1 bigo-prod-2 bigo-inv*)
    (*simp-all add*:*power-divide prod-filter-eq-bot*)

have *6*: (λ-. *1*) ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x))
  unfolding *var-simps*
  by (*intro bigo-prod-1 bigo-prod-2 bigo-inv*) (*simp-all add*:*prod-filter-eq-bot*)

have *7*: *state-space-usage* ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) ∗ (*1* / (*ε-of* x)²) + *ln* (*ln* (*n-of* x)))
  using *asymptotic-state-space-complexity* **unfolding** *δ-of-def ε-of-def n-of-def*
  by (*simp add*:*case-prod-beta'*)

have *8*: *seed-space-usage* ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) + (*ln* (*1* / *ε-of* x))² + *ln* (*n-of* x))
  using *asymptotic-seed-space-complexity* **unfolding** *δ-of-def ε-of-def n-of-def*
  by (*simp add*:*case-prod-beta'*)

have *9*: (λx. *ln* (*n-of* x)) ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) ∗ (*1* / (*ε-of* x)²) + *ln* (*n-of* x))
  by (*intro landau-sum-2 evt-n-1 1*) *simp*

have (λx. (*ln* (*1* / *ε-of* x))²) ∈ *O*[*F*](λx. *1* / *ε-of* x^*2*)
  unfolding *var-simps*
  by (*intro bigo-prod-1 bigo-prod-2 bigo-inv*) (*simp-all add*:*power-divide prod-filter-eq-bot*)
hence *10*: (λx. (*ln* (*1* / *ε-of* x))²) ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) ∗ (*1* / *ε-of* x^*2*) + *ln* (*n-of* x))
  by (*intro landau-sum-1 evt-n-1 1 landau-o.big-mult-1' 6*)
have *11*: (λx. *ln* (*1* / *δ-of* x)) ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) ∗ (*1* / *ε-of* x^*2*) + *ln* (*n-of* x))
  by (*intro landau-sum-1 evt-n-1 1 landau-o.big-mult-1 5*) *simp*
have *12*: (λx. *ln* (*1*/*δ-of* x) ∗ (*1*/*ε-of* x^*2*)) ∈ *O*[*F*](λx. *ln* (*1*/*δ-of* x)∗(*1*/*ε-of* x^*2*)+*ln* (*n-of* x))
  by (*intro landau-sum-1 1 evt-n-1*) *simp*

have (λx. *ln* (*ln* (*n-of* x))) ∈ *O*[*F*](λx. *ln* (*n-of* x))
  unfolding *var-simps* **by** (*intro bigo-prod-1 bigo-prod-2*) (*simp-all add*:*prod-filter-eq-bot*)
hence *13*: (λx. *ln* (*ln* (*n-of* x))) ∈ *O*[*F*](λx. *ln* (*1* / *δ-of* x) ∗ (*1* / *ε-of* x^*2*) + *ln* (*n-of* x))
  by (*intro landau-sum-2 evt-n-1 1*)

have *space-usage* = (λx. *state-space-usage* x + *seed-space-usage* x)
  unfolding *space-usage-def* **by** *simp*
also have ... ∈ *O*[*F*](λx. *?f1* x + *?f2* x)

```
    by (intro landau-sum 2 4 7 8)
  also have ... ⊆ O[F](λx. ln (1 / δ-of x) * (1/ε-of x^2) + ln (n-of x))
    by (intro landau-o.big.subsetI sum-in-bigo 9 10 11 12 13)
  also have ... = O[F](λ(n, ε, δ). ln (1/δ)/ε^2 + ln n)
    unfolding δ-of-def ε-of-def n-of-def
    by (simp add:case-prod-beta')
  finally show ?thesis by simp
qed

end

unbundle no intro-cong-syntax

end
```

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002.

[3] J. Błasiok. Optimal streaming and tracking distinct elements with high probability. *ACM Trans. Algorithms*, 16(1):3:1–3:28, 2020.

[4] P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[5] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 281–291, 2001.

[6] V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *J. ACM*, 56(4), jul 2009.

[7] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, 2010.

[8] E. Karayel. Finite fields. *Archive of Formal Proofs*, June 2022. https://isa-afp.org/entries/Finite_Fields.html, Formal proof development.

[9] E. Karayel. Formalization of randomized approximation algorithms for frequency moments. *Archive of Formal Proofs*, April 2022. https://isa-afp.org/entries/Frequency_Moments.html, Formal proof development.

[10] E. Karayel. Expander graphs. *Archive of Formal Proofs*, March 2023. https://isa-afp.org/entries/Expander_Graphs.html, Formal proof development.

[11] D. Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 167–175, USA, 2004. Society for Industrial and Applied Mathematics.