Formalization of Randomized Approximation Algorithms for Frequency Moments

Emin Karayel

March 29, 2023

Abstract

In 1999 Alon et. al. introduced the still active research topic of approximating the frequency moments of a data stream using randomized algorithms with minimal space usage. This includes the problem of estimating the cardinality of the stream elements—the zeroth frequency moment. But, also higher-order frequency moments that provide information about the skew of the data stream. (The k-th frequency moment of a data stream is the sum of the k-th powers of the occurrence counts of each element in the stream.) This entry formalizes three randomized algorithms for the approximation of F_0, F_2 and F_k for $k \geq 3$ based on [1, 2] and verifies their expected accuracy, success probability and space usage.

Contents

1 Preliminary Results

```
theory Frequency-Moments-Preliminary-Results
  imports
    HOL.Transcendental
    HOL−Computational-Algebra.Primes
    HOL−Library.Extended-Real
    HOL−Library.Multiset
    HOL−Library.Sublist
    Prefix-Free-Code-Combinators.Prefix-Free-Code-Combinators
    Bertrands-Postulate.Bertrand
begin
```

This section contains various preliminary results.

```
lemma card-ordered-pairs:
  fixes M :: ('a ::linorder) set
  assumes finite M
  shows 2 ∗ card {(x,y) ∈ M × M. x < y} = card M ∗ (card M − 1)
proof −
  have a: finite (M × M) using assms by simp

  have inj-swap: inj (λx. (snd x, fst x))
    by (rule inj-onI, simp add: prod-eq-iff)

  have 2 ∗ card {(x,y) ∈ M × M. x < y} =
    card {(x,y) ∈ M × M. x < y} + card ((λx. (snd x, fst x))`{(x,y) ∈ M × M. x
< y})
    by (simp add: card-image[OF inj-on-subset[OF inj-swap]])
  also have ... = card {(x,y) ∈ M × M. x < y} + card {(x,y) ∈ M × M. y < x}
    by (auto intro: arg-cong[where f=card] simp add:set-eq-iff image-iff)
  also have ... = card ({(x,y) ∈ M × M. x < y} ∪ {(x,y) ∈ M × M. y < x})
    by (intro card-Un-disjoint[symmetric] a finite-subset[where B=M × M] sub-
setI) auto
  also have ... = card ((M × M) − {(x,y) ∈ M × M. x = y})
    by (auto intro: arg-cong[where f=card] simp add:set-eq-iff)
  also have ... = card (M × M) − card {(x,y) ∈ M × M. x = y}
    by (intro card-Diff-subset a finite-subset[where B=M × M] subsetI) auto
  also have ... = card M ^ 2 − card ((λx. (x,x)) ` M)
    using assms
    by (intro arg-cong2[where f=(−)] arg-cong[where f=card])
      (auto simp:power2-eq-square set-eq-iff image-iff)
  also have ... = card M ^ 2 − card M
    by (intro arg-cong2[where f=(−)] card-image inj-onI, auto)
  also have ... = card M ∗ (card M − 1)
```

```
    by (cases card M ≥ 0, auto simp:power2-eq-square algebra-simps)
  finally show ?thesis by simp
qed

lemma ereal-mono: x ≤ y ⟹ ereal x ≤ ereal y
  by simp

lemma log-mono: a > 1 ⟹ x ≤ y ⟹ 0 < x ⟹ log a x ≤ log a y
  by (subst log-le-cancel-iff, auto)

lemma abs-ge-iff: ((x::real) ≤ abs y) = (x ≤ y ∨ x ≤ −y)
  by linarith

lemma count-list-gr-1:
  (x ∈ set xs) = (count-list xs x ≥ 1)
  by (induction xs, simp, simp)

lemma count-list-append: count-list (xs@ys) v = count-list xs v + count-list ys v
  by (induction xs, simp, simp)

lemma count-list-lt-suffix:
  assumes suffix a b
  assumes x ∈ {b ! i| i. i <  length b − length a}
  shows  count-list a x < count-list b x
proof −
  have length a ≤ length b using assms(1)
    by (simp add: suffix-length-le)
  hence x ∈ set (nths b {i. i < length b − length a})
    using assms diff-commute by (auto simp add:set-nths)
  hence a:x ∈ set (take (length b − length a) b)
    by (subst (asm) lessThan-def[symmetric], simp)
  have b = (take (length b − length a) b)@drop (length b − length a) b
    by simp
  also have ... = (take (length b − length a) b)@a
    using assms(1) suffix-take by auto
  finally have b:b = (take (length b − length a) b)@a by simp

  have count-list a x < 1 + count-list a x by simp
  also have ... ≤ count-list (take (length b − length a) b) x + count-list a x
    using a count-list-gr-1
    by (intro add-mono, fast, simp)
  also have ... = count-list b x
    using b count-list-append by metis
  finally show ?thesis by simp
qed

lemma suffix-drop-drop:
  assumes x ≥ y
  shows suffix (drop x a) (drop y a)
```

```
proof -
  have drop y a = take (x - y) (drop y a)@drop (x- y) (drop y a)
    by (subst append-take-drop-id, simp)
  also have ... = take (x-y) (drop y a)@drop x a
    using assms by simp
  finally have drop y a = take (x-y) (drop y a)@drop x a by simp
  thus ?thesis
    by (auto simp add:suffix-def)
qed

lemma count-list-card: count-list xs x = card {k. k < length xs ∧ xs ! k = x}
proof -
  have count-list xs x = length (filter ((=) x) xs)
    by (induction xs, simp, simp)
  also have ... = card {k. k < length xs ∧ xs ! k = x}
    by (subst length-filter-conv-card, metis)
  finally show ?thesis by simp
qed

lemma card-gr-1-iff:
  assumes finite S  x ∈ S  y ∈ S  x ≠ y
  shows card S > 1
  using assms card-le-Suc0-iff-eq leI by auto

lemma count-list-ge-2-iff:
  assumes y < z
  assumes z < length xs
  assumes xs ! y = xs ! z
  shows count-list xs (xs ! y) > 1
proof -
  have  1 < card {k. k < length xs ∧ xs ! k = xs ! y}
    using assms by (intro card-gr-1-iff[where x=y and y=z], auto)

  thus ?thesis
    by (simp add: count-list-card)
qed
```

Results about multisets and sorting

This is a induction scheme over the distinct elements of a multisets: We can represent each multiset as a sum like: *replicate-mset* n_1 x_1 + *replicate-mset* n_2 x_2 + ... + *replicate-mset* n_k x_k where the x_i are distinct.

```
lemma disj-induct-mset:
  assumes P {#}
  assumes ⋀n M x. P M ⟹ ¬(x ∈# M) ⟹ n > 0 ⟹ P (M + replicate-mset
n x)
  shows P M
proof (induction size M arbitrary: M rule:nat-less-induct)
  case 1
```

show *?case*
proof (*cases* $M = \{\#\}$)
case *True*
then show *?thesis* **using** *assms* **by** *simp*
next
case *False*
then obtain x **where** *x-def*: $x \in\# M$ **using** *multiset-nonemptyE* **by** *auto*
define *M1* **where** *M1* = *M* − *replicate-mset* (*count M x*) *x*
then have *M-def*: *M* = *M1* + *replicate-mset* (*count M x*) *x*
by (*metis count-le-replicate-mset-subset-eq dual-order.refl subset-mset.diff-add*)
have *size M1* < *size M*
by (*metis M-def x-def count-greater-zero-iff less-add-same-cancel1 size-replicate-mset size-union*)
hence *P M1* **using** *1* **by** *blast*
then show *P M*
apply (*subst M-def*, *rule assms*(*2*), *simp*)
by (*simp add*:*M1-def x-def count-eq-zero-iff*[*symmetric*])+
qed
qed

lemma *prod-mset-conv*:
fixes $f :: 'a \Rightarrow 'b::\{comm\text{-}monoid\text{-}mult\}$
shows *prod-mset* (*image-mset f A*) = *prod* ($\lambda x.\ f\ x$ ^(*count A x*)) (*set-mset A*)
proof (*induction A rule*: *disj-induct-mset*)
case *1*
then show *?case* **by** *simp*
next
case (*2 n M x*)
moreover have *count M x* = *0* **using** *2* **by** (*simp add*: *count-eq-zero-iff*)
moreover have $\bigwedge y.\ y \in set\text{-}mset\ M \Longrightarrow y \neq x$ **using** *2* **by** *blast*
ultimately show *?case* **by** (*simp add*:*algebra-simps*)
qed

There is a version *sum-list-map-eq-sum-count* but it doesn't work if the function maps into the reals.

lemma *sum-list-eval*:
fixes $f :: 'a \Rightarrow 'b::\{ring,semiring\text{-}1\}$
shows *sum-list* (*map f xs*) = ($\sum x \in set\ xs.\ of\text{-}nat\ (count\text{-}list\ xs\ x) * f\ x$)
proof −
define *M* **where** *M* = *mset xs*
have *sum-mset* (*image-mset f M*) = ($\sum x \in set\text{-}mset\ M.\ of\text{-}nat\ (count\ M\ x) * f\ x$)
proof (*induction M rule*:*disj-induct-mset*)
case *1*
then show *?case* **by** *simp*
next
case (*2 n M x*)
have *a*:$\bigwedge y.\ y \in set\text{-}mset\ M \Longrightarrow y \neq x$ **using** *2*(*2*) **by** *blast*
show *?case* **using** *2* **by** (*simp add*:*a count-eq-zero-iff*[*symmetric*])

qed
moreover have $\bigwedge x.$ *count-list xs x* = *count* (*mset xs*) *x*
by (*induction xs*, *simp*, *simp*)
ultimately show *?thesis*
by (*simp add*:*M-def sum-mset-sum-list*[*symmetric*])
qed

lemma *prod-list-eval*:
fixes *f* :: *'a* ⇒ *'b*::{*ring*,*semiring-1*,*comm-monoid-mult*}
shows *prod-list* (*map f xs*) = ($\prod$ *x* ∈ *set xs*. (*f x*)^(*count-list xs x*))
proof −
define *M* **where** *M* = *mset xs*
have *prod-mset* (*image-mset f M*) = ($\prod$ *x* ∈ *set-mset M*. *f x* ^ (*count M x*))
proof (*induction M rule*:*disj-induct-mset*)
case *1*
then show *?case* **by** *simp*
next
case (*2 n M x*)
have *a*:$\bigwedge y.$ *y* ∈ *set-mset M* ⟹ *y* ≠ *x* **using** *2*(*2*) **by** *blast*
have *b*:*count M x* = *0* **using** *2* **by** (*subst count-eq-zero-iff*) *blast*
show *?case* **using** *2* **by** (*simp add*:*a b mult.commute*)
qed
moreover have $\bigwedge x.$ *count-list xs x* = *count* (*mset xs*) *x*
by (*induction xs*, *simp*, *simp*)
ultimately show *?thesis*
by (*simp add*:*M-def prod-mset-prod-list*[*symmetric*])
qed

lemma *sorted-sorted-list-of-multiset*: *sorted* (*sorted-list-of-multiset M*)
by (*induction M*, *auto simp*:*sorted-insort*)

lemma *count-mset*: *count* (*mset xs*) *a* = *count-list xs a*
by (*induction xs*, *auto*)

lemma *swap-filter-image*: *filter-mset g* (*image-mset f A*) = *image-mset f* (*filter-mset* (*g* ∘ *f*) *A*)
by (*induction A*, *auto*)

lemma *list-eq-iff*:
assumes *mset xs* = *mset ys*
assumes *sorted xs*
assumes *sorted ys*
shows *xs* = *ys*
using *assms properties-for-sort* **by** *blast*

lemma *sorted-list-of-multiset-image-commute*:
assumes *mono f*
shows *sorted-list-of-multiset* (*image-mset f M*) = *map f* (*sorted-list-of-multiset M*)

proof −
  have *sorted* (*sorted-list-of-multiset* (*image-mset f M*))
    by (*simp add*:*sorted-sorted-list-of-multiset*)
  moreover have *sorted-wrt* ($\lambda x\ y.\ f\ x \leq f\ y$) (*sorted-list-of-multiset M*)
    by (*rule sorted-wrt-mono-rel*[**where** $P{=}\lambda x\ y.\ x \leq y$])
      (*auto intro*: *monoD*[*OF assms*] *sorted-sorted-list-of-multiset*)
  hence *sorted* (*map f* (*sorted-list-of-multiset M*))
    by (*subst sorted-wrt-map*)
  ultimately show *?thesis*
    by (*intro list-eq-iff*, *auto*)
qed

Results about rounding and floating point numbers

lemma *round-down-ge*:
  $x \leq$ *round-down prec x* + *2 powr* (−*prec*)
  using *round-down-correct* **by** (*simp*, *meson diff-diff-eq diff-eq-diff-less-eq*)

lemma *truncate-down-ge*:
  $x \leq$ *truncate-down prec x* + *abs x* ∗ *2 powr* (−*prec*)
proof (*cases abs x* > *0*)
  case *True*
  have $x \leq$ *round-down* (*int prec* − ⌊*log 2* |*x*|⌋) *x* + *2 powr* (−*real-of-int*(*int prec* − ⌊*log 2* |*x*|⌋))
    by (*rule round-down-ge*)
  also have ... ≤ *truncate-down prec x* + *2 powr* (⌊*log 2* |*x*|⌋) ∗ *2 powr* (−*real prec*)
    by (*rule add-mono*, *simp-all add*:*powr-add*[*symmetric*] *truncate-down-def*)
  also have ... ≤ *truncate-down prec x* + |*x*| ∗ *2 powr* (−*real prec*)
    using *True*
    by (*intro add-mono mult-right-mono*, *simp-all add*:*le-log-iff*[*symmetric*])
  finally show *?thesis* **by** *simp*
next
  case *False*
  then show *?thesis* **by** *simp*
qed

lemma *truncate-down-pos*:
  assumes $x \geq 0$
  shows *x* ∗ (*1* − *2 powr* (−*prec*)) ≤ *truncate-down prec x*
  by (*simp add*:*right-diff-distrib diff-le-eq*)
   (*metis truncate-down-ge assms abs-of-nonneg*)

lemma *truncate-down-eq*:
  assumes *truncate-down r x* = *truncate-down r y*
  shows *abs* (*x*−*y*) ≤ *max* (*abs x*) (*abs y*) ∗ *2 powr* (−*real r*)
proof −
  have *x* − *y* ≤ *truncate-down r x* + *abs x* ∗ *2 powr* (−*real r*) − *y*
    by (*rule diff-right-mono*, *rule truncate-down-ge*)
  also have ... ≤ *y* + *abs x* ∗ *2 powr* (−*real r*) − *y*

using *truncate-down-le*
by (*intro diff-right-mono add-mono*, *subst assms*(*1*), *simp-all*)
also have ... ≤ *abs x* ∗ *2 powr* (−*real r*) **by** *simp*
also have ... ≤ *max* (*abs x*) (*abs y*) ∗ *2 powr* (−*real r*) **by** *simp*
finally have *a*:*x* − *y* ≤ *max* (*abs x*) (*abs y*) ∗ *2 powr* (−*real r*) **by** *simp*

have *y* − *x* ≤ *truncate-down r y* + *abs y* ∗ *2 powr* (−*real r*) − *x*
by (*rule diff-right-mono*, *rule truncate-down-ge*)
also have ... ≤ *x* + *abs y* ∗ *2 powr* (−*real r*) − *x*
using *truncate-down-le*
by (*intro diff-right-mono add-mono*, *subst assms*(*1*)[*symmetric*], *auto*)
also have ... ≤ *abs y* ∗ *2 powr* (−*real r*) **by** *simp*
also have ... ≤ *max* (*abs x*) (*abs y*) ∗ *2 powr* (−*real r*) **by** *simp*
finally have *b*:*y* − *x* ≤ *max* (*abs x*) (*abs y*) ∗ *2 powr* (−*real r*) **by** *simp*

show *?thesis*
using *abs-le-iff a b* **by** *linarith*
qed

definition *rat-of-float* :: *float* ⇒ *rat* **where**
rat-of-float f = *of-int* (*mantissa f*) ∗
(*if exponent f* ≥ *0 then 2* ^ (*nat* (*exponent f*)) *else 1* / *2* ^ (*nat* (−*exponent f*)))

lemma *real-of-rat-of-float*: *real-of-rat* (*rat-of-float x*) = *real-of-float x*
proof −
have *real-of-rat* (*rat-of-float x*) = *mantissa x* ∗ (*2 powr* (*exponent x*))
by (*simp add*:*rat-of-float-def of-rat-mult of-rat-divide of-rat-power powr-realpow*[*symmetric*] *powr-minus-divide*)
also have ... = *real-of-float x*
using *mantissa-exponent* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *log-est*: *log 2* (*real n* + *1*) ≤ *n*
proof −
have *1* + *real n* = *real* (*n* + *1*)
by *simp*
also have ... ≤ *real* (*2* ^ *n*)
by (*intro of-nat-mono suc-n-le-2-pow-n*)
also have ... = *2 powr* (*real n*)
by (*simp add*:*powr-realpow*)
finally have *1* + *real n* ≤ *2 powr* (*real n*)
by *simp*
thus *?thesis*
by (*simp add*: *Transcendental.log-le-iff*)
qed

lemma *truncate-mantissa-bound*:

```
  abs (⌊x ∗ 2 powr (real r − real-of-int ⌊log 2 |x|⌋)⌋) ≤ 2 ^ (r+1) (is ?lhs ≤ -)
proof −
  define q where q = ⌊x ∗ 2 powr (real r − real-of-int (⌊log 2 |x|⌋))⌋

  have abs q ≤ 2 ^ (r + 1) if a:x > 0
  proof −
    have abs q = q
      using a by (intro abs-of-nonneg, simp add:q-def)
    also have ... ≤ x ∗ 2 powr (real r − real-of-int ⌊log 2 |x|⌋)
      unfolding q-def using of-int-floor-le by blast
    also have ... = x ∗ 2 powr real-of-int (int r − ⌊log 2 |x|⌋)
      by auto
    also have ... = 2 powr (log 2 x + real-of-int (int r − ⌊log 2 |x|⌋))
      using a by (simp add:powr-add)
    also have ... ≤ 2 powr (real r + 1)
      using a by (intro powr-mono, linarith+)
    also have ... = 2 ^ (r+1)
      by (subst powr-realpow[symmetric], simp-all add:add.commute)
    finally show abs q ≤ 2 ^ (r+1)
      by (metis of-int-le-iff of-int-numeral of-int-power)
  qed

  moreover have abs q ≤ (2 ^ (r + 1)) if a: x < 0
  proof −
    have −(2 ^ (r+1) + 1) = −(2 powr (real r + 1)+1)
      by (subst powr-realpow[symmetric], simp-all add: add.commute)
    also have  ... < −(2 powr (log 2 (− x) + (r − ⌊log 2 |x|⌋)) + 1)
      using a by (simp, linarith)
    also have ... = x ∗ 2 powr (r − ⌊log 2 |x|⌋) − 1
      using a by (simp add:powr-add)
    also have ... ≤ q
      by (simp add:q-def)
    also have ... = − abs q
      using a
      by (subst abs-of-neg, simp-all add: mult-pos-neg2 q-def)
    finally have −(2 ^ (r+1)+1) < − abs q using of-int-less-iff by fastforce
    hence −(2 ^ (r+1)) ≤ − abs q by linarith
    thus abs q ≤ 2^(r+1) by linarith
  qed

  moreover have x = 0 =⇒ abs q ≤ 2^(r+1)
    by (simp add:q-def)
  ultimately have abs q ≤ 2^(r+1)
    by fastforce
  thus ?thesis using q-def by blast
qed

lemma truncate-float-bit-count:
  bit-count (F_e (float-of (truncate-down r x))) ≤ 10 + 4 ∗ real r + 2∗log 2 (2 +
```

|*log 2* |*x*||)
  (**is** *?lhs* ≤ *?rhs*)
proof −
  define *m* **where** *m* = ⌊*x* ∗ *2 powr* (*real r* − *real-of-int* ⌊*log 2* |*x*|⌋)⌋
  define *e* **where** *e* = ⌊*log 2* |*x*|⌋ − *int r*

  have *a*: (*real-of-int* ⌊*log 2* |*x*|⌋ − *real r*) = *e*
    by (*simp add*:*e-def*)
  have *abs m* + *2* ≤ *2* ^ (*r* + *1*) + *2*^*1*
    using *truncate-mantissa-bound*
    by (*intro add-mono*, *simp-all add*:*m-def*)
  also have ... ≤ *2* ^ (*r*+*2*)
    by *simp*
  finally have *b*:*abs m* + *2* ≤ *2* ^ (*r*+*2*) **by** *simp*
  hence *real-of-int* (|*m*| + *2*) ≤ *real-of-int* (*4* ∗ *2* ^ *r*)
    by (*subst of-int-le-iff*, *simp*)
  hence |*real-of-int m*| + *2* ≤ *4* ∗ *2* ^ *r*
    by *simp*
  hence *c*:*log 2* (*real-of-int* (|*m*| + *2*)) ≤ *r*+*2*
    by (*simp add*: *Transcendental.log-le-iff powr-add powr-realpow*)

  have *real-of-int* (*abs e* + *1*) ≤ *real-of-int* |⌊*log 2* |*x*|⌋| + *real-of-int r* + *1*
    by (*simp add*:*e-def*)
  also have ... ≤ *1* + *abs* (*log 2* (*abs x*)) + *real-of-int r* + *1*
    by (*simp add*:*abs-le-iff*, *linarith*)
  also have ... ≤ (*real-of-int r*+ *1*) ∗ (*2* + *abs* (*log 2* (*abs x*)))
    by (*simp add*:*distrib-left distrib-right*)
  finally have *d*:*real-of-int* (*abs e* + *1*) ≤ (*real-of-int r*+ *1*) ∗ (*2* + *abs* (*log 2* (*abs x*))) **by** *simp*

  have *log 2* (*real-of-int* (*abs e* + *1*)) ≤ *log 2* (*real-of-int r* + *1*) + *log 2* (*2* + *abs* (*log 2* (*abs x*)))
    using *d* **by** (*simp add*: *log-mult*[*symmetric*])
  also have ... ≤ *r* + *log 2* (*2* + *abs* (*log 2* (*abs x*)))
    using *log-est* **by** (*intro add-mono*, *simp-all add*:*add.commute*)
  finally have *e*: *log 2* (*real-of-int* (*abs e* + *1*)) ≤ *r* + *log 2* (*2* + *abs* (*log 2* (*abs x*))) **by** *simp*

  have *?lhs* = *bit-count* (F_e (*float-of* (*real-of-int m* ∗ *2 powr real-of-int e*)))
    by (*simp add*:*truncate-down-def round-down-def m-def*[*symmetric*] *a*)
  also have ... ≤ *ereal* (*6* + (*2* ∗ *log 2* (*real-of-int* (|*m*| + *2*)) + *2* ∗ *log 2* (*real-of-int* (|*e*| + *1*))))
    using *float-bit-count-2* **by** *simp*
  also have ... ≤ *ereal* (*6* + (*2* ∗ *real* (*r*+*2*) + *2* ∗ (*r* + *log 2* (*2* + *abs* (*log 2* (*abs x*))))))
    using *c e*
    by (*subst ereal-less-eq*, *intro add-mono mult-left-mono*, *linarith*+)
  also have ... = *?rhs* **by** *simp*
  finally show *?thesis* **by** *simp*

qed

definition *prime-above* :: *nat* ⇒ *nat*
  where *prime-above n* = (*SOME x. x* ∈ {*n*..(*2*∗*n*+*2*)} ∧ *prime x*)

The term *prime-above n* returns a prime between n and $2 * n + 2$. Because of Bertrand's postulate there always is such a value. In a refinement of the algorithms, it may make sense to replace this with an algorithm, that finds such a prime exactly or approximately.

The definition is intentionally inexact, to allow refinement with various algorithms, without modifying the high-level mathematical correctness proof.

lemma *ex-subset*:
  assumes ∃ *x* ∈ *A*. *P x*
  assumes *A* ⊆ *B*
  shows ∃ *x* ∈ *B*. *P x*
  using *assms* **by** *auto*

lemma
  shows *prime-above-prime*: *prime* (*prime-above n*)
  and *prime-above-range*: *prime-above n* ∈ {*n*..(*2*∗*n*+*2*)}
proof −
  define *r* **where** *r* = (λ*x*. *x* ∈ {*n*..(*2*∗*n*+*2*)} ∧ *prime x*)
  have ∃ *x*. *r x*
  proof (*cases n*>*2*)
    case *True*
    hence *n*−*1* > *1* **by** *simp*
    hence ∃ *x* ∈ {(*n*−*1*)<..<(*2*∗(*n*−*1*))}. *prime x*
      using *bertrand* **by** *simp*
    moreover have {*n* − *1*<..<*2* ∗ (*n* − *1*)} ⊆ {*n*..*2* ∗ *n* + *2*}
      by (*intro subsetI*, *auto*)
    ultimately have ∃ *x* ∈ {*n*..(*2*∗*n*+*2*)}. *prime x*
      by (*rule ex-subset*)
    then show *?thesis* **by** (*simp add*:*r-def Bex-def*)
  next
    case *False*
    hence *2* ∈ {*n*..(*2*∗*n*+*2*)}
      by *simp*
    moreover have *prime* (*2*::*nat*)
      using *two-is-prime-nat* **by** *blast*
    ultimately have *r 2*
      using *r-def* **by** *simp*
    then show *?thesis* **by** (*rule exI*)
  qed
  moreover have *prime-above n* = (*SOME x*. *r x*)
    by (*simp add*:*prime-above-def r-def*)
  ultimately have *a*:*r* (*prime-above n*)
    using *someI-ex* **by** *metis*
  show *prime* (*prime-above n*)

using *a* **unfolding** *r-def* **by** *blast*
show *prime-above n* ∈ {*n*..(*2*∗*n*+*2*)}
using *a* **unfolding** *r-def* **by** *blast*
qed

lemma *prime-above-min*: *prime-above n* ≥ *2*
using *prime-above-prime*
by (*simp add*: *prime-ge-2-nat*)

lemma *prime-above-lower-bound*: *prime-above n* ≥ *n*
using *prime-above-range*
by *simp*

lemma *prime-above-upper-bound*: *prime-above n* ≤ *2*∗*n*+*2*
using *prime-above-range*
by *simp*

end

2 Frequency Moments

theory *Frequency-Moments*
imports
Frequency-Moments-Preliminary-Results
Universal-Hash-Families.Field
Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities
begin

This section contains a definition of the frequency moments of a stream and a few general results about frequency moments..

definition *F* **where**
F k xs = (∑ *x* ∈ *set xs*. (*rat-of-nat* (*count-list xs x*)^*k*))

lemma *F-ge-0*: *F k as* ≥ *0*
unfolding *F-def* **by** (*rule sum-nonneg*, *simp*)

lemma *F-gr-0*:
assumes *as* ≠ []
shows *F k as* > *0*
proof −
have *rat-of-nat 1* ≤ *rat-of-nat* (*card* (*set as*))
using *assms card-0-eq*[**where** *A*=*set as*]
by (*intro of-nat-mono*)
(*metis List.finite-set One-nat-def Suc-leI neq0-conv set-empty*)
also have ... = (∑*x*∈*set as*. *1*) **by** *simp*
also have ... ≤ (∑*x*∈*set as*. *rat-of-nat* (*count-list as x*) ^ *k*)
by (*intro sum-mono one-le-power*)
(*metis count-list-gr-1 of-nat-1 of-nat-le-iff*)
also have ... ≤ *F k as*

by (*simp add*:*F-def*)
finally show *?thesis* **by** *simp*
qed

definition P_e :: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *bool list option* **where**
P_e *p n f* = (*if p* > *1* ∧ *f* ∈ *bounded-degree-polynomials* (*Field.mod-ring p*) *n then*
([*0*..<*n*] →_e Nb_e *p*) (λ*i* ∈ {..<*n*}. *ring.coeff* (*Field.mod-ring p*) *f i*) *else None*)

lemma *poly-encoding*:
is-encoding (P_e *p n*)
proof (*cases p* > *1*)
case *True*
interpret *cring Field.mod-ring p*
using *mod-ring-is-cring True* **by** *blast*
have *a*:*inj-on* (λ*x*. (λ*i* ∈ {..<*n*}. (*coeff x i*))) (*bounded-degree-polynomials* (*mod-ring p*) *n*)
proof (*rule inj-onI*)
fix *x y*
assume *b*:*x* ∈ *bounded-degree-polynomials* (*mod-ring p*) *n*
assume *c*:*y* ∈ *bounded-degree-polynomials* (*mod-ring p*) *n*
assume *d*:*restrict* (*coeff x*) {..<*n*} = *restrict* (*coeff y*) {..<*n*}
have *coeff x i* = *coeff y i* **for** *i*
proof (*cases i* < *n*)
case *True*
then show *?thesis* **by** (*metis lessThan-iff restrict-apply d*)
next
case *False*
hence *e*: *i* ≥ *n* **by** *linarith*
have *coeff x i* = $\mathbf{0}_{mod\text{-}ring\ p}$
using *b e* **by** (*subst coeff-length*, *auto simp*:*bounded-degree-polynomials-length*)
also have ... = *coeff y i*
using *c e* **by** (*subst coeff-length*, *auto simp*:*bounded-degree-polynomials-length*)
finally show *?thesis* **by** *simp*
qed
then show *x* = *y*
using *b c univ-poly-carrier*
by (*subst coeff-iff-polynomial-cond*) (*auto simp*:*bounded-degree-polynomials-length*)

qed

have *is-encoding* (λ*f*. P_e *p n f*)
unfolding P_e-*def* **using** *a True*
by (*intro encoding-compose*[**where** *f*=([*0*..<*n*] →$_e$ *Nb*$_e$ *p*)] *fun-encoding bounded-nat-encoding*)

auto
thus *?thesis* **by** *simp*
next
case *False*
hence *is-encoding* (λ*f*. P_e *p n f*)

```
    unfolding P_e-def using encoding-triv by simp
  then show ?thesis by simp
qed

lemma bounded-degree-polynomial-bit-count:
  assumes p > 1
  assumes x ∈ bounded-degree-polynomials (Field.mod-ring p) n
  shows bit-count (P_e p n x) ≤ ereal (real n ∗ (log 2 p + 1))
proof −
  interpret cring Field.mod-ring p
    using mod-ring-is-cring assms by blast

  have a: x ∈ carrier (poly-ring (mod-ring p))
    using assms(2) by (simp add:bounded-degree-polynomials-def)

  have real-of-int ⌊log 2 (p−1)⌋+1 ≤ log 2 (p−1) + 1
    using floor-eq-iff by (intro add-mono, auto)
  also have ... ≤ log 2 p + 1
    using assms by (intro add-mono, auto)
  finally have b: ⌊log 2 (p−1)⌋+1 ≤ log 2 p + 1
    by simp

  have bit-count (P_e p n x) = (∑ k ← [0..<n]. bit-count (Nb_e p (coeff x k)))
    using assms restrict-extensional
   by (auto intro!:arg-cong[where f=sum-list] simp add:P_e-def fun-bit-count lessThan-atLeast0)
  also have ... = (∑ k ← [0..<n]. ereal (floorlog 2 (p−1)))
    using coeff-in-carrier[OF a] mod-ring-carr
    by (subst bounded-nat-bit-count-2, auto)
  also have ... = n ∗ ereal (floorlog 2 (p−1))
    by (simp add: sum-list-triv)
  also have ... = n ∗ real-of-int (⌊log 2 (p−1)⌋+1)
    using assms(1) by (simp add:floorlog-def)
  also have ... ≤ ereal (real n ∗ (log 2 p + 1))
    by (subst ereal-less-eq, intro mult-left-mono b, auto)
  finally show ?thesis by simp
qed

end
```

3 Ranks, k smallest element and elements

```
theory K-Smallest
  imports
    Frequency-Moments-Preliminary-Results
    Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities
begin
```

This section contains definitions and results for the selection of the k smallest elements, the k-th smallest element, rank of an element in an ordered set.

definition *rank-of* :: $'a$ :: *linorder* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *nat* **where** *rank-of* x S = *card* $\{y \in S.\ y < x\}$

The function *rank-of* returns the rank of an element within a set.

lemma *rank-mono*:
  assumes *finite* S
  shows $x \le y \Longrightarrow$ *rank-of* x $S \le$ *rank-of* y S
  unfolding *rank-of-def* **using** *assms* **by** (*intro card-mono*, *auto*)

lemma *rank-mono-2*:
  assumes *finite* S
  shows $S' \subseteq S \Longrightarrow$ *rank-of* x $S' \le$ *rank-of* x S
  unfolding *rank-of-def* **using** *assms* **by** (*intro card-mono*, *auto*)

lemma *rank-mono-commute*:
  assumes *finite* S
  assumes $S \subseteq T$
  assumes *strict-mono-on* T f
  assumes $x \in T$
  shows *rank-of* x S = *rank-of* $(f\ x)$ $(f$ ‘ $S)$
proof –
  have *a*: *inj-on* f T
    by (*metis assms*(*3*) *strict-mono-on-imp-inj-on*)

  have *rank-of* $(f\ x)$ $(f$ ‘ $S)$ = *card* $(f$ ‘ $\{y \in S.\ f\ y < f\ x\})$
    unfolding *rank-of-def* **by** (*intro arg-cong*[**where** *f*=*card*], *auto*)
  also have ... = *card* $(f$ ‘ $\{y \in S.\ y < x\})$
    using *assms* **by** (*intro arg-cong*[**where** *f*=*card*] *arg-cong*[**where** *f*=(‘) *f*])
    (*meson in-mono linorder-not-le strict-mono-onD strict-mono-on-leD set-eq-iff*)
  also have ... = *card* $\{y \in S.\ y < x\}$
    using *assms* **by** (*intro card-image inj-on-subset*[*OF a*], *blast*)
  also have ... = *rank-of* x S
    by (*simp add*:*rank-of-def*)
  finally show *?thesis*
    by *simp*
qed

definition *least* **where** *least* k S = $\{y \in S.\ $ *rank-of* y $S < k\}$

The function *K-Smallest.least* returns the k smallest elements of a finite set.

lemma *rank-strict-mono*:
  assumes *finite* S
  shows *strict-mono-on* S $(\lambda x.$ *rank-of* x $S)$
proof –
  have $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow x < y \Longrightarrow$ *rank-of* x $S <$ *rank-of* y S
    unfolding *rank-of-def* **using** *assms*
    by (*intro psubset-card-mono*, *auto*)

  thus *?thesis*

```
    by (simp add:rank-of-def strict-mono-on-def)
qed

lemma rank-of-image:
  assumes finite S
  shows (λx. rank-of x S) ` S = {0..<card S}
proof (rule card-seteq)
  show finite {0..<card S} by simp

  have ⋀x. x ∈ S ⟹ card {y ∈ S. y < x} < card S
    by (rule psubset-card-mono, metis assms, blast)
  thus (λx. rank-of x S) ` S ⊆ {0..<card S}
    by (intro image-subsetI, simp add:rank-of-def)

  have inj-on (λx. rank-of x S) S
    by (metis strict-mono-on-imp-inj-on rank-strict-mono assms)
  thus card {0..<card S} ≤ card ((λx. rank-of x S) ` S)
    by (simp add:card-image)
qed

lemma card-least:
  assumes finite S
  shows card (least k S) = min k (card S)
proof (cases card S < k)
  case True
  have ⋀t. rank-of t S ≤ card S
    unfolding rank-of-def using assms
    by (intro card-mono, auto)
  hence ⋀t. rank-of t S < k
    by (metis True not-less-iff-gr-or-eq order-less-le-trans)
  hence least k S = S
    by (simp add:least-def)
  then show ?thesis using True by simp
next
  case False
  hence a:card S ≥ k using leI by blast
  hence card ((λx. rank-of x S) -` {0..<k} ∩ S) = card {0..<k}
    using assms
    by (intro card-vimage-inj-on strict-mono-on-imp-inj-on rank-strict-mono)
     (simp-all add: rank-of-image)
  hence card (least k S) = k
    by (simp add: Collect-conj-eq Int-commute least-def vimage-def)
  then show ?thesis using a by linarith
qed

lemma least-subset: least k S ⊆ S
  by (simp add:least-def)

lemma least-mono-commute:
```

assumes *finite S*
assumes *strict-mono-on S f*
shows *f ‘ least k S = least k (f ‘ S)*
proof −
have *a:inj-on f S*
using *strict-mono-on-imp-inj-on[OF assms(2)]* **by** *simp*

have *card (least k (f ‘ S)) = min k (card (f ‘ S))*
by *(subst card-least, auto simp add:assms)*
also have ... = *min k (card S)*
by *(subst card-image, metis a, auto)*
also have ... = *card (least k S)*
by *(subst card-least, auto simp add:assms)*
also have ... = *card (f ‘ least k S)*
by *(subst card-image[OF inj-on-subset[OF a]], simp-all add:least-def)*
finally have *b: card (least k (f ‘ S)) ≤ card (f ‘ least k S)* **by** *simp*

have *c: f ‘ least k S ⊆least k (f ‘ S)*
using *assms* **by** *(intro image-subsetI)*
(simp add:least-def rank-mono-commute[symmetric, **where** *T=S])*

show *?thesis*
using *b c assms* **by** *(intro card-seteq, simp-all add:least-def)*
qed

lemma *least-eq-iff*:
assumes *finite B*
assumes *A ⊆ B*
assumes $\bigwedge x.\ x \in B \Longrightarrow$ *rank-of x B* $< k \Longrightarrow x \in A$
shows *least k A = least k B*
proof −
have *least k B ⊆ least k A*
using *assms rank-mono-2[OF assms(1,2)] order-le-less-trans*
by *(simp add:least-def, blast)*
moreover have *card (least k B) ≥ card (least k A)*
using *assms finite-subset[OF assms(2,1)] card-mono[OF assms(1,2)]*
by *(simp add: card-least min-le-iff-disj)*
moreover have *finite (least k A)*
using *finite-subset least-subset assms(1,2)* **by** *metis*
ultimately show *?thesis*
by *(intro card-seteq[symmetric], simp-all)*
qed

lemma *least-insert*:
assumes *finite S*
shows *least k (insert x (least k S)) = least k (insert x S)* (**is** *?lhs = ?rhs*)
proof *(rule least-eq-iff)*
show *finite (insert x S)*
using *assms(1)* **by** *simp*

```
  show insert x (least k S) ⊆ insert x S
    using least-subset by blast
  show y ∈ insert x (least k S) if a: y ∈ insert x S and b: rank-of y (insert x S)
< k for y
  proof −
    have rank-of y S ≤ rank-of y (insert x S)
      using assms by (intro rank-mono-2, auto)
    also have ... < k using b by simp
    finally have rank-of y S < k by simp
    hence y = x ∨ (y ∈ S ∧ rank-of y S < k)
      using a by simp
    thus ?thesis by (simp add:least-def)
  qed
qed


definition count-le where count-le x M = size {#y ∈# M. y ≤ x#}
definition count-less where count-less x M = size {#y ∈# M. y < x#}

definition nth-mset :: nat ⇒ ('a :: linorder) multiset ⇒ 'a where
  nth-mset k M = sorted-list-of-multiset M ! k

lemma nth-mset-bound-left:
  assumes k < size M
  assumes count-less x M ≤ k
  shows x ≤ nth-mset k M
proof (rule ccontr)
  define xs where xs = sorted-list-of-multiset M
  have s-xs: sorted xs by (simp add:xs-def sorted-sorted-list-of-multiset)
  have l-xs: k < length xs
    using  assms(1) by (simp add:xs-def size-mset[symmetric])
  have M-xs: M = mset xs by (simp add:xs-def)
  hence a:⋀i. i ≤ k ⟹ xs ! i ≤ xs ! k
    using s-xs l-xs sorted-iff-nth-mono by blast

  assume ¬(x ≤ nth-mset k M)
  hence x > nth-mset k M by simp
  hence b:x > xs ! k by (simp add:nth-mset-def xs-def[symmetric])

  have k < card {0..k} by simp
  also have ... ≤ card {i. i < length xs ∧ xs ! i < x}
    using a b l-xs order-le-less-trans
    by (intro card-mono subsetI) auto
  also have ... = length (filter (λy. y < x) xs)
    by (subst length-filter-conv-card, simp)
  also have ... = size (mset (filter (λy. y < x) xs))
    by (subst size-mset, simp)
  also have ... = count-less x M
    by (simp add:count-less-def M-xs)
```

also have ... $\leq$ *k*
using *assms* **by** *simp*
finally show *False* **by** *simp*
qed

lemma *nth-mset-bound-left-excl*:
assumes *k* < *size M*
assumes *count-le x M* $\leq$ *k*
shows *x* < *nth-mset k M*
proof (*rule ccontr*)
define *xs* **where** *xs* = *sorted-list-of-multiset M*
have *s-xs*: *sorted xs* **by** (*simp add*:*xs-def sorted-sorted-list-of-multiset*)
have *l-xs*: *k* < *length xs*
using *assms*(*1*) **by** (*simp add*:*xs-def size-mset*[*symmetric*])
have *M-xs*: *M* = *mset xs* **by** (*simp add*:*xs-def*)
hence *a*:$\bigwedge$*i*. *i* $\leq$ *k* $\Longrightarrow$ *xs* ! *i* $\leq$ *xs* ! *k*
using *s-xs l-xs sorted-iff-nth-mono* **by** *blast*

assume ¬(*x* < *nth-mset k M*)
hence *x* $\geq$ *nth-mset k M* **by** *simp*
hence *b*:*x* $\geq$ *xs* ! *k* **by** (*simp add*:*nth-mset-def xs-def*[*symmetric*])

have *k*+*1* $\leq$ *card* {*0*..*k*} **by** *simp*
also have ... $\leq$ *card* {*i*. *i* < *length xs* ∧ *xs* ! *i* $\leq$ *xs* ! *k*}
using *a b l-xs order-le-less-trans*
by (*intro card-mono subsetI*, *auto*)
also have ... $\leq$ *card* {*i*. *i* < *length xs* ∧ *xs* ! *i* $\leq$ *x*}
using *b* **by** (*intro card-mono subsetI*, *auto*)
also have ... = *length* (*filter* (λ*y*. *y* $\leq$ *x*) *xs*)
by (*subst length-filter-conv-card*, *simp*)
also have ... = *size* (*mset* (*filter* (λ*y*. *y* $\leq$ *x*) *xs*))
by (*subst size-mset*, *simp*)
also have ... = *count-le x M*
by (*simp add*:*count-le-def M-xs*)
also have ... $\leq$ *k*
using *assms* **by** *simp*
finally show *False* **by** *simp*
qed

lemma *nth-mset-bound-right*:
assumes *k* < *size M*
assumes *count-le x M* > *k*
shows *nth-mset k M* $\leq$ *x*
proof (*rule ccontr*)
define *xs* **where** *xs* = *sorted-list-of-multiset M*
have *s-xs*: *sorted xs* **by** (*simp add*:*xs-def sorted-sorted-list-of-multiset*)
have *l-xs*: *k* < *length xs*
using *assms*(*1*) **by** (*simp add*:*xs-def size-mset*[*symmetric*])
have *M-xs*: *M* = *mset xs* **by** (*simp add*:*xs-def*)

assume ¬(*nth-mset k M* ≤ *x*)
hence *x* < *nth-mset k M* **by** *simp*
hence *x* < *xs* ! *k*
by (*simp add*:*nth-mset-def xs-def*[*symmetric*])
hence *a*:⋀*i*. *i* < *length xs* ∧ *xs* ! *i* ≤ *x* ⟹ *i* < *k*
using *s-xs l-xs sorted-iff-nth-mono leI* **by** *fastforce*
have *count-le x M* = *size* (*mset* (*filter* (λ*y*. *y* ≤ *x*) *xs*))
by (*simp add*:*count-le-def M-xs*)
also have ... = *length* (*filter* (λ*y*. *y* ≤ *x*) *xs*)
by (*subst size-mset*, *simp*)
also have ... = *card* {*i*. *i* < *length xs* ∧ *xs* ! *i* ≤ *x*}
by (*subst length-filter-conv-card*, *simp*)
also have ... ≤ *card* {*i*. *i* < *k*}
using *a* **by** (*intro card-mono subsetI*, *auto*)
also have ... = *k* **by** *simp*
finally have *count-le x M* ≤ *k* **by** *simp*
thus *False* **using** *assms* **by** *simp*
qed

lemma *nth-mset-commute-mono*:
assumes *mono f*
assumes *k* < *size M*
shows *f* (*nth-mset k M*) = *nth-mset k* (*image-mset f M*)
proof −
have *a*:*k* < *length* (*sorted-list-of-multiset M*)
by (*metis assms*(*2*) *mset-sorted-list-of-multiset size-mset*)
show *?thesis*
using *a* **by** (*simp add*:*nth-mset-def sorted-list-of-multiset-image-commute*[*OF assms*(*1*)])
qed

lemma *nth-mset-max*:
assumes *size A* > *k*
assumes ⋀*x*. *x* ≤ *nth-mset k A* ⟹ *count A x* ≤ *1*
shows *nth-mset k A* = *Max* (*least* (*k*+*1*) (*set-mset A*)) **and** *card* (*least* (*k*+*1*) (*set-mset A*)) = *k*+*1*
proof −
define *xs* **where** *xs* = *sorted-list-of-multiset A*
have *k-bound*: *k* < *length xs* **unfolding** *xs-def*
by (*metis size-mset mset-sorted-list-of-multiset assms*(*1*))

have *A-def*: *A* = *mset xs* **by** (*simp add*:*xs-def*)
have *s-xs*: *sorted xs* **by** (*simp add*:*xs-def sorted-sorted-list-of-multiset*)
have ⋀*x*. *x* ≤ *xs* ! *k* ⟹ *count A x* ≤ *Suc 0*
using *assms*(*2*) **by** (*simp add*:*xs-def*[*symmetric*] *nth-mset-def*)
hence *no-col*: ⋀*x*. *x* ≤ *xs* ! *k* ⟹ *count-list xs x* ≤ *1*
by (*simp add*:*A-def count-mset*)

```
  have inj-xs: inj-on (λk. xs ! k) {0..k}
    by (rule inj-onI, simp) (metis (full-types) count-list-ge-2-iff k-bound no-col
       le-neq-implies-less linorder-not-le order-le-less-trans s-xs sorted-iff-nth-mono)

  have ⋀y. y < length xs ⟹ rank-of (xs ! y) (set xs) < k+1 ⟹ y < k+1
  proof (rule ccontr)
    fix y
    assume b:y < length xs
    assume ¬y < k +1
    hence a:k + 1 ≤ y by simp

    have d:Suc k < length xs using a b by simp

    have k+1 = card ((!) xs ` {0..k})
      by (subst card-image[OF inj-xs], simp)
    also have ... ≤ rank-of (xs ! (k+1)) (set xs)
      unfolding rank-of-def using k-bound
       by (intro card-mono image-subsetI conjI, simp-all) (metis count-list-ge-2-iff
no-col not-le le-imp-less-Suc s-xs
          sorted-iff-nth-mono d order-less-le)
    also have ... ≤ rank-of (xs ! y) (set xs)
      unfolding rank-of-def
      by (intro card-mono subsetI, simp-all)
       (metis Suc-eq-plus1 a b s-xs order-less-le-trans sorted-iff-nth-mono)
    also assume ... < k+1
    finally show False by force
  qed

  moreover have rank-of (xs ! y) (set xs) < k+1 if a:y < k + 1 for y
  proof −
    have rank-of (xs ! y) (set xs) ≤ card ((λk. xs ! k) ` {k. k < length xs ∧ xs ! k
< xs ! y})
      unfolding rank-of-def
      by (intro card-mono subsetI, simp)
       (metis (no-types, lifting) imageI in-set-conv-nth mem-Collect-eq)
    also have ... ≤ card {k. k < length xs ∧ xs ! k < xs ! y}
      by (rule card-image-le, simp)
    also have ... ≤ card {k. k < y}
      by (intro card-mono subsetI, simp-all add:not-less)
       (metis sorted-iff-nth-mono s-xs linorder-not-less)
    also have ... = y by simp
    also have ... < k + 1 using a by simp
    finally show rank-of (xs ! y) (set xs) < k+1 by simp
  qed

  ultimately have rank-conv: ⋀y. y < length xs ⟹ rank-of (xs ! y) (set xs) <
k+1 ⟷ y < k+1
    by blast
```

```
  have y ≤ xs ! k  if a:y ∈ least (k+1) (set xs)  for y
  proof −
    have y ∈ set xs using a least-subset by blast
     then obtain i where i-bound: i < length xs and y-def: y = xs ! i using
in-set-conv-nth by metis
    hence rank-of (xs ! i) (set xs) < k+1
      using a y-def i-bound by (simp add: least-def)
    hence i < k+1
      using rank-conv i-bound by blast
    hence i ≤ k by linarith
    hence xs ! i ≤ xs ! k
      using s-xs i-bound k-bound sorted-nth-mono by blast
    thus y ≤ xs ! k using y-def by simp
  qed

  moreover have xs ! k ∈ least (k+1) (set xs)
    using k-bound rank-conv by (simp add:least-def)

  ultimately have Max (least (k+1) (set xs)) = xs ! k
    by (intro Max-eqI finite-subset[OF least-subset], auto)

  hence nth-mset k A = Max (K-Smallest.least (Suc k) (set xs))
    by (simp add:nth-mset-def xs-def[symmetric])
  also have ... = Max (least (k+1) (set-mset A))
    by (simp add:A-def)
  finally show nth-mset k A = Max (least (k+1) (set-mset A))  by simp

  have k + 1 = card ((λi. xs ! i) ` {0..k})
    by (subst card-image[OF inj-xs], simp)
  also have ... ≤ card (least (k+1) (set xs))
    using rank-conv k-bound
   by (intro card-mono image-subsetI finite-subset[OF least-subset], simp-all add:least-def)
  finally have card (least (k+1) (set xs)) ≥ k+1 by simp
  moreover have card (least (k+1) (set xs)) ≤ k+1
    by (subst card-least, simp, simp)
  ultimately have card (least (k+1) (set xs)) = k+1 by simp
  thus card (least (k+1) (set-mset A)) = k+1  by (simp add:A-def)
qed

end
```

4 Landau Symbols

```
theory Landau-Ext
  imports
    HOL−Library.Landau-Symbols
    HOL.Topological-Spaces
begin
```

This section contains results about Landau Symbols in addition to "HOL-Library.Landau".

```
lemma landau-sum:
  assumes eventually (λx. g1 x ≥ (0::real)) F
  assumes eventually (λx. g2 x ≥ 0) F
  assumes f1 ∈ O[F](g1)
  assumes f2 ∈ O[F](g2)
  shows (λx. f1 x + f2 x) ∈ O[F](λx. g1 x + g2 x)
proof −
  obtain c1 where a1: c1 > 0 and b1: eventually (λx. abs (f1 x) ≤ c1 ∗ abs (g1
x)) F
    using assms(3) by (simp add:bigo-def, blast)
  obtain c2 where a2: c2 > 0 and b2: eventually (λx. abs (f2 x) ≤ c2 ∗ abs (g2
x)) F
    using assms(4) by (simp add:bigo-def, blast)
  have eventually (λx. abs (f1 x + f2 x) ≤ (max c1 c2) ∗ abs (g1 x + g2 x)) F
   proof (rule eventually-mono[OF eventually-conj[OF b1 eventually-conj[OF b2
eventually-conj[OF assms(1,2)]]]])
    fix x
    assume a: |f1 x| ≤ c1 ∗ |g1 x| ∧ |f2 x| ≤ c2 ∗ |g2 x| ∧ 0 ≤ g1 x ∧ 0 ≤ g2 x
    have |f1 x + f2 x| ≤ |f1 x | + |f2 x| using abs-triangle-ineq by blast
    also have ... ≤ c1 ∗  |g1 x| + c2 ∗ |g2 x| using a add-mono by blast
    also have ... ≤ max c1 c2 ∗ |g1 x| + max c1 c2 ∗ |g2 x|
      by (intro add-mono mult-right-mono) auto
    also have ... = max c1 c2 ∗ (|g1 x| + |g2 x|)
      by (simp add:algebra-simps)
    also have ... ≤ max c1 c2 ∗ (|g1 x + g2 x|)
      using a a1 a2 by (intro mult-left-mono) auto
    finally show |f1 x + f2 x| ≤ max c1 c2 ∗ |g1 x + g2 x|
      by (simp add:algebra-simps)
  qed
  hence  0 < max c1 c2 ∧ (∀F x in F. |f1 x + f2 x| ≤ max c1 c2 ∗ |g1 x + g2 x|)
    using a1 a2 by linarith
  thus ?thesis
    by (simp add: bigo-def, blast)
qed

lemma landau-sum-1:
  assumes eventually (λx. g1 x ≥ (0::real)) F
  assumes eventually (λx. g2 x ≥ 0) F
  assumes f ∈ O[F](g1)
  shows f ∈ O[F](λx. g1 x + g2 x)
proof −
  have f = (λx. f x + 0) by simp
  also have ... ∈ O[F](λx. g1 x + g2 x)
    using assms zero-in-bigo by (intro landau-sum)
  finally show ?thesis by simp
qed
```

lemma *landau-sum-2*:
  assumes *eventually* (λ*x*. *g1 x* ≥ (*0*::*real*)) *F*
  assumes *eventually* (λ*x*. *g2 x* ≥ *0*) *F*
  assumes *f* ∈ *O*[*F*](*g2*)
  shows *f* ∈ *O*[*F*](λ*x*. *g1 x* + *g2 x*)
proof −
  have *f* = (λ*x*. *0* + *f x*) **by** *simp*
  also have ... ∈ *O*[*F*](λ*x*. *g1 x* + *g2 x*)
    using *assms zero-in-bigo* **by** (*intro landau-sum*)
  finally show *?thesis* **by** *simp*
qed

lemma *landau-ln-3*:
  assumes *eventually* (λ*x*. (*1*::*real*) ≤ *f x*) *F*
  assumes *f* ∈ *O*[*F*](*g*)
  shows (λ*x*. *ln* (*f x*)) ∈ *O*[*F*](*g*)
proof −
  have *1* ≤ *x* ⟹ |*ln x*| ≤ |*x*| **for** *x* :: *real*
    using *ln-bound* **by** *auto*
  hence (λ*x*. *ln* (*f x*)) ∈ *O*[*F*](*f*)
    by (*intro landau-o.big-mono eventually-mono*[*OF assms*(*1*)]) *simp*
  thus *?thesis*
    using *assms*(*2*) *landau-o.big-trans* **by** *blast*
qed

lemma *landau-ln-2*:
  assumes *a* > (*1*::*real*)
  assumes *eventually* (λ*x*. *1* ≤ *f x*) *F*
  assumes *eventually* (λ*x*. *a* ≤ *g x*) *F*
  assumes *f* ∈ *O*[*F*](*g*)
  shows (λ*x*. *ln* (*f x*)) ∈ *O*[*F*](λ*x*. *ln* (*g x*))
proof −
  obtain *c* **where** *a*: *c* > *0* **and** *b*: *eventually* (λ*x*. *abs* (*f x*) ≤ *c* ∗ *abs* (*g x*)) *F*
    using *assms*(*4*) **by** (*simp add*:*bigo-def*, *blast*)
  define *d* **where** *d* = *1* + (*max 0* (*ln c*)) / *ln a*
  have *d*:*eventually* (λ*x*. *abs* (*ln* (*f x*)) ≤ *d* ∗ *abs* (*ln* (*g x*))) *F*
  proof (*rule eventually-mono*[*OF eventually-conj*[*OF b eventually-conj*[*OF assms*(*3*,*2*)]]])
    fix *x*
    assume *c*:|*f x*| ≤ *c* ∗ |*g x*| ∧ *a* ≤ *g x* ∧ *1* ≤ *f x*
    have *abs* (*ln* (*f x*)) = *ln* (*f x*)
      by (*subst abs-of-nonneg*, *rule ln-ge-zero*, *metis c*, *simp*)
    also have ... ≤ *ln* (*c* ∗ *abs* (*g x*))
      using *c assms*(*1*) *mult-pos-pos*[*OF a*] **by** *auto*
    also have ... ≤ *ln c* + *ln* (*abs* (*g x*))
      using *c assms*(*1*)
      by (*simp add*: *ln-mult*[*OF a*])
    also have ... ≤ (*d*−*1*)∗*ln a* + *ln* (*g x*)
      using *assms*(*1*) *c*
      by (*intro add-mono iffD2*[*OF ln-le-cancel-iff*], *simp-all add*:*d-def*)

  also have ... $\leq$ (*d*−*1*)∗ *ln* (*g x*) + *ln* (*g x*)
   using *assms*(*1*) *c*
  by (*intro add-mono mult-left-mono iffD2*[*OF ln-le-cancel-iff*], *simp-all add*:*d-def*)
  also have ... = *d* ∗ *ln* (*g x*) **by** (*simp add*:*algebra-simps*)
  also have ... = *d* ∗ *abs* (*ln* (*g x*))
   using *c assms*(*1*) **by** *auto*
  finally show *abs* (*ln* (*f x*)) $\leq$ *d* ∗ *abs* (*ln* (*g x*)) **by** *simp*
 qed
 hence $\forall_F$ *x in F*. |*ln* (*f x*)| $\leq$ *d* ∗ |*ln* (*g x*)|
  by *simp*
 moreover have *0* < *d*
  unfolding *d-def* **using** *assms*(*1*)
  by (*intro add-pos-nonneg divide-nonneg-pos*, *auto*)
 ultimately show *?thesis*
  by (*auto simp*:*bigo-def*)
qed

lemma *landau-real-nat*:
 fixes *f* :: *'a* ⇒ *int*
 assumes (λ*x*. *of-int* (*f x*)) ∈ *O*[*F*](*g*)
 shows (λ*x*. *real* (*nat* (*f x*))) ∈ *O*[*F*](*g*)
proof −
 obtain *c* **where** *a*: *c* > *0* **and** *b*: *eventually* (λ*x*. *abs* (*of-int* (*f x*)) $\leq$ *c* ∗ *abs* (*g*
x)) *F*
  using *assms*(*1*) **by** (*simp add*:*bigo-def*, *blast*)
 have $\forall_F$ *x in F*. *real* (*nat* (*f x*)) $\leq$ *c* ∗ |*g x*|
  by (*rule eventually-mono*[*OF b*], *simp*)
 thus *?thesis* **using** *a*
  by (*auto simp*:*bigo-def*)
qed

lemma *landau-ceil*:
 assumes (λ-. *1*) ∈ *O*[*F'*](*g*)
 assumes *f* ∈ *O*[*F'*](*g*)
 shows (λ*x*. *real-of-int* ⌈*f x*⌉) ∈ *O*[*F'*](*g*)
proof −
 have (λ*x*. *real-of-int* ⌈*f x*⌉) ∈ *O*[*F'*](λ*x*. *1* + *abs* (*f x*))
  by (*intro landau-o.big-mono always-eventually allI*, *simp*, *linarith*)
 also have (λ*x*. *1* + *abs*(*f x*)) ∈ *O*[*F'*](*g*)
  using *assms*(*2*) **by** (*intro sum-in-bigo assms*(*1*), *auto*)
 finally show *?thesis* **by** *simp*
qed

lemma *landau-rat-ceil*:
 assumes (λ-. *1*) ∈ *O*[*F'*](*g*)
 assumes (λ*x*. *real-of-rat* (*f x*)) ∈ *O*[*F'*](*g*)
 shows (λ*x*. *real-of-int* ⌈*f x*⌉) ∈ *O*[*F'*](*g*)
proof −
 have *a*:|*real-of-int* ⌈*x*⌉| $\leq$ *1* + *real-of-rat* |*x*| **for** *x* :: *rat*

```
  proof (cases x ≥ 0)
    case True
    then show ?thesis
      by (simp, metis add.commute of-int-ceiling-le-add-one of-rat-ceiling)
  next
    case False
    have real-of-rat x − 1 ≤ real-of-rat x
      by simp
    also have ... ≤ real-of-int ⌈x⌉
      by (metis ceiling-correct of-rat-ceiling)
    finally have real-of-rat (x)−1 ≤ real-of-int ⌈x⌉ by simp

    hence − real-of-int ⌈x⌉ ≤ 1 + real-of-rat (− x)
      by (simp add: of-rat-minus)
    then show ?thesis using False by simp
  qed
  have (λx. real-of-int ⌈f x⌉) ∈ O[F'](λx. 1 + abs (real-of-rat (f x)))
    using a
    by (intro landau-o.big-mono always-eventually allI, simp)
  also have (λx. 1 + abs (real-of-rat (f x))) ∈ O[F'](g)
    using assms
    by (intro sum-in-bigo assms(1), subst landau-o.big.abs-in-iff, simp)
  finally show ?thesis by simp
qed

lemma landau-nat-ceil:
  assumes (λ-. 1) ∈ O[F'](g)
  assumes f ∈ O[F'](g)
  shows (λx. real (nat ⌈f x⌉)) ∈ O[F'](g)
  using assms
  by (intro landau-real-nat landau-ceil, auto)

lemma eventually-prod1':
  assumes B ≠ bot
  assumes (∀F x in A. P x)
  shows (∀F x in A ×F B. P (fst x))
proof −
  have (∀F x in A ×F B. P (fst x)) = (∀F (x,y) in A ×F B. P x)
    by (simp add:case-prod-beta')
  also have ... = (∀F x in A. P x)
    by (subst eventually-prod1[OF assms(1)], simp)
  finally show ?thesis using assms(2) by simp
qed

lemma eventually-prod2':
  assumes A ≠ bot
  assumes (∀F x in B. P x)
  shows (∀F x in A ×F B. P (snd x))
proof −
```

```
  have (∀F x in A ×F B. P (snd x)) = (∀F (x,y) in A ×F B. P y)
    by (simp add:case-prod-beta′)
  also have ... = (∀F x in B. P x)
    by (subst eventually-prod2[OF assms(1)], simp)
  finally show ?thesis using assms(2) by simp
qed

lemma sequentially-inf: ∀F x in sequentially. n ≤ real x
  by (meson eventually-at-top-linorder nat-ceiling-le-eq)

instantiation rat :: linorder-topology
begin

definition open-rat :: rat set ⇒ bool
  where open-rat = generate-topology (range (λa. {..< a}) ∪ range (λa. {a <..}))

instance
  by standard (rule open-rat-def)
end

lemma inv-at-right-0-inf:
  ∀F x in at-right 0. c ≤ 1 / real-of-rat x
proof –
  have a: c ≤ 1 / real-of-rat x if b: x ∈ {0<..<1 / rat-of-int (max ⌈c⌉ 1)} for x
  proof –
    have c ∗ real-of-rat x ≤ real-of-int (max ⌈c⌉ 1) ∗ real-of-rat x
      using b by (intro mult-right-mono, linarith, auto)
     also have ... < real-of-int (max ⌈c⌉ 1)  ∗ real-of-rat (1/rat-of-int  (max ⌈c⌉
1) )
      using b by (intro mult-strict-left-mono iffD2[OF of-rat-less], auto)
    also have ... ≤ 1
      by (simp add:of-rat-divide)
    finally have c ∗ real-of-rat x ≤ 1 by simp
    moreover have 0 < real-of-rat x
      using b by simp
    ultimately show ?thesis by (subst pos-le-divide-eq, auto)
  qed

  show ?thesis
    using a
    by (intro eventually-at-rightI[where b=1/rat-of-int (max ⌈c⌉ 1)], simp-all)
qed

end
```

5 Probability Spaces

Some additional results about probability spaces in addition to "HOL-Probability".

theory *Probability-Ext*
imports
HOL−Probability.Stream-Space
Universal-Hash-Families.Carter-Wegman-Hash-Family
Frequency-Moments-Preliminary-Results
begin

Random variables that depend on disjoint sets of the components of a product space are independent.

lemma *make-ext*:
assumes $\bigwedge x.\ P\ x = P\ (restrict\ x\ I)$
shows $(\forall x \in Pi\ I\ A.\ P\ x) = (\forall x \in PiE\ I\ A.\ P\ x)$
using *assms* **by** (*simp add*:*PiE-def Pi-def set-eq-iff*, *force*)

lemma *PiE-reindex*:
assumes *inj-on f I*
shows $PiE\ I\ (A \circ f) = (\lambda a.\ restrict\ (a \circ f)\ I)$ ' $PiE\ (f$ ' $I)\ A$ (**is** *?lhs* = *?g* ' *?rhs*)
proof −
have *?lhs* ⊆ *?g*' *?rhs*
proof (*rule subsetI*)
fix *x*
assume $a{:}x \in Pi_E\ I\ (A \circ f)$
define *y* **where** *y-def*: $y = (\lambda k.$ *if* $k \in f$ ' I *then* x (*the-inv-into I f k*) *else undefined*)
have $b{:}y \in PiE\ (f$ ' $I)\ A$
using *a assms the-inv-into-f-eq*[*OF assms*]
by (*simp add*: *y-def PiE-iff extensional-def*)
have $c{:}\ x = (\lambda a.\ restrict\ (a \circ f)\ I)\ y$
using *a assms the-inv-into-f-eq extensional-arb*
by (*intro ext*, *simp add*:*y-def PiE-iff*, *fastforce*)
show $x \in$ *?g* ' *?rhs* **using** *b c* **by** *blast*
qed
moreover have *?g* ' *?rhs* ⊆ *?lhs*
by (*rule image-subsetI*, *simp add*:*Pi-def PiE-def*)
ultimately show *?thesis* **by** *blast*
qed

context *prob-space*
begin

lemma *indep-sets-reindex*:
assumes *inj-on f I*
shows *indep-sets* $A\ (f$ ' $I)$ = *indep-sets* $(\lambda i.\ A\ (f\ i))\ I$
proof −
have $a{:}\bigwedge J\ g.\ J \subseteq I \Longrightarrow (\prod j \in f\ `\ J.\ g\ j) = (\prod j \in J.\ g\ (f\ j))$
by (*metis assms prod.reindex-cong subset-inj-on*)

have $J \subseteq I \Longrightarrow (\Pi_E\ i \in J.\ A\ (f\ i)) = (\lambda a.\ restrict\ (a \circ f)\ J)$ ' $PiE\ (f$ ' $J)\ A$

for *J*
  using *assms inj-on-subset*
  by (*subst PiE-reindex*[*symmetric*]) *auto*

hence *b*:$\bigwedge P\ J.\ J \subseteq I \Longrightarrow (\bigwedge x.\ P\ x = P\ (restrict\ x\ J)) \Longrightarrow (\forall A' \in \Pi_E\ i \in J.\ A\ (f\ i).\ P\ A') = (\forall A' \in PiE\ (f\ `\ J)\ A.\ P\ (A' \circ f))$
  by *simp*

have *c*:$\bigwedge J.\ J \subseteq I \Longrightarrow finite\ (f\ `\ J) = finite\ J$
  by (*meson assms finite-image-iff inj-on-subset*)

show *?thesis*
  by (*simp add*:*indep-sets-def all-subset-image a c*)
  (*simp add*:*make-ext b cong*:*restrict-cong*)+
qed

lemma *indep-vars-cong-AE*:
assumes $AE\ x\ in\ M.\ (\forall i \in I.\ X'\ i\ x = Y'\ i\ x)$
assumes $indep\text{-}vars\ M'\ X'\ I$
assumes $\bigwedge i.\ i \in I \Longrightarrow random\text{-}variable\ (M'\ i)\ (Y'\ i)$
shows $indep\text{-}vars\ M'\ Y'\ I$
proof (*cases* $I \neq \{\}$)
case *True*

have *a*: $AE\ x\ in\ M.\ (\lambda i{\in}I.\ Y'\ i\ x) = (\lambda i{\in}I.\ X'\ i\ x)$
  by (*rule AE-mp*[*OF assms*(*1*)], *rule AE-I2*, *simp cong*:*restrict-cong*)
have *b*: $\bigwedge i.\ i \in I \Longrightarrow random\text{-}variable\ (M'\ i)\ (X'\ i)$
  using *assms*(*2*) **by** (*simp add*:*indep-vars-def2*)
have *c*: $\bigwedge x.\ x \in I \Longrightarrow AE\ xa\ in\ M.\ X'\ x\ xa = Y'\ x\ xa$
  by (*rule AE-mp*[*OF assms*(*1*)], *rule AE-I2*, *simp*)

have $distr\ M\ (Pi_M\ I\ M')\ (\lambda x.\ \lambda i{\in}I.\ Y'\ i\ x) = distr\ M\ (Pi_M\ I\ M')\ (\lambda x.\ \lambda i{\in}I.\ X'\ i\ x)$
  by (*intro distr-cong-AE measurable-restrict a b assms*(*3*)) *auto*
also have ... $= Pi_M\ I\ (\lambda i.\ distr\ M\ (M'\ i)\ (X'\ i))$
  using *assms True b* **by** (*subst indep-vars-iff-distr-eq-PiM'*[*symmetric*]) *auto*
also have ... $= Pi_M\ I\ (\lambda i.\ distr\ M\ (M'\ i)\ (Y'\ i))$
  by (*intro PiM-cong distr-cong-AE c assms*(*3*) *b*) *auto*
finally have $distr\ M\ (Pi_M\ I\ M')\ (\lambda x.\ \lambda i{\in}I.\ Y'\ i\ x) = Pi_M\ I\ (\lambda i.\ distr\ M\ (M'\ i)\ (Y'\ i))$
  by *simp*

thus *?thesis*
  using *True assms*(*3*)
  by (*subst indep-vars-iff-distr-eq-PiM'*) *auto*
next
case *False*
then show *?thesis*
  by (*simp add*:*indep-vars-def2 indep-sets-def*)

qed

lemma *indep-vars-reindex*:
  assumes *inj-on f I*
  assumes *indep-vars M′ X′* (*f* ‘ *I*)
  shows *indep-vars* (*M′* ∘ *f*) (λ*k ω*. *X′* (*f k*) *ω*) *I*
  using *assms* **by** (*simp add*:*indep-vars-def2 indep-sets-reindex*)

lemma *variance-divide*:
  fixes *f* :: ′*a* ⇒ *real*
  assumes *integrable M f*
  shows *variance* (λ*ω*. *f ω* / *r*) = *variance f* / *r*^*2*
  using *assms*
  by (*subst Bochner-Integration.integral-divide*[*OF assms*(*1*)])
    (*simp add*:*diff-divide-distrib*[*symmetric*] *power2-eq-square algebra-simps*)

lemma *pmf-mono*:
  assumes *M* = *measure-pmf p*
  assumes ⋀*x*. *x* ∈ *P* ⟹ *x* ∈ *set-pmf p* ⟹ *x* ∈ *Q*
  shows *prob P* ≤ *prob Q*
proof −
  have *prob P* = *prob* (*P* ∩ (*set-pmf p*))
    by (*rule measure-pmf-eq*[*OF assms*(*1*)], *blast*)
  also have ... ≤ *prob Q*
    using *assms* **by** (*intro finite-measure.finite-measure-mono*, *auto*)
  finally show *?thesis* **by** *simp*
qed

lemma *pmf-add*:
  assumes *M* = *measure-pmf p*
  assumes ⋀*x*. *x* ∈ *P* ⟹ *x* ∈ *set-pmf p* ⟹ *x* ∈ *Q* ∨ *x* ∈ *R*
  shows *prob P* ≤ *prob Q* + *prob R*
proof −
  have [*simp*]:*events* = *UNIV* **by** (*subst assms*(*1*), *simp*)
  have *prob P* ≤ *prob* (*Q* ∪ *R*)
    using *assms* **by** (*intro pmf-mono*[*OF assms*(*1*)], *blast*)
  also have ... ≤ *prob Q* + *prob R*
    by (*rule measure-subadditive*, *auto*)
  finally show *?thesis* **by** *simp*
qed

lemma *pmf-add-2*:
  assumes *M* = *measure-pmf p*
  assumes *prob* {*ω*. *P ω*} ≤ *r1*
  assumes *prob* {*ω*. *Q ω*} ≤ *r2*
  shows *prob* {*ω*. *P ω* ∨ *Q ω*} ≤ *r1* + *r2* (**is** *?lhs* ≤ *?rhs*)
proof −
  have *?lhs* ≤ *prob* {*ω*. *P ω*} + *prob* {*ω*. *Q ω*}
    by (*intro pmf-add*[*OF assms*(*1*)], *auto*)

also have ... $\leq$ *?rhs*
by (*intro add-mono assms*(*2*−*3*))
finally show *?thesis*
by *simp*
qed

definition *covariance* **where**
covariance f g = *expectation* ($\lambda\omega$. (*f* ω − *expectation f*) ∗ (*g* ω − *expectation g*))

lemma *real-prod-integrable*:
fixes *f g* :: *'a* ⇒ *real*
assumes [*measurable*]: *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
assumes *sq-int*: *integrable M* ($\lambda\omega$. *f* ω^*2*) *integrable M* ($\lambda\omega$. *g* ω^*2*)
shows *integrable M* ($\lambda\omega$. *f* ω ∗ *g* ω)
unfolding *integrable-iff-bounded*
proof
have ($\int^+ \omega$. *ennreal* (*norm* (*f* ω ∗ *g* ω)) $\partial M)^2$ = ($\int^+ \omega$. *ennreal* |*f* ω| ∗ *ennreal* |*g* ω| $\partial M)^2$
by (*simp add*: *abs-mult ennreal-mult*)
also have ... $\leq$ ($\int^+ \omega$. *ennreal* |*f* ω|^*2* ∂M) ∗ ($\int^+ \omega$. *ennreal* |*g* ω|^*2* ∂M)
by (*rule Cauchy-Schwarz-nn-integral*, *auto*)
also have ... < ∞
using *sq-int* **by** (*auto simp*: *integrable-iff-bounded ennreal-power ennreal-mult-less-top*)
finally have ($\int^+ x$. *ennreal* (*norm* (*f x* ∗ *g x*)) $\partial M)^2$ < ∞
by *simp*
thus ($\int^+ x$. *ennreal* (*norm* (*f x* ∗ *g x*)) ∂M) < ∞
by (*simp add*: *power-less-top-ennreal*)
qed *auto*

lemma *covariance-eq*:
fixes *f* :: *'a* ⇒ *real*
assumes *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
assumes *integrable M* ($\lambda\omega$. *f* ω^*2*) *integrable M* ($\lambda\omega$. *g* ω^*2*)
shows *covariance f g* = *expectation* ($\lambda\omega$. *f* ω ∗ *g* ω) − *expectation f* ∗ *expectation g*
proof −
have *integrable M f* **using** *square-integrable-imp-integrable assms* **by** *auto*
moreover have *integrable M g* **using** *square-integrable-imp-integrable assms* **by** *auto*
ultimately show *?thesis*
using *assms real-prod-integrable*
by (*simp add*:*covariance-def algebra-simps prob-space*)
qed

lemma *covar-integrable*:
fixes *f g* :: *'a* ⇒ *real*
assumes *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
assumes *integrable M* ($\lambda\omega$. *f* ω^*2*) *integrable M* ($\lambda\omega$. *g* ω^*2*)
shows *integrable M* ($\lambda\omega$. (*f* ω − *expectation f*) ∗ (*g* ω − *expectation g*))

proof −
have *integrable M f* **using** *square-integrable-imp-integrable assms* **by** *auto*
moreover have *integrable M g* **using** *square-integrable-imp-integrable assms* **by** *auto*
ultimately show *?thesis* **using** *assms real-prod-integrable* **by** (*simp add*: *algebra-simps*)
qed

lemma *sum-square-int*:
fixes $f :: 'b \Rightarrow 'a \Rightarrow real$
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in$ *borel-measurable M*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *integrable M* $(\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
shows *integrable M* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)^2)$
proof −
have *integrable M* $(\lambda\omega.\ \sum i{\in}I.\ \sum j{\in}I.\ f\ j\ \omega * f\ i\ \omega)$
using *assms*
by (*intro Bochner-Integration.integrable-sum real-prod-integrable*, *auto*)
thus *?thesis*
by (*simp add*:*power2-eq-square sum-distrib-left sum-distrib-right*)
qed

lemma *var-sum-1*:
fixes $f :: 'b \Rightarrow 'a \Rightarrow real$
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in$ *borel-measurable M*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *integrable M* $(\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
shows
variance $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.\ (\sum j \in I.$ *covariance* $(f\ i)\ (f\ j)))$
proof −
have *a*:$\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow$ *integrable M* $(\lambda\omega.\ (f\ i\ \omega -$ *expectation* $(f\ i)) * (f\ j\ \omega -$ *expectation* $(f\ j)))$
using *assms covar-integrable* **by** *simp*
have *variance* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) =$ *expectation* $(\lambda\omega.\ (\sum i{\in}I.\ f\ i\ \omega -$ *expectation* $(f\ i))^2)$
using *square-integrable-imp-integrable*[*OF assms*(*2*,*3*)]
by (*simp add*: *Bochner-Integration.integral-sum sum-subtractf*)
also have ... = *expectation* $(\lambda\omega.\ (\sum i \in I.\ (\sum j \in I.\ (f\ i\ \omega -$ *expectation* $(f\ i)) * (f\ j\ \omega -$ *expectation* $(f\ j)))))$
by (*simp add*: *power2-eq-square sum-distrib-right sum-distrib-left mult.commute*)
also have ... = $(\sum i \in I.\ (\sum j \in I.$ *covariance* $(f\ i)\ (f\ j)))$
using *a* **by** (*simp add*: *Bochner-Integration.integral-sum covariance-def*)
finally show *?thesis* **by** *simp*
qed

lemma *covar-self-eq*:
fixes $f :: 'a \Rightarrow real$
shows *covariance f f* = *variance f*
by (*simp add*:*covariance-def power2-eq-square*)

lemma *covar-indep-eq-zero*:
fixes *f g* :: *'a ⇒ real*
assumes *integrable M f*
assumes *integrable M g*
assumes *indep-var borel f borel g*
shows *covariance f g = 0*
proof −
have *a*:*indep-var borel* ((*λt. t − expectation f*) ∘ *f*) *borel* ((*λt. t − expectation g*) ∘ *g*)
by (*rule indep-var-compose*[*OF assms*(*3*)], *auto*)

have *b*:*expectation* (*λω.* (*f ω − expectation f*) ∗ (*g ω − expectation g*)) = *0*
using *a assms* **by** (*subst indep-var-lebesgue-integral*, *auto simp add*:*comp-def prob-space*)

thus *?thesis* **by** (*simp add*:*covariance-def*)
qed

lemma *var-sum-2*:
fixes *f* :: *'b ⇒ 'a ⇒ real*
assumes *finite I*
assumes $\bigwedge$*i. i ∈ I ⟹ f i ∈ borel-measurable M*
assumes $\bigwedge$*i. i ∈ I ⟹ integrable M* (*λω. f i ω^2*)
shows *variance* (*λω.* (∑ *i ∈ I. f i ω*)) =
(∑ *i ∈ I. variance* (*f i*)) + (∑ *i ∈ I.* ∑ *j ∈ I* − {*i*}. *covariance* (*f i*) (*f j*))
proof −
have *variance* (*λω.* (∑ *i ∈ I. f i ω*)) = (∑ *i∈I.* ∑ *j∈I. covariance* (*f i*) (*f j*))
by (*simp add*: *var-sum-1*[*OF assms*(*1*,*2*,*3*)])
also have ... = (∑ *i∈I. covariance* (*f i*) (*f i*) + (∑ *j∈I*−{*i*}. *covariance* (*f i*) (*f j*)))
using *assms* **by** (*subst sum.insert*[*symmetric*], *auto simp add*:*insert-absorb*)
also have ... = (∑ *i∈I. variance* (*f i*)) + (∑ *i ∈ I.* (∑ *j∈I*−{*i*}. *covariance* (*f i*) (*f j*)))
by (*simp add*: *covar-self-eq*[*symmetric*] *sum.distrib*)
finally show *?thesis* **by** *simp*
qed

lemma *var-sum-pairwise-indep*:
fixes *f* :: *'b ⇒ 'a ⇒ real*
assumes *finite I*
assumes $\bigwedge$*i. i ∈ I ⟹ f i ∈ borel-measurable M*
assumes $\bigwedge$*i. i ∈ I ⟹ integrable M* (*λω. f i ω^2*)
assumes $\bigwedge$*i j. i ∈ I ⟹ j ∈ I ⟹ i ≠ j ⟹ indep-var borel* (*f i*) *borel* (*f j*)
shows *variance* (*λω.* (∑ *i ∈ I. f i ω*)) = (∑ *i ∈ I. variance* (*f i*))
proof −
have $\bigwedge$*i j. i ∈ I ⟹ j ∈ I* − {*i*} *⟹ covariance* (*f i*) (*f j*) = *0*
using *covar-indep-eq-zero assms*(*4*) *square-integrable-imp-integrable*[*OF assms*(*2*,*3*)]
by *auto*

hence *a*:($\sum$ *i* ∈ *I*. $\sum$ *j* ∈ *I* − {*i*}. *covariance* (*f i*) (*f j*)) = *0*
by *simp*
thus *?thesis* **by** (*simp add*: *var-sum-2*[*OF assms*(*1*,*2*,*3*)])
qed

lemma *indep-var-from-indep-vars*:
assumes *i* ≠ *j*
assumes *indep-vars* (λ-. *M′*) *f* {*i*, *j*}
shows *indep-var M′* (*f i*) *M′* (*f j*)
proof −
have *a*:*inj* (*case-bool i j*) **using** *assms*(*1*)
by (*simp add*: *bool.case-eq-if inj-def*)
have *b*:*range* (*case-bool i j*) = {*i*,*j*}
by (*simp add*: *UNIV-bool insert-commute*)
have *c*:*indep-vars* (λ-. *M′*) *f* (*range* (*case-bool i j*)) **using** *assms*(*2*) *b* **by** *simp*

have *True* = *indep-vars* (λ*x*. *M′*) (λ*x*. *f* (*case-bool i j x*)) *UNIV*
using *indep-vars-reindex*[*OF a c*]
by (*simp add*:*comp-def*)
also have ... = *indep-vars* (λ*x*. *case-bool M′ M′ x*) (λ*x*. *case-bool* (*f i*) (*f j*) *x*) *UNIV*
by (*rule indep-vars-cong*, *auto simp*:*bool.case-distrib bool.case-eq-if*)
also have ... = *?thesis*
by (*simp add*: *indep-var-def*)
finally show *?thesis* **by** *simp*
qed

lemma *var-sum-pairwise-indep-2*:
fixes *f* :: *′b* ⇒ *′a* ⇒ *real*
assumes *finite I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel-measurable M*
assumes ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (λω. *f i* ω*^2*)
assumes ⋀*J*. *J* ⊆ *I* ⟹ *card J* = *2* ⟹ *indep-vars* (λ -. *borel*) *f J*
shows *variance* (λω. ($\sum$ *i* ∈ *I*. *f i* ω)) = ($\sum$ *i* ∈ *I*. *variance* (*f i*))
using *assms*(*4*)
by (*intro var-sum-pairwise-indep*[*OF assms*(*1*,*2*,*3*)] *indep-var-from-indep-vars*, *auto*)

lemma *var-sum-all-indep*:
fixes *f* :: *′b* ⇒ *′a* ⇒ *real*
assumes *finite I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel-measurable M*
assumes ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (λω. *f i* ω*^2*)
assumes *indep-vars* (λ -. *borel*) *f I*
shows *variance* (λω. ($\sum$ *i* ∈ *I*. *f i* ω)) = ($\sum$ *i* ∈ *I*. *variance* (*f i*))
by (*intro var-sum-pairwise-indep-2*[*OF assms*(*1*,*2*,*3*)] *indep-vars-subset*[*OF assms*(*4*)], *auto*)

end

end

6 Indexed Products of Probability Mass Functions

theory *Product-PMF-Ext*
imports *Main Probability-Ext HOL−Probability.Product-PMF Universal-Hash-Families.Preliminary-Results*
begin

hide-const *Isolated.discrete*

This section introduces a restricted version of *Pi-pmf* where the default value is *undefined* and contains some additional results about that case in addition to *HOL−Probability.Product-PMF*

abbreviation *prod-pmf* **where** *prod-pmf I M* ≡ *Pi-pmf I undefined M*

lemma *pmf-prod-pmf*:
assumes *finite I*
shows *pmf* (*prod-pmf I M*) *x* = (*if x* ∈ *extensional I then* ∏ *i* ∈ *I*. (*pmf* (*M i*)) (*x i*) *else 0*)
by (*simp add*: *pmf-Pi*[*OF assms*(*1*)] *extensional-def*)

lemma *PiE-defaut-undefined-eq*: *PiE-dflt I undefined M* = *PiE I M*
by (*simp add*:*PiE-dflt-def PiE-def extensional-def Pi-def set-eq-iff*) *blast*

lemma *set-prod-pmf*:
assumes *finite I*
shows *set-pmf* (*prod-pmf I M*) = *PiE I* (*set-pmf* ∘ *M*)
by (*simp add*:*set-Pi-pmf*[*OF assms*] *PiE-defaut-undefined-eq*)

A more general version of *measure-Pi-pmf-Pi*.

lemma *prob-prod-pmf'*:
assumes *finite I*
assumes *J* ⊆ *I*
shows *measure* (*measure-pmf* (*Pi-pmf I d M*)) (*Pi J A*) = (∏ *i* ∈ *J*. *measure* (*M i*) (*A i*))
proof −
have *a*:*Pi J A* = *Pi I* (λ*i*. *if i* ∈ *J then A i else UNIV*)
using *assms* **by** (*simp add*:*Pi-def set-eq-iff*, *blast*)
show *?thesis*
using *assms*
by (*simp add*:*if-distrib a measure-Pi-pmf-Pi*[*OF assms*(*1*)] *prod.If-cases*[*OF assms*(*1*)] *Int-absorb1*)
qed

lemma *prob-prod-pmf-slice*:
assumes *finite I*
assumes *i* ∈ *I*

shows *measure* (*measure-pmf* (*prod-pmf I M*)) {ω. *P* (ω *i*)} = *measure* (*M i*) {ω. *P* ω}
using *prob-prod-pmf'*[*OF assms*(*1*), **where** *J*={*i*} **and** *M*=*M* **and** *A*=λ-. *Collect P*]
by (*simp add*:*assms Pi-def*)

definition *restrict-dfl* **where** *restrict-dfl f A d* = (λ*x*. *if x* ∈ *A then f x else d*)

lemma *pi-pmf-decompose*:
assumes *finite I*
shows *Pi-pmf I d M* = *map-pmf* (λω. *restrict-dfl* (λ*i*. ω (*f i*) *i*) *I d*) (*Pi-pmf* (*f* ' *I*) (λ-. *d*) (λ*j*. *Pi-pmf* (*f* −' {*j*} ∩ *I*) *d M*))
proof −
have *fin-F-I*:*finite* (*f* ' *I*) **using** *assms* **by** *blast*

have *finite I* ⟹ *?thesis*
using *fin-F-I*
proof (*induction f* ' *I arbitrary*: *I rule*:*finite-induct*)
case *empty*
then show *?case* **by** (*simp add*:*restrict-dfl-def*)
next
case (*insert x F*)
have *a*: (*f* −' {*x*} ∩ *I*) ∪ (*f* −' *F* ∩ *I*) = *I*
using *insert*(*4*) **by** *blast*
have *b*: *F* = *f* ' (*f* −' *F* ∩ *I*) **using** *insert*(*2*,*4*)
by (*auto simp add*:*set-eq-iff image-def vimage-def*)
have *c*: *finite* (*f* −' *F* ∩ *I*) **using** *insert* **by** *blast*
have *d*: ⋀*j*. *j* ∈ *F* ⟹ (*f* −' {*j*} ∩ (*f* −' *F* ∩ *I*)) = (*f* −' {*j*} ∩ *I*)
using *insert*(*4*) **by** *blast*

have *Pi-pmf I d M* = *Pi-pmf* ((*f* −' {*x*} ∩ *I*) ∪ (*f* −' *F* ∩ *I*)) *d M*
by (*simp add*:*a*)
also have ... = *map-pmf* (λ(*g*, *h*) *i*. *if i* ∈ *f* −' {*x*} ∩ *I then g i else h i*) (*pair-pmf* (*Pi-pmf* (*f* −' {*x*} ∩ *I*) *d M*) (*Pi-pmf* (*f* −' *F* ∩ *I*) *d M*))
using *insert* **by** (*subst Pi-pmf-union*) *auto*
also have ... = *map-pmf* (λ(*g*,*h*) *i*. *if f i* = *x* ∧ *i* ∈ *I then g i else if f i* ∈ *F* ∧ *i* ∈ *I then h* (*f i*) *i else d*)
(*pair-pmf* (*Pi-pmf* (*f* −' {*x*} ∩ *I*) *d M*) (*Pi-pmf F* (λ-. *d*) (λ*j*. *Pi-pmf* (*f* −' {*j*} ∩ (*f* −' *F* ∩ *I*)) *d M*)))
by (*simp add*:*insert*(*3*)[*OF b c*] *map-pmf-comp case-prod-beta' apsnd-def map-prod-def*
pair-map-pmf2 b[*symmetric*] *restrict-dfl-def*) (*metis fst-conv snd-conv*)
also have ... = *map-pmf* (λ(*g*,*h*) *i*. *if i* ∈ *I then* (*h*(*x* := *g*)) (*f i*) *i else d*)
(*pair-pmf* (*Pi-pmf* (*f* −' {*x*} ∩ *I*) *d M*) (*Pi-pmf F* (λ-. *d*) (λ*j*. *Pi-pmf* (*f* −' {*j*} ∩ *I*) *d M*)))
using *insert*(*4*) *d*
by (*intro arg-cong2*[**where** *f*=*map-pmf*] *ext*) (*auto simp add*:*case-prod-beta' cong*:*Pi-pmf-cong*)

also have ... = *map-pmf* (*λω i. if i ∈ I then ω* (*f i*) *i else d*) (*Pi-pmf* (*insert x F*) (*λ-. d*) (*λj. Pi-pmf* (*f −‘ {j} ∩ I*) *d M*))
by (*simp add*:*Pi-pmf-insert*[*OF insert*(*1*,*2*)] *map-pmf-comp case-prod-beta′*)
finally show *?case* **by** (*simp add*:*insert*(*4*) *restrict-dfl-def*)
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *restrict-dfl-iter*: *restrict-dfl* (*restrict-dfl f I d*) *J d* = *restrict-dfl f* (*I ∩ J*) *d*
by (*rule ext*, *simp add*:*restrict-dfl-def*)

lemma *indep-vars-restrict′*:
assumes *finite I*
shows *prob-space.indep-vars* (*Pi-pmf I d M*) (*λ-. discrete*) (*λi ω. restrict-dfl ω* (*f −‘ {i} ∩ I*) *d*) (*f ‘ I*)
proof −
let *?Q* = (*Pi-pmf* (*f ‘ I*) (*λ-. d*) (*λi. Pi-pmf* (*I ∩ f −‘ {i}*) *d M*))
have *a*:*prob-space.indep-vars ?Q* (*λ-. discrete*) (*λi ω. ω i*) (*f ‘ I*)
using *assms* **by** (*intro indep-vars-Pi-pmf*, *blast*)
have *b*: *AE x in measure-pmf ?Q. ∀ i∈f ‘ I. x i = restrict-dfl* (*λi. x* (*f i*) *i*) (*I ∩ f −‘ {i}*) *d*
using *assms*
by (*auto simp add*:*PiE-dflt-def restrict-dfl-def AE-measure-pmf-iff set-Pi-pmf comp-def Int-commute*)
have *prob-space.indep-vars ?Q* (*λ-. discrete*) (*λi x. restrict-dfl* (*λi. x* (*f i*) *i*) (*I ∩ f −‘ {i}*) *d*) (*f ‘ I*)
by (*rule prob-space.indep-vars-cong-AE*[*OF prob-space-measure-pmf b a*], *simp*)
thus *?thesis*
using *prob-space-measure-pmf*
by (*auto intro*!:*prob-space.indep-vars-distr simp*:*pi-pmf-decompose*[*OF assms*, **where** *f*=*f*]
map-pmf-rep-eq comp-def restrict-dfl-iter Int-commute)
qed

lemma *indep-vars-restrict-intro′*:
assumes *finite I*
assumes ⋀*i ω. i ∈ J* ⟹ *X′ i ω* = *X′ i* (*restrict-dfl ω* (*f −‘ {i} ∩ I*) *d*)
assumes *J* = *f ‘ I*
assumes ⋀*ω i. i ∈ J* ⟹ *X′ i ω ∈ space* (*M′ i*)
shows *prob-space.indep-vars* (*measure-pmf* (*Pi-pmf I d p*)) *M′* (*λi ω. X′ i ω*) *J*
proof −
define *M* **where** *M* ≡ *measure-pmf* (*Pi-pmf I d p*)
interpret *prob-space M*
using *M-def prob-space-measure-pmf* **by** *blast*
have *indep-vars* (*λ-. discrete*) (*λi x. restrict-dfl x* (*f −‘ {i} ∩ I*) *d*) (*f ‘ I*)
unfolding *M-def* **by** (*rule indep-vars-restrict′*[*OF assms*(*1*)])
hence *indep-vars M′* (*λi ω. X′ i* (*restrict-dfl ω* (*f −‘ {i} ∩ I*) *d*)) (*f ‘ I*)
using *assms*(*4*)

by (*intro indep-vars-compose2*[**where** $Y=X'$ **and** $N=M'$ **and** $M'=\lambda$-. *discrete*]) (*auto simp*:*assms*(*3*))
hence *indep-vars* M' ($\lambda i\ \omega.\ X'\ i\ \omega$) ($f$ ‘ I)
using *assms*(*2*)[*symmetric*]
by (*simp add*:*assms*(*3*) *cong*:*indep-vars-cong*)
thus *?thesis*
unfolding *M-def* **using** *assms*(*3*) **by** *simp*
qed

lemma
fixes f :: $'b \Rightarrow ('c$ :: {*second-countable-topology*,*banach*,*real-normed-field*})
assumes *finite I*
assumes $i \in I$
assumes *integrable* (*measure-pmf* (M i)) f
shows *integrable-Pi-pmf-slice*: *integrable* (*Pi-pmf I d M*) ($\lambda x.\ f\ (x\ i)$)
and *expectation-Pi-pmf-slice*: *integral*^{L} (*Pi-pmf I d M*) ($\lambda x.\ f\ (x\ i)$) = *integral*^{L} (M i) f
proof −
have *a*:*distr* (*Pi-pmf I d M*) (M i) ($\lambda\omega.\ \omega\ i$) = *distr* (*Pi-pmf I d M*) *discrete* ($\lambda\omega.\ \omega\ i$)
by (*rule distr-cong*, *auto*)

have *b*: *measure-pmf*.*random-variable* (M i) *borel f*
using *assms*(*3*) **by** *simp*

have *c*:*integrable* (*distr* (*Pi-pmf I d M*) (M i) ($\lambda\omega.\ \omega\ i$)) f
using *assms*(*1*,*2*,*3*)
by (*subst a*, *subst map-pmf-rep-eq*[*symmetric*], *subst Pi-pmf-component*, *auto*)

show *integrable* (*Pi-pmf I d M*) ($\lambda x.\ f\ (x\ i)$)
by (*rule integrable-distr*[**where** $f=f$ **and** $M'=M\ i$]) (*auto intro*: *c*)

have *integral*^{L} (*Pi-pmf I d M*) ($\lambda x.\ f\ (x\ i)$) = *integral*^{L} (*distr* (*Pi-pmf I d M*) (M i) ($\lambda\omega.\ \omega\ i$)) f
using *b* **by** (*intro integral-distr*[*symmetric*], *auto*)
also have ... = *integral*^{L} (*map-pmf* ($\lambda\omega.\ \omega\ i$) (*Pi-pmf I d M*)) f
by (*subst a*, *subst map-pmf-rep-eq*[*symmetric*], *simp*)
also have ... = *integral*^{L} (M i) f
using *assms*(*1*,*2*) **by** (*simp add*: *Pi-pmf-component*)
finally show *integral*^{L} (*Pi-pmf I d M*) ($\lambda x.\ f\ (x\ i)$) = *integral*^{L} (M i) f **by** *simp*
qed

This is an improved version of *expectation-prod-Pi-pmf*. It works for general normed fields instead of non-negative real functions .

lemma *expectation-prod-Pi-pmf*:
fixes f :: $'a \Rightarrow 'b \Rightarrow ('c$ :: {*second-countable-topology*,*banach*,*real-normed-field*})
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *integrable* (*measure-pmf* (M i)) (f i)
shows *integral*^{L} (*Pi-pmf I d M*) ($\lambda x.\ (\prod i \in I.\ f\ i\ (x\ i))$) = ($\prod\ i \in I.$ *integral*^{L}

(*M i*) (*f i*))
proof −
  have *a*: *prob-space.indep-vars* (*measure-pmf* (*Pi-pmf I d M*)) (λ-. *borel*) (λ*i* ω. *f i* (ω *i*)) *I*
    by (*intro prob-space.indep-vars-compose2*[**where** *Y*=*f* **and** *M'*=λ-. *discrete*]
        prob-space-measure-pmf indep-vars-Pi-pmf assms(*1*)) *auto*
  have *integral*^{L} (*Pi-pmf I d M*) (λ*x*. (∏ *i* ∈ *I*. *f i* (*x i*))) = (∏ *i* ∈ *I*. *integral*^{L} (*Pi-pmf I d M*) (λ*x*. *f i* (*x i*)))
    by (*intro prob-space.indep-vars-lebesgue-integral prob-space-measure-pmf assms*(*1*,*2*)

        a integrable-Pi-pmf-slice) *auto*
  also have ... = (∏ *i* ∈ *I*. *integral*^{L} (*M i*) (*f i*))
    by (*intro prod.cong expectation-Pi-pmf-slice assms*(*1*,*2*)) *auto*
  finally show *?thesis* **by** *simp*
qed

lemma *variance-prod-pmf-slice*:
  fixes *f* :: *'a* ⇒ *real*
  assumes *i* ∈ *I finite I*
  assumes *integrable* (*measure-pmf* (*M i*)) (λω. *f* ω^*2*)
  shows *prob-space.variance* (*Pi-pmf I d M*) (λω. *f* (ω *i*)) = *prob-space.variance* (*M i*) *f*
proof −
  have *a*:*integrable* (*measure-pmf* (*M i*)) *f*
    using *assms*(*3*) *measure-pmf.square-integrable-imp-integrable* **by** *auto*
  have *b*: *integrable* (*measure-pmf* (*Pi-pmf I d M*)) (λ*x*. (*f* (*x i*))^2)
    by (*rule integrable-Pi-pmf-slice*[*OF assms*(*2*) *assms*(*1*)], *metis assms*(*3*))
  have *c*: *integrable* (*measure-pmf* (*Pi-pmf I d M*)) (λ*x*. (*f* (*x i*)))
    by (*rule integrable-Pi-pmf-slice*[*OF assms*(*2*) *assms*(*1*)], *metis a*)

  have *measure-pmf.expectation* (*Pi-pmf I d M*) (λ*x*. (*f* (*x i*))^2) − (*measure-pmf.expectation* (*Pi-pmf I d M*) (λ*x*. *f* (*x i*)))^2 =
      measure-pmf.expectation (*M i*) (λ*x*. (*f x*)^2) − (*measure-pmf.expectation* (*M i*) *f*)^2
    using *assms a b c* **by** ((*subst expectation-Pi-pmf-slice*[*OF assms*(*2*,*1*)])*?*, *simp*)+

  thus *?thesis*
    using *assms a b c* **by** (*simp add*: *measure-pmf.variance-eq*)
qed

lemma *Pi-pmf-bind-return*:
  assumes *finite I*
  shows *Pi-pmf I d* (λ*i*. *M i* ≫= (λ*x*. *return-pmf* (*f i x*))) = *Pi-pmf I d' M* ≫= (λ*x*. *return-pmf* (λ*i*. *if i* ∈ *I then f i* (*x i*) *else d*))
  using *assms* **by** (*simp add*: *Pi-pmf-bind*[**where** *d'*=*d'*])

end

7 Frequency Moment 0

```
theory Frequency-Moment-0
  imports
    Frequency-Moments-Preliminary-Results
    Median-Method.Median
    K-Smallest
    Universal-Hash-Families.Carter-Wegman-Hash-Family
    Frequency-Moments
    Landau-Ext
    Product-PMF-Ext
    Universal-Hash-Families.Field
begin
```

This section contains a formalization of a new algorithm for the zero-th frequency moment inspired by ideas described in [2]. It is a KMV-type (k-minimum value) algorithm with a rounding method and matches the space complexity of the best algorithm described in [2].

In addition to the Isabelle proof here, there is also an informal hand-written proof in Appendix A.

```
type-synonym f0-state = nat × nat × nat × nat × (nat ⇒ nat list) × (nat ⇒
float set)

definition hash where hash p = ring.hash (mod-ring p)

fun f0-init :: rat ⇒ rat ⇒ nat ⇒ f0-state pmf where
  f0-init δ ε n =
    do {
      let s = nat ⌈−18 ∗ ln (real-of-rat ε)⌉;
      let t = nat ⌈80 / (real-of-rat δ)^2⌉;
      let p = prime-above (max n 19);
      let r = nat (4 ∗ ⌈log 2 (1 / real-of-rat δ)⌉ + 23);
      h ← prod-pmf {..<s} (λ-. pmf-of-set (bounded-degree-polynomials (mod-ring
p) 2));
      return-pmf (s, t, p, r, h, (λ- ∈ {0..<s}. {}))
    }

fun f0-update :: nat ⇒ f0-state ⇒ f0-state pmf where
  f0-update x (s, t, p, r, h, sketch) =
    return-pmf (s, t, p, r, h, λi ∈ {..<s}.
      least t (insert (float-of (truncate-down r (hash p x (h i)))) (sketch i)))

fun f0-result :: f0-state ⇒ rat pmf where
  f0-result (s, t, p, r, h, sketch) = return-pmf (median s (λi ∈ {..<s}.
      (if card (sketch i) < t then of-nat (card (sketch i)) else
        rat-of-nat t∗ rat-of-nat p / rat-of-float (Max (sketch i)))
    ))

fun f0-space-usage :: (nat × rat × rat) ⇒ real where
```

f0-space-usage (n, ε, δ) = (
let s = *nat* $\lceil -18 * ln$ (*real-of-rat* ε)$\rceil$ *in*
let r = *nat* (*4* $*$ $\lceil$*log 2* (*1* / *real-of-rat* δ)$\rceil$ + *23*) *in*
let t = *nat* $\lceil$*80* / (*real-of-rat* $\delta)^2$ $\rceil$ *in*
6 +
2 $*$ *log 2* (*real* s + *1*) +
2 $*$ *log 2* (*real* t + *1*) +
2 $*$ *log 2* (*real* n + *21*) +
2 $*$ *log 2* (*real* r + *1*) +
real s $*$ (*5* + *2* $*$ *log 2* (*21* + *real* n) +
real t $*$ (*13* + *4* $*$ r + *2* $*$ *log 2* (*log 2* (*real* n + *13*)))))

definition *encode-f0-state* :: *f0-state* ⇒ *bool list option* **where**
encode-f0-state =
$N_e \bowtie_e (\lambda s.$
$N_e \times_e ($
$N_e \bowtie_e (\lambda p.$
$N_e \times_e ($
([*0*..<s] $\rightarrow_e$ (P_e p *2*)) $\times_e$
([*0*..<s] $\rightarrow_e$ (S_e F_e))))))

lemma *inj-on encode-f0-state* (*dom encode-f0-state*)
proof −
have *is-encoding encode-f0-state*
unfolding *encode-f0-state-def*
by (*intro dependent-encoding exp-golomb-encoding poly-encoding fun-encoding set-encoding float-encoding*)
thus *?thesis* **by** (*rule encoding-imp-inj*)
qed

context
fixes ε δ :: *rat*
fixes n :: *nat*
fixes *as* :: *nat list*
fixes *result*
assumes *ε-range*: $\varepsilon \in \{0<..<1\}$
assumes *δ-range*: $\delta \in \{0<..<1\}$
assumes *as-range*: *set as* ⊆ {..<n}
defines *result* ≡ *fold* (λ*a state. state* ⤜ *f0-update a*) *as* (*f0-init* δ ε n) ⤜ *f0-result*
begin

private definition t **where** t = *nat* $\lceil$*80* / (*real-of-rat* $\delta)^2\rceil$
private lemma *t-gt-0*: $t > 0$ **using** *δ-range* **by** (*simp add:t-def*)

private definition s **where** s = *nat* $\lceil -(18 * ln$ (*real-of-rat* ε))$\rceil$
private lemma *s-gt-0*: $s > 0$ **using** *ε-range* **by** (*simp add:s-def*)

private definition p **where** p = *prime-above* (*max* n *19*)

```
private lemma p-prime:Factorial-Ring.prime p
  using p-def prime-above-prime by presburger

private lemma p-ge-18: p ≥ 18
proof -
  have p ≥ 19
    by (metis p-def prime-above-lower-bound max.bounded-iff)
  thus ?thesis by simp
qed

private lemma p-gt-0: p > 0 using p-ge-18 by simp
private lemma p-gt-1: p > 1 using p-ge-18 by simp

private lemma n-le-p: n ≤ p
proof -
  have n ≤ max n 19 by simp
  also have ... ≤ p
    unfolding p-def by (rule prime-above-lower-bound)
  finally show ?thesis by simp
qed

private lemma p-le-n: p ≤ 2*n + 40
proof -
  have p ≤ 2 * (max n 19) + 2
    by (subst p-def, rule prime-above-upper-bound)
  also have ... ≤ 2 * n + 40
    by (cases n ≥ 19, auto)
  finally show ?thesis by simp
qed

private lemma as-lt-p: ⋀x. x ∈ set as ⟹ x < p
  using as-range atLeastLessThan-iff
  by (intro order-less-le-trans[OF - n-le-p]) blast

private lemma as-subset-p: set as ⊆ {..<p}
   using as-lt-p  by (simp add: subset-iff)

private definition r where r = nat (4 * ⌈log 2 (1 / real-of-rat δ)⌉ + 23)

private lemma r-bound: 4 * log 2 (1 / real-of-rat δ) + 23 ≤ r
proof -
  have 0 ≤ log 2 (1 / real-of-rat δ) using δ-range by simp
  hence 0 ≤ ⌈log 2 (1 / real-of-rat δ)⌉ by simp
  hence 0 ≤ 4 * ⌈log 2 (1 / real-of-rat δ)⌉ + 23
    by (intro add-nonneg-nonneg mult-nonneg-nonneg, auto)
  thus ?thesis by (simp add:r-def)
qed
```

private lemma *r-ge-23*: $r \geq 23$
proof −
  have $(23::real) = 0 + 23$ **by** *simp*
  also have ... $\leq 4 * log\ 2\ (1\ /\ real\text{-}of\text{-}rat\ \delta) + 23$
    using *δ-range* **by** (*intro add-mono mult-nonneg-nonneg*, *auto*)
  also have ... $\leq r$ **using** *r-bound* **by** *simp*
  finally show $23 \leq r$ **by** *simp*
qed

private lemma *two-pow-r-le-1*: $0 < 1 - 2\ powr - real\ r$
proof −
  have *a*: *2 powr* (*0::real*) = *1*
    by *simp*
  show *?thesis* **using** *r-ge-23*
    by (*simp*, *subst a*[*symmetric*], *intro powr-less-mono*, *auto*)
qed

interpretation *carter-wegman-hash-family mod-ring p 2*
  rewrites *ring.hash* (*mod-ring p*) = *Frequency-Moment-0.hash p*
  using *carter-wegman-hash-familyI*[*OF mod-ring-is-field mod-ring-finite*]
  using *hash-def p-prime* **by** *auto*

private definition *tr-hash* **where** *tr-hash x ω* = *truncate-down r* (*hash x ω*)

private definition *sketch-rv* **where**
  sketch-rv ω = *least t* ((*λx. float-of* (*tr-hash x ω*)) ‘ *set as*)

private definition *estimate*
   where *estimate S* = (*if card S* < *t then of-nat* (*card S*) *else of-nat t* ∗ *of-nat p* / *rat-of-float* (*Max S*))

private definition *sketch-rv′* **where** *sketch-rv′ ω* = *least t* ((*λx. tr-hash x ω*) ‘ *set as*)
private definition *estimate′* **where** *estimate′ S* = (*if card S* < *t then real* (*card S*) *else real t* ∗ *real p* / *Max S*)

private definition Ω_0 **where** Ω_0 = *prod-pmf* {..<*s*} (*λ-. pmf-of-set space*)

private lemma *f0-alg-sketch*:
  defines *sketch* ≡ *fold* (*λa state. state* ⤜ *f0-update a*) *as* (*f0-init δ ε n*)
  shows *sketch* = *map-pmf* (*λx.* (*s*,*t*,*p*,*r*, *x*, *λi* ∈ {..<*s*}. *sketch-rv* (*x i*))) Ω_0
  unfolding *sketch-rv-def*
proof (*subst sketch-def*, *induction as rule*:*rev-induct*)
  case *Nil*
  then show *?case*
    by (*simp add*:*s-def p-def*[*symmetric*] *map-pmf-def t-def r-def Let-def least-def restrict-def space-def* Ω_0*-def*)
next
  case (*snoc x xs*)

```
  let ?sketch = λω xs. least t ((λa. float-of (tr-hash a ω)) ‘ set xs)
  have fold (λa state. state ≫= f0-update a) (xs @ [x]) (f0-init δ ε n) =
     (map-pmf (λω. (s, t, p, r, ω, λi ∈ {..<s}. ?sketch (ω i) xs)) Ω0) ≫= f0-update
x
    by (simp add: restrict-def snoc del:f0-init.simps)
  also have ... = Ω0 ≫= (λω. f0-update x (s, t, p, r, ω, λi∈{..<s}. ?sketch (ω i)
xs))
    by (simp add:map-pmf-def bind-assoc-pmf bind-return-pmf del:f0-update.simps)
  also have ... = map-pmf (λω. (s, t, p, r, ω, λi∈{..<s}. ?sketch (ω i) (xs@[x])))
Ω0
    by (simp add:least-insert map-pmf-def tr-hash-def cong:restrict-cong)
  finally show ?case by blast
qed

private lemma card-nat-in-ball:
  fixes x :: nat
  fixes q :: real
  assumes q ≥ 0
  defines A ≡ {k. abs (real x − real k) ≤ q ∧ k ≠ x}
  shows real (card A) ≤ 2 ∗ q and finite A
proof −
  have a: of-nat x ∈ {⌈real x−q⌉..⌊real x+q⌋}
    using assms
    by (simp add: ceiling-le-iff)

  have card A = card (int ‘ A)
    by (rule card-image[symmetric], simp)
  also have ... ≤ card ({⌈real x−q⌉..⌊real x+q⌋} − {of-nat x})
    by (intro card-mono image-subsetI, simp-all add:A-def abs-le-iff, linarith)
  also have ... = card {⌈real x−q⌉..⌊real x+q⌋} − 1
    by (rule card-Diff-singleton, rule a)
  also have ... = int (card {⌈real x−q⌉..⌊real x+q⌋}) − int 1
    by (intro of-nat-diff)
     (metis a card-0-eq empty-iff finite-atLeastAtMost-int less-one linorder-not-le)
  also have ... ≤ ⌊q+real x⌋+1 −⌈real x−q⌉ − 1
    using assms by (simp, linarith)
  also have ... ≤ 2∗q
    by linarith
  finally show card A ≤ 2 ∗ q
    by simp

  have A ⊆ {..x + nat ⌈q⌉}
    by (rule subsetI, simp add:A-def abs-le-iff, linarith)
  thus finite A
    by (rule finite-subset, simp)
qed

private lemma prob-degree-lt-1:
   prob {ω. degree ω < 1} ≤ 1/real p
```

proof −
  have *space* ∩ {ω. *length* ω ≤ *Suc 0*} = *bounded-degree-polynomials* (*mod-ring p*) *1*
    by (*auto simp*:*set-eq-iff bounded-degree-polynomials-def space-def*)
  moreover have *field-size* = *p* **by** (*simp add*:*mod-ring-def*)
  hence *real* (*card* (*bounded-degree-polynomials* (*mod-ring p*) (*Suc 0*))) / *real* (*card space*) = *1* / *real p*
    by (*simp add*:*space-def bounded-degree-polynomials-card power2-eq-square*)
  ultimately show *?thesis*
    by (*simp add*:*M-def measure-pmf-of-set*)
qed

private lemma *collision-prob*:
  assumes $c \geq 1$
  shows *prob* {ω. ∃ *x* ∈ *set as*. ∃ *y* ∈ *set as*. *x* ≠ *y* ∧ *tr-hash x* ω ≤ *c* ∧ *tr-hash x* ω = *tr-hash y* ω} ≤
    $(5/2) * (real\ (card\ (set\ as)))^2 * c^2 * 2\ powr\ -(real\ r)\ /\ (real\ p)^2 + 1/real\ p$
(**is** *prob* {ω. *?l* ω} ≤ *?r1* + *?r2*)
proof −
  define ϱ :: *real* **where** ϱ = *9/8*

  have *rho-c-ge-0*: ϱ ∗ *c* ≥ *0* **unfolding** *ϱ-def* **using** *assms* **by** *simp*

  have *c-ge-0*: *c*≥*0* **using** *assms* **by** *simp*

  have *degree* ω ≥ *1* ⟹ ω ∈ *space* ⟹ *degree* ω = *1* **for** ω
    by (*simp add*:*bounded-degree-polynomials-def space-def*)
     (*metis One-nat-def Suc-1 le-less-Suc-eq less-imp-diff-less list.size*(*3*) *pos2*)

  hence *a*: ⋀ω *x y*. *x* < *p* ⟹ *y* < *p* ⟹ *x* ≠ *y* ⟹ *degree* ω ≥ *1* ⟹ ω ∈ *space* ⟹ *hash x* ω ≠ *hash y* ω
    using *inj-onD*[*OF inj-if-degree-1*] *mod-ring-carr* **by** *blast*

  have *b*: *prob* {ω. *degree* ω ≥ *1* ∧ *tr-hash x* ω ≤ *c* ∧ *tr-hash x* ω = *tr-hash y* ω} ≤ $5 * c^2 * 2\ powr\ (-real\ r)\ /(real\ p)^2$
    if *b-assms*: *x* ∈ *set as* *y* ∈ *set as* *x* < *y* **for** *x y*
  proof −
    have *c*: *real u* ≤ ϱ ∗ *c* ∧ |*real u* − *real v*| ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*)
      if *c-assms*:*truncate-down r* (*real u*) ≤ *c* *truncate-down r* (*real u*) = *truncate-down r* (*real v*) **for** *u v*
    proof −
      have *9* ∗ *2 powr* − *real r* ≤ *9* ∗ *2 powr* (− *real 23*)
        using *r-ge-23* **by** (*intro mult-left-mono powr-mono*, *auto*)

      also have ... ≤ *1* **by** *simp*

      finally have *9* ∗ *2 powr* − *real r* ≤ *1* **by** *simp*

      hence *1* ≤ ϱ ∗ (*1* − *2 powr* (− *real r*))

by (*simp add*:*ϱ-def*)

hence *d*: (*c*∗*1*) / (*1* − *2 powr* (−*real r*)) ≤ *c* ∗ *ϱ*
using *assms two-pow-r-le-1* **by** (*simp add*: *pos-divide-le-eq*)

have ⋀*x*. *truncate-down r* (*real x*) ≤ *c* ⟹ *real x* ∗ (*1* − *2 powr* − *real r*) ≤ *c* ∗ *1*
using *truncate-down-pos*[*OF of-nat-0-le-iff*] *order-trans* **by** (*simp*, *blast*)

hence ⋀*x*. *truncate-down r* (*real x*) ≤ *c* ⟹ *real x* ≤ *c* ∗ *ϱ*
using *two-pow-r-le-1* **by** (*intro order-trans*[*OF - d*], *simp add*: *pos-le-divide-eq*)

hence *e*: *real u* ≤ *c* ∗ *ϱ real v* ≤ *c* ∗ *ϱ*
using *c-assms* **by** *auto*

have |*real u* − *real v*| ≤ (*max* |*real u*| |*real v*|) ∗ *2 powr* (−*real r*)
using *c-assms* **by** (*intro truncate-down-eq*, *simp*)

also have ... ≤ (*c* ∗ *ϱ*) ∗ *2 powr* (−*real r*)
using *e* **by** (*intro mult-right-mono*, *auto*)

finally have |*real u* − *real v*| ≤ *ϱ* ∗ *c* ∗ *2 powr* (−*real r*)
by (*simp add*:*algebra-simps*)

thus *?thesis* **using** *e* **by** (*simp add*:*algebra-simps*)
qed

have *prob* {*ω*. *degree ω* ≥ *1* ∧ *tr-hash x ω* ≤ *c* ∧ *tr-hash x ω* = *tr-hash y ω*} ≤ *prob* (⋃ *i* ∈ {(*u*,*v*) ∈ {..<*p*} × {..<*p*}. *u* ≠ *v* ∧ *truncate-down r u* ≤ *c* ∧ *truncate-down r u* = *truncate-down r v*}.
{*ω*. *hash x ω* = *fst i* ∧ *hash y ω* = *snd i*})
using *a* **by** (*intro pmf-mono*[*OF M-def*], *simp add*:*tr-hash-def*)
(*metis hash-range mod-ring-carr b-assms as-subset-p lessThan-iff nat-neq-iff subset-eq*)

also have ... ≤ (∑ *i*∈ {(*u*,*v*) ∈ {..<*p*} × {..<*p*}. *u* ≠ *v* ∧
truncate-down r u ≤ *c* ∧ *truncate-down r u* = *truncate-down r v*}.
prob {*ω*. *hash x ω* = *fst i* ∧ *hash y ω* = *snd i*})
by (*intro measure-UNION-le finite-cartesian-product finite-subset*[**where** *B*={*0*..<*p*} × {*0*..<*p*}])
(*auto simp add*:*M-def*)

also have ... ≤ (∑ *i*∈ {(*u*,*v*) ∈ {..<*p*} × {..<*p*}. *u* ≠ *v* ∧
truncate-down r u ≤ *c* ∧ *truncate-down r u* = *truncate-down r v*}.
prob {*ω*. (∀ *u* ∈ {*x*,*y*}. *hash u ω* = (*if u* = *x then* (*fst i*) *else* (*snd i*)))})
by (*intro sum-mono pmf-mono*[*OF M-def*]) *force*

also have ... ≤ (∑ *i*∈ {(*u*,*v*) ∈ {..<*p*} × {..<*p*}. *u* ≠ *v* ∧

truncate-down r u ≤ c ∧ truncate-down r u = truncate-down r v}. *1*/(*real p*)^2)
using *assms as-subset-p b-assms*
by (*intro sum-mono*, *subst hash-prob*) (*auto simp add*: *mod-ring-def power2-eq-square*)

also have ... = *1*/(*real p*)^2 ∗
card {(*u*,*v*) ∈ {*0*..<*p*} × {*0*..<*p*}. *u* ≠ *v* ∧ *truncate-down r u* ≤ *c* ∧ *truncate-down r u* = *truncate-down r v*}
by *simp*

also have ... ≤ *1*/(*real p*)^2 ∗
card {(*u*,*v*) ∈ {..<*p*} × {..<*p*}. *u* ≠ *v* ∧ *real u* ≤ ϱ ∗ *c* ∧ *abs* (*real u* − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*)}
using *c*
by (*intro mult-mono of-nat-mono card-mono finite-cartesian-product finite-subset*[**where** *B*={..<*p*}×{..<*p*}])
auto

also have ... ≤ *1*/(*real p*)^2 ∗ *card* (⋃ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
{(*u*::*nat*,*v*::*nat*). *u* = *u*′ ∧ *abs* (*real u* − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* < *p* ∧ *v* ≠ *u*′})
by (*intro mult-left-mono of-nat-mono card-mono finite-cartesian-product finite-subset*[**where** *B*={..<*p*}×{..<*p*}])
auto

also have ... ≤ *1*/(*real p*)^2 ∗ (∑ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
card {(*u*,*v*). *u* = *u*′ ∧ *abs* (*real u* − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* < *p* ∧ *v* ≠ *u*′})
by (*intro mult-left-mono of-nat-mono card-UN-le*, *auto*)

also have ... = *1*/(*real p*)^2 ∗ (∑ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
card ((λ*x*. (*u*′ ,*x*)) ‘ {*v*. *abs* (*real u*′ − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* < *p* ∧ *v* ≠ *u*′}))
by (*intro arg-cong2*[**where** *f*=(∗)] *arg-cong*[**where** *f*=*real*] *sum.cong arg-cong*[**where** *f*=*card*])
(*auto simp add*:*set-eq-iff*)

also have ... ≤ *1*/(*real p*)^2 ∗ (∑ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
card {*v*. *abs* (*real u*′ − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* < *p* ∧ *v* ≠ *u*′})
by (*intro mult-left-mono of-nat-mono sum-mono card-image-le*, *auto*)

also have ... ≤ *1*/(*real p*)^2 ∗ (∑ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
card {*v*. *abs* (*real u*′ − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* ≠ *u*′})
by (*intro mult-left-mono sum-mono of-nat-mono card-mono card-nat-in-ball subsetI*) *auto*

also have ... ≤ *1*/(*real p*)^2 ∗ (∑ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ ϱ ∗ *c*}.
real (*card* {*v*. *abs* (*real u*′ − *real v*) ≤ ϱ ∗ *c* ∗ *2 powr* (−*real r*) ∧ *v* ≠ *u*′}))
by *simp*

also have ... $\leq$ *1*/(*real p*)^2 $*$ ($\sum$ *u*′ ∈ {*u*. *u* < *p* ∧ *real u* ≤ *ϱ* ∗ *c*}. *2* ∗ (*ϱ* ∗ *c* ∗ *2 powr* (−*real r*)))
by (*intro mult-left-mono sum-mono card-nat-in-ball*(*1*), *auto*)

also have ... = *1*/(*real p*)^2 ∗ (*real* (*card* {*u*. *u* < *p* ∧ *real u* ≤ *ϱ* ∗ *c*}) ∗ (*2* ∗ (*ϱ* ∗ *c* ∗ *2 powr* (−*real r*))))
by *simp*

also have ... ≤ *1*/(*real p*)^2 ∗ (*real* (*card* {*u*. *u* ≤ *nat* (⌊*ϱ* ∗ *c* ⌋)}) ∗ (*2* ∗ (*ϱ* ∗ *c* ∗ *2 powr* (−*real r*))))
using *rho-c-ge-0 le-nat-floor*
by (*intro mult-left-mono mult-right-mono of-nat-mono card-mono subsetI*) *auto*

also have ... ≤ *1*/(*real p*)^2 ∗ ((*1*+*ϱ* ∗ *c*) ∗ (*2* ∗ (*ϱ* ∗ *c* ∗ *2 powr* (−*real r*))))
using *rho-c-ge-0* **by** (*intro mult-left-mono mult-right-mono*, *auto*)

also have ... ≤ *1*/(*real p*)^2 ∗ (((*1*+*ϱ*) ∗ *c*) ∗ (*2* ∗ (*ϱ* ∗ *c* ∗ *2 powr* (−*real r*))))
using *assms* **by** (*intro mult-mono*, *auto simp add*:*distrib-left distrib-right ϱ-def*)

also have ... = (*ϱ* ∗ (*2* + *ϱ* ∗ *2*)) ∗ *c*^{2} ∗ *2 powr* (−*real r*) /(*real p*)^2
by (*simp add*:*ac-simps power2-eq-square*)

also have ... ≤ *5* ∗ *c*^{2} ∗ *2 powr* (−*real r*) /(*real p*)^2
by (*intro divide-right-mono mult-right-mono*) (*auto simp add*:*ϱ-def*)

finally show *?thesis* **by** *simp*
qed

have *prob* {*ω*. *?l ω* ∧ *degree ω* ≥ *1*} ≤
prob (⋃ *i* ∈ {(*x*,*y*) ∈ (*set as*) × (*set as*). *x* < *y*}. {*ω*. *degree ω* ≥ *1* ∧ *tr-hash* (*fst i*) *ω* ≤ *c* ∧
tr-hash (*fst i*) *ω* = *tr-hash* (*snd i*) *ω*})
by (*rule pmf-mono*[*OF M-def*], *simp*, *metis linorder-neqE-nat*)

also have ... ≤ ($\sum$ *i* ∈ {(*x*,*y*) ∈ (*set as*) × (*set as*). *x* < *y*}. *prob*
{*ω*. *degree ω* ≥ *1* ∧ *tr-hash* (*fst i*) *ω* ≤ *c* ∧ *tr-hash* (*fst i*) *ω* = *tr-hash* (*snd i*) *ω*})
unfolding *M-def*
by (*intro measure-UNION-le finite-cartesian-product finite-subset*[**where** *B*=(*set as*) × (*set as*)])
auto

also have ... ≤ ($\sum$ *i* ∈ {(*x*,*y*) ∈ (*set as*) × (*set as*). *x* < *y*}. *5* ∗ *c*^{2} ∗ *2 powr* (−*real r*) /(*real p*)^2)
using *b* **by** (*intro sum-mono*, *simp add*:*case-prod-beta*)

also have ... = ((*5/2*) ∗ c^2 ∗ *2 powr* (−*real r*) /(*real p*)^2) ∗ (*2* ∗ *card* {(*x*,*y*) ∈ (*set as*) × (*set as*). *x* < *y*})
by *simp*

also have ... = ((*5/2*) ∗ c^2 ∗ *2 powr* (−*real r*) /(*real p*)^2) ∗ (*card* (*set as*) ∗ (*card* (*set as*) − *1*))
by (*subst card-ordered-pairs*, *auto*)

also have ... ≤ ((*5/2*) ∗ c^2 ∗ *2 powr* (−*real r*) /(*real p*)^2) ∗ (*real* (*card* (*set as*)))^2
by (*intro mult-left-mono*) (*auto simp add*:*power2-eq-square mult-left-mono*)

also have ... = (*5/2*) ∗ (*real* (*card* (*set as*)))^2 ∗ c^2 ∗ *2 powr* (−*real r*) /(*real p*)^2
by (*simp add*:*algebra-simps*)

finally have *f*:*prob* {ω. *?l* ω ∧ *degree* ω ≥ *1*} ≤ *?r1* **by** *simp*

have *prob* {ω. *?l* ω} ≤ *prob* {ω. *?l* ω ∧ *degree* ω ≥ *1*} + *prob* {ω. *degree* ω < *1*}
by (*rule pmf-add*[*OF M-def*], *auto*)
also have ... ≤ *?r1* + *?r2*
by (*intro add-mono f prob-degree-lt-1*)
finally show *?thesis* **by** *simp*
qed

private lemma *of-bool-square*: (*of-bool x*)^2 = ((*of-bool x*)::*real*)
by (*cases x*, *auto*)

private definition *Q* **where** *Q y* ω = *card* {*x* ∈ *set as*. *int* (*hash x* ω) < *y*}

private definition *m* **where** *m* = *card* (*set as*)

private lemma
assumes *a* ≥ *0*
assumes *a* ≤ *int p*
shows *exp-Q*: *expectation* (λω. *real* (*Q a* ω)) = *real m* ∗ (*of-int a*) / *p*
and *var-Q*: *variance* (λω. *real* (*Q a* ω)) ≤ *real m* ∗ (*of-int a*) / *p*
proof −
have *exp-single*: *expectation* (λω. *of-bool* (*int* (*hash x* ω) < *a*)) = *real-of-int a* /*real p*
if *a*:*x* ∈ *set as* **for** *x*
proof −
have *x-le-p*: *x* < *p* **using** *a as-lt-p* **by** *simp*
have *expectation* (λω. *of-bool* (*int* (*hash x* ω) < *a*)) = *expectation* (*indicat-real* {ω. *int* (*Frequency-Moment-0.hash p x* ω) < *a*})
by (*intro arg-cong2*[**where** *f*=$integral^L$] *ext*, *simp-all*)
also have ... = *prob* {ω. *hash x* ω ∈ {*k*. *int k* < *a*}}
by (*simp add*:*M-def*)
also have ... = *card* ({*k*. *int k* < *a*} ∩ {..<*p*}) / *real p*
by (*subst prob-range*, *simp-all add*: *x-le-p mod-ring-def*)

also have ... = *card* {..<*nat a*} / *real p*
using *assms* **by** (*intro arg-cong2*[**where** *f*=(/)] *arg-cong*[**where** *f*=*real*] *arg-cong*[**where** *f*=*card*])
(*auto simp add*:*set-eq-iff*)
also have ... = *real-of-int a*/*real p*
using *assms* **by** *simp*
finally show *expectation* (λω. *of-bool* (*int* (*hash x* ω) < *a*)) = *real-of-int a* /*real p*
by *simp*
qed

have *expectation*(λω. *real* (*Q a* ω)) = *expectation* (λω. (∑*x* ∈ *set as*. *of-bool* (*int* (*hash x* ω) < *a*)))
by (*simp add*:*Q-def Int-def*)
also have ... = (∑*x* ∈ *set as*. *expectation* (λω. *of-bool* (*int* (*hash x* ω) < *a*)))
by (*rule Bochner-Integration.integral-sum*, *simp*)
also have ... = (∑ *x* ∈ *set as*. *a* /*real p*)
by (*rule sum.cong*, *simp*, *subst exp-single*, *simp*, *simp*)
also have ... = *real m* ∗ *real-of-int a* / *real p*
by (*simp add*:*m-def*)
finally show *expectation* (λω. *real* (*Q a* ω)) = *real m* ∗ *real-of-int a* / *p* **by** *simp*

have *indep*: *J* ⊆ *set as* ⟹ *card J* = *2* ⟹ *indep-vars* (λ-. *borel*) (λ*i x*. *of-bool* (*int* (*hash i x*) < *a*)) *J* **for** *J*
using *as-subset-p mod-ring-carr*
by (*intro indep-vars-compose2*[**where** *Y*=λ*i x*. *of-bool* (*int x* < *a*) **and** *M′*=λ-. *discrete*]
k-wise-indep-vars-subset[*OF k-wise-indep*] *finite-subset*[*OF - finite-set*]) *auto*

have *rv*: ⋀*x*. *x* ∈ *set as* ⟹ *random-variable borel* (λω. *of-bool* (*int* (*hash x* ω) < *a*))
by (*simp add*:*M-def*)

have *variance* (λω. *real* (*Q a* ω)) = *variance* (λω. (∑*x* ∈ *set as*. *of-bool* (*int* (*hash x* ω) < *a*)))
by (*simp add*:*Q-def Int-def*)
also have ... = (∑*x* ∈ *set as*. *variance* (λω. *of-bool* (*int* (*hash x* ω) < *a*)))
by (*intro var-sum-pairwise-indep-2 indep rv*) *auto*
also have ... ≤ (∑ *x* ∈ *set as*. *a* / *real p*)
by (*rule sum-mono*, *simp add*: *variance-eq of-bool-square*, *simp add*: *exp-single*)
also have ... = *real m* ∗ *real-of-int a* /*real p*
by (*simp add*:*m-def*)
finally show *variance* (λω. *real* (*Q a* ω)) ≤ *real m* ∗ *real-of-int a* / *p*
by *simp*
qed

private lemma *t-bound*: *t* ≤ *81* / (*real-of-rat* δ)^2
proof −
have *t* ≤ *80* / (*real-of-rat* δ)^2 + *1* **using** *t-def t-gt-0* **by** *linarith*

also have ... ≤ *80* / (*real-of-rat* δ)^2 + *1* / (*real-of-rat* δ)^2
using *δ-range* **by** (*intro add-mono*, *simp*, *simp add*:*power-le-one*)
also have ... = *81* / (*real-of-rat* δ)^2 **by** *simp*
finally show *?thesis* **by** *simp*
qed

private lemma *t-r-bound*:
18 ∗ *40* ∗ (*real t*)^2 ∗ *2 powr* (−*real r*) ≤ *1*
proof −
have *720* ∗ (*real t*)^2 ∗ *2 powr* (−*real r*) ≤ *720* ∗ (*81* / (*real-of-rat* δ)^2)^2 ∗ *2 powr* (−*4* ∗ *log 2* (*1* / *real-of-rat* δ) − *23*)
using *r-bound t-bound* **by** (*intro mult-left-mono mult-mono power-mono powr-mono*, *auto*)

also have ... ≤ *720* ∗ (*81* / (*real-of-rat* δ)^2)^2 ∗ (*2 powr* (−*4* ∗ *log 2* (*1* / *real-of-rat* δ)) ∗ *2 powr* (−*23*))
using *δ-range* **by** (*intro mult-left-mono mult-mono power-mono add-mono*)
(*simp-all add*:*power-le-one powr-diff*)

also have ... = *720* ∗ (*81*^{2} / (*real-of-rat* δ)^*4*) ∗ (*2 powr* (*log 2* ((*real-of-rat* δ)^*4*)) ∗ *2 powr* (−*23*))
using *δ-range* **by** (*intro arg-cong2*[**where** *f*=(∗)])
(*simp-all add*:*power2-eq-square power4-eq-xxxx log-divide log-powr*[*symmetric*])

also have ... = *720* ∗ *81*^{2} ∗ *2 powr* (−*23*) **using** *δ-range* **by** *simp*

also have ... ≤ *1* **by** *simp*

finally show *?thesis* **by** *simp*
qed

private lemma *m-eq-F-0*: *real m* = *of-rat* (*F 0 as*)
by (*simp add*:*m-def F-def*)

private lemma *estimate′-bounds*:
prob {ω. *of-rat* δ ∗ *real-of-rat* (*F 0 as*) < |*estimate′* (*sketch-rv′* ω) − *of-rat* (*F 0 as*)|} ≤ *1*/*3*
proof (*cases card* (*set as*) ≥ *t*)
case *True*
define δ′ **where** δ′ = *3* ∗ *real-of-rat* δ / *4*
define *u* **where** *u* = ⌈*real t* ∗ *p* / (*m* ∗ (*1*+δ′))⌉
define *v* **where** *v* = ⌊*real t* ∗ *p* / (*m* ∗ (*1*−δ′))⌋

define *has-no-collision* **where**
has-no-collision = (λω. ∀ *x*∈ *set as*. ∀ *y* ∈ *set as*. (*tr-hash x* ω = *tr-hash y* ω ⟶ *x* = *y*) ∨ *tr-hash x* ω > *v*)

have *2 powr* (−*real r*) ≤ *2 powr* (−(*4* ∗ *log 2* (*1* / *real-of-rat* δ) + *23*))
using *r-bound* **by** (*intro powr-mono*, *linarith*, *simp*)

also have ... = *2 powr* (−*4* ∗ *log 2* (*1* /*real-of-rat* δ) −*23*)
by (*rule arg-cong2*[**where** *f*=(*powr*)], *auto simp add*:*algebra-simps*)
also have ... ≤ *2 powr* (−*1* ∗ *log 2* (*1* /*real-of-rat* δ) −*4*)
using *δ-range* **by** (*intro powr-mono diff-mono*, *auto*)
also have ... = *2 powr* (−*1* ∗ *log 2* (*1* /*real-of-rat* δ)) / *16*
by (*simp add*: *powr-diff*)
also have ... = *real-of-rat* δ / *16*
using *δ-range* **by** (*simp add*:*log-divide*)
also have ... < *real-of-rat* δ / *8*
using *δ-range* **by** (*subst pos-divide-less-eq*, *auto*)
finally have *r-le-δ*: *2 powr* (−*real r*) < *real-of-rat* δ / *8*
by *simp*

have *δ′-gt-0*: δ′ > *0* **using** *δ-range* **by** (*simp add*:*δ′-def*)
have δ′ < *3*/*4* **using** *δ-range* **by** (*simp add*:*δ′-def*)+
also have ... < *1* **by** *simp*
finally have *δ′-lt-1*: δ′ < *1* **by** *simp*

have *t* ≤ *81* / (*real-of-rat* δ)^2
using *t-bound* **by** *simp*
also have ... = (*81*∗*9*/*16*) / (δ′)^2
by (*simp add*:*δ′-def power2-eq-square*)
also have ... ≤ *46* / δ′^{2}
by (*intro divide-right-mono*, *simp*, *simp*)
finally have *t-le-δ′*: *t* ≤ *46*/ δ′^{2} **by** *simp*

have *80* ≤ (*real-of-rat* δ)^2 ∗ (*80* / (*real-of-rat* δ)^2) **using** *δ-range* **by** *simp*
also have ... ≤ (*real-of-rat* δ)^2 ∗ *t*
by (*intro mult-left-mono*, *simp add*:*t-def of-nat-ceiling*, *simp*)
finally have *80* ≤ (*real-of-rat* δ)^2 ∗ *t* **by** *simp*
hence *t-ge-δ′*: *45* ≤ *t* ∗ δ′ ∗ δ′ **by** (*simp add*:*δ′-def power2-eq-square*)

have *m* ≤ *card* {..<*n*} **unfolding** *m-def* **using** *as-range* **by** (*intro card-mono*, *auto*)
also have ... ≤ *p* **using** *n-le-p* **by** *simp*
finally have *m-le-p*: *m* ≤ *p* **by** *simp*

hence *t-le-m*: *t* ≤ *card* (*set as*) **using** *True* **by** *simp*
have *m-ge-0*: *real m* > *0* **using** *m-def True t-gt-0* **by** *simp*

have *v* ≤ *real t* ∗ *real p* / (*real m* ∗ (*1* − δ′)) **by** (*simp add*:*v-def*)

also have ... ≤ *real t* ∗ *real p* / (*real m* ∗ (*1*/*4*))
using *δ′-lt-1 m-ge-0 δ-range*
by (*intro divide-left-mono mult-left-mono mult-nonneg-nonneg mult-pos-pos*, *simp-all add*:*δ′-def*)

finally have *v-ubound*: *v* ≤ *4* ∗ *real t* ∗ *real p* / *real m* **by** (*simp add*:*algebra-simps*)

```
  have a-ge-1: u ≥ 1 using δ'-gt-0 p-gt-0 m-ge-0 t-gt-0
    by (auto intro!:mult-pos-pos divide-pos-pos simp add:u-def)
  hence a-ge-0: u ≥ 0 by simp
  have real m ∗ (1 − δ') < real m using δ'-gt-0 m-ge-0 by simp
  also have ... ≤ 1 ∗ real p using m-le-p by simp
  also have ... ≤ real t ∗ real p using t-gt-0 by (intro mult-right-mono, auto)
  finally have  real m ∗ (1 − δ') < real t ∗ real p by simp
  hence v-gt-0: v > 0 using mult-pos-pos m-ge-0 δ'-lt-1 by (simp add:v-def)
  hence v-ge-1: real-of-int v ≥ 1 by linarith

  have real t ≤ real m using True m-def by linarith
  also have ... < (1 + δ') ∗ real m using δ'-gt-0 m-ge-0 by force
  finally have a-le-p-aux: real t < (1 + δ') ∗ real m  by simp

  have u ≤ real t ∗ real p / (real m ∗ (1 + δ'))+1 by (simp add:u-def)
  also have ... < real p + 1
    using m-ge-0 δ'-gt-0 a-le-p-aux  a-le-p-aux p-gt-0
    by (simp add: pos-divide-less-eq ac-simps)
  finally have u ≤ real p
    by (metis int-less-real-le not-less of-int-le-iff of-int-of-nat-eq)
  hence u-le-p: u ≤ int p by linarith

  have prob {ω. Q u ω ≥ t} ≤ prob {ω ∈ Sigma-Algebra.space M. abs (real (Q u
ω) −
    expectation (λω. real (Q u ω))) ≥ 3 ∗ sqrt (m ∗ real-of-int u / p)}
  proof (rule pmf-mono[OF M-def])
    fix ω
    assume ω ∈ {ω. t ≤ Q u ω}
    hence t-le: t ≤ Q u ω by simp
    have real m ∗ real-of-int u / real p ≤ real m ∗ (real t ∗ real p / (real m ∗ (1 +
δ'))+1) / real p
     using m-ge-0 p-gt-0 by (intro divide-right-mono mult-left-mono, simp-all add:
u-def)
    also have ... = real m ∗ real t ∗ real p / (real m ∗ (1+δ') ∗ real p) + real m /
real p
      by (simp add:distrib-left add-divide-distrib)
    also have ... = real t / (1+δ') + real m / real p
      using p-gt-0 m-ge-0 by simp
    also have ... ≤ real t / (1+δ') + 1
      using m-le-p p-gt-0 by (intro add-mono, auto)
    finally have real m ∗ real-of-int u / real p ≤ real t / (1 + δ') + 1
      by simp

    hence 3 ∗ sqrt (real m ∗ of-int u / real p) + real m ∗ of-int u / real p ≤
      3 ∗ sqrt (t / (1+δ')+1)+(t/(1+δ')+1)
      by (intro add-mono mult-left-mono real-sqrt-le-mono, auto)
    also have ... ≤ 3 ∗ sqrt (real t+1) + ((t ∗ (1 − δ' / (1+δ'))) + 1)
      using δ'-gt-0 t-gt-0 by (intro add-mono mult-left-mono real-sqrt-le-mono)
        (simp-all add: pos-divide-le-eq left-diff-distrib)
```

also have ... = *3* ∗ *sqrt* (*real t*+*1*) + (*t* − δ′ ∗ *t* / (*1*+δ′)) + *1* **by** (*simp add*:*algebra-simps*)
also have ... ≤ *3* ∗ *sqrt* (*46* / δ′² + *1* / δ′²) + (*t* − δ′ ∗ *t*/*2*) + *1* / δ′
using δ′*-gt-0 t-gt-0* δ′*-lt-1 add-pos-pos t-le-*δ′
by (*intro add-mono mult-left-mono real-sqrt-le-mono add-mono*)
(*simp-all add*: *power-le-one pos-le-divide-eq*)
also have ... ≤ (*21* / δ′ + (*t* − *45* / (*2*∗δ′))) + *1* / δ′
using δ′*-gt-0 t-ge-*δ′ **by** (*intro add-mono*)
(*simp-all add*:*real-sqrt-divide divide-le-cancel real-le-lsqrt pos-divide-le-eq ac-simps*)
also have ... ≤ *t* **using** δ′*-gt-0* **by** *simp*
also have ... ≤ *Q u* ω **using** *t-le* **by** *simp*
finally have *3* ∗ *sqrt* (*real m* ∗ *of-int u* / *real p*) + *real m* ∗ *of-int u* / *real p* ≤ *Q u* ω
by *simp*
hence *3* ∗ *sqrt* (*real m* ∗ *real-of-int u* / *real p*) ≤ |*real* (*Q u* ω) − *expectation* (λω. *real* (*Q u* ω))|
using *a-ge-0 u-le-p True* **by** (*simp add*:*exp-Q abs-ge-iff*)

thus ω ∈ {ω ∈ *Sigma-Algebra.space M*. *3* ∗ *sqrt* (*real m* ∗ *real-of-int u* / *real p*) ≤
|*real* (*Q u* ω) − *expectation* (λω. *real* (*Q u* ω))|}
by (*simp add*: *M-def*)
qed
also have ... ≤ *variance* (λω. *real* (*Q u* ω)) / (*3* ∗ *sqrt* (*real m* ∗ *of-int u* / *real p*))²
using *a-ge-1 p-gt-0 m-ge-0*
by (*intro Chebyshev-inequality*, *simp add*:*M-def*, *auto*)

also have ... ≤ (*real m* ∗ *real-of-int u* / *real p*) / (*3* ∗ *sqrt* (*real m* ∗ *of-int u* / *real p*))²
using *a-ge-0 u-le-p* **by** (*intro divide-right-mono var-Q*, *auto*)

also have ... ≤ *1*/*9* **using** *a-ge-0* **by** *simp*

finally have *case-1*: *prob* {ω. *Q u* ω ≥ *t*} ≤ *1*/*9* **by** *simp*

have *case-2*: *prob* {ω. *Q v* ω < *t*} ≤ *1*/*9*
proof (*cases v* ≤ *p*)
case *True*
have *prob* {ω. *Q v* ω < *t*} ≤ *prob* {ω ∈ *Sigma-Algebra.space M*. *abs* (*real* (*Q v* ω) − *expectation* (λω. *real* (*Q v* ω)))
≥ *3* ∗ *sqrt* (*m* ∗ *real-of-int v* / *p*)}
proof (*rule pmf-mono*[*OF M-def*])
fix ω
assume ω ∈ *set-pmf* (*pmf-of-set space*)
have (*real t* + *3* ∗ *sqrt* (*real t* / (*1* − δ′))) ∗ (*1* − δ′) = *real t* − δ′ ∗ *t* + *3* ∗ ((*1*−δ′) ∗ *sqrt*(*real t* / (*1*−δ′)))
by (*simp add*:*algebra-simps*)

also have ... = *real t* − δ′ ∗ *t* + *3* ∗ *sqrt* ((*1*−δ′)^2 ∗ (*real t* / (*1*−δ′)))
using δ′*-lt-1* **by** (*subst real-sqrt-mult*, *simp*)

also have ... = *real t* − δ′ ∗ *t* + *3* ∗ *sqrt* (*real t* ∗ (*1*− δ′))
by (*simp add*:*power2-eq-square distrib-left*)

also have ... ≤ *real t* − *45*/ δ′ + *3* ∗ *sqrt* (*real t*)
using δ′*-gt-0 t-ge-*δ′ δ′*-lt-1* **by** (*intro add-mono mult-left-mono real-sqrt-le-mono*)
(*simp-all add*:*pos-divide-le-eq ac-simps left-diff-distrib power-le-one*)

also have ... ≤ *real t* − *45*/ δ′ + *3* ∗ *sqrt* (*46* / δ′^{2})
using *t-le-*δ′ δ′*-lt-1* δ′*-gt-0*
by (*intro add-mono mult-left-mono real-sqrt-le-mono*, *simp-all add*:*pos-divide-le-eq power-le-one*)

also have ... = *real t* + (*3* ∗ *sqrt*(*46*) − *45*)/ δ′
using δ′*-gt-0* **by** (*simp add*:*real-sqrt-divide diff-divide-distrib*)

also have ... ≤ *t*
using δ′*-gt-0* **by** (*simp add*:*pos-divide-le-eq real-le-lsqrt*)

finally have *aux*: (*real t* + *3* ∗ *sqrt* (*real t* / (*1* − δ′))) ∗ (*1* − δ′) ≤ *real t*
by *simp*

assume ω ∈ {ω. *Q v* ω < *t*}
hence *Q v* ω < *t* **by** *simp*

hence *real* (*Q v* ω) + *3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*)
≤ *real t* − *1* + *3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*)
using *m-le-p p-gt-0* **by** (*intro add-mono*, *auto simp add*: *algebra-simps add-divide-distrib*)

also have ... ≤ (*real t*−*1*) + *3* ∗ *sqrt* (*real m* ∗ (*real t* ∗ *real p* / (*real m* ∗ (*1*− δ′))) / *real p*)
by (*intro add-mono mult-left-mono real-sqrt-le-mono divide-right-mono*)
(*auto simp add*:*v-def*)

also have ... ≤ *real t* + *3* ∗ *sqrt*(*real t* / (*1*−δ′)) − *1*
using *m-ge-0 p-gt-0* **by** *simp*

also have ... ≤ *real t* / (*1*−δ′)−*1*
using δ′*-lt-1 aux* **by** (*simp add*: *pos-le-divide-eq*)
also have ... ≤ *real m* ∗ (*real t* ∗ *real p* / (*real m* ∗ (*1*−δ′))) / *real p* − *1*
using *p-gt-0 m-ge-0* **by** *simp*
also have ... ≤ *real m* ∗ (*real t* ∗ *real p* / (*real m* ∗ (*1*−δ′))) / *real p* − *real m* / *real p*
using *m-le-p p-gt-0*
by (*intro diff-mono*, *auto*)

also have ... = *real m* ∗ (*real t* ∗ *real p* / (*real m* ∗ (*1*−δ′))−*1*) / *real p*
by (*simp add*: *left-diff-distrib right-diff-distrib diff-divide-distrib*)
also have ... ≤ *real m* ∗ *real-of-int v* / *real p*
by (*intro divide-right-mono mult-left-mono*, *simp-all add*:*v-def*)

finally have *real* (*Q v ω*) + *3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*)
≤ *real m* ∗ *real-of-int v* / *real p* **by** *simp*

hence *3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*) ≤ |*real* (*Q v ω*) −*expectation* (λω. *real* (*Q v ω*))|
using *v-gt-0 True* **by** (*simp add*: *exp-Q abs-ge-iff*)

thus ω ∈ {ω∈ *Sigma-Algebra.space M*. *3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*) ≤
|*real* (*Q v ω*) − *expectation* (λω. *real* (*Q v ω*))|}
by (*simp add*:*M-def*)
qed
also have ... ≤ *variance* (λω. *real* (*Q v ω*)) / (*3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*))^2
using *v-gt-0 p-gt-0 m-ge-0*
by (*intro Chebyshev-inequality*, *simp add*:*M-def*, *auto*)

also have ... ≤ (*real m* ∗ *real-of-int v* / *real p*) / (*3* ∗ *sqrt* (*real m* ∗ *real-of-int v* / *real p*))^2
using *v-gt-0 True* **by** (*intro divide-right-mono var-Q*, *auto*)

also have ... = *1*/*9*
using *p-gt-0 v-gt-0 m-ge-0* **by** (*simp add*:*power2-eq-square*)

finally show *?thesis* **by** *simp*
next
case *False*
have *prob* {ω. *Q v ω* < *t*} ≤ *prob* {ω. *False*}
proof (*rule pmf-mono*[*OF M-def*])
fix ω
assume *a*:ω ∈ {ω. *Q v ω* < *t*}
assume ω ∈ *set-pmf* (*pmf-of-set space*)
hence *b*:⋀*x*. *x* < *p* ⟹ *hash x ω* < *p*
using *hash-range mod-ring-carr* **by** (*simp add*:*M-def measure-pmf-inverse*)
have *t* ≤ *card* (*set as*) **using** *True* **by** *simp*
also have ... ≤ *Q v ω*
unfolding *Q-def* **using** *b False as-lt-p* **by** (*intro card-mono subsetI*, *simp*, *force*)
also have ... < *t* **using** *a* **by** *simp*
finally have *False* **by** *auto*
thus ω ∈ {ω. *False*} **by** *simp*
qed
also have ... = *0* **by** *auto*
finally show *?thesis* **by** *simp*

qed

have *prob* {ω. ¬*has-no-collision* ω} ≤
prob {ω. ∃ *x* ∈ *set as*. ∃ *y* ∈ *set as*. *x* ≠ *y* ∧ *tr-hash x* ω ≤ *real-of-int v* ∧ *tr-hash x* ω = *tr-hash y* ω}
by (*rule pmf-mono*[*OF M-def*]) (*simp add:has-no-collision-def M-def*, *force*)

also have ... ≤ (*5/2*) ∗ (*real* (*card* (*set as*)))^2 ∗ (*real-of-int v*)^2 ∗ *2 powr* − *real r* / (*real p*)^2 + *1* / *real p*
using *collision-prob v-ge-1* **by** *blast*

also have ... ≤ (*5/2*) ∗ (*real m*)^2 ∗ (*real-of-int v*)^2 ∗ *2 powr* − *real r* / (*real p*)^2 + *1* / *real p*
by (*intro divide-right-mono add-mono mult-right-mono mult-mono power-mono*, *simp-all add:m-def*)

also have ... ≤ (*5/2*) ∗ (*real m*)^2 ∗ (*4* ∗ *real t* ∗ *real p* / *real m*)^2 ∗ (*2 powr* − *real r*) / (*real p*)^2 + *1* / *real p*
using *v-def v-ge-1 v-ubound*
by (*intro add-mono divide-right-mono mult-right-mono mult-left-mono*, *auto*)

also have ... = *40* ∗ (*real t*)^2 ∗ (*2 powr* −*real r*) + *1* / *real p*
using *p-gt-0 m-ge-0 t-gt-0* **by** (*simp add:algebra-simps power2-eq-square*)

also have ... ≤ *1/18* + *1/18*
using *t-r-bound p-ge-18* **by** (*intro add-mono*, *simp-all add*: *pos-le-divide-eq*)

also have ... = *1/9* **by** *simp*

finally have *case-3*: *prob* {ω. ¬*has-no-collision* ω} ≤ *1/9* **by** *simp*

have *prob* {ω. *real-of-rat* δ ∗ *of-rat* (*F 0 as*) < |*estimate'* (*sketch-rv'* ω) − *of-rat* (*F 0 as*)|} ≤
prob {ω. *Q u* ω ≥ *t* ∨ *Q v* ω < *t* ∨ ¬(*has-no-collision* ω)}
proof (*rule pmf-mono*[*OF M-def*], *rule ccontr*)
fix ω
assume ω ∈ *set-pmf* (*pmf-of-set space*)
assume ω ∈ {ω. *real-of-rat* δ ∗ *real-of-rat* (*F 0 as*) < |*estimate'* (*sketch-rv'* ω) − *real-of-rat* (*F 0 as*)|}
hence *est*: *real-of-rat* δ ∗ *real-of-rat* (*F 0 as*) < |*estimate'* (*sketch-rv'* ω) − *real-of-rat* (*F 0 as*)| **by** *simp*
assume ω ∉ {ω. *t* ≤ *Q u* ω ∨ *Q v* ω < *t* ∨ ¬ *has-no-collision* ω}
hence ¬(*t* ≤ *Q u* ω ∨ *Q v* ω < *t* ∨ ¬ *has-no-collision* ω) **by** *simp*
hence *lb*: *Q u* ω < *t* **and** *ub*: *Q v* ω ≥ *t* **and** *no-col*: *has-no-collision* ω **by** *simp+*

define *y* **where** *y* = *nth-mset* (*t*−*1*) {#*int* (*hash x* ω). *x* ∈# *mset-set* (*set as*)#}
define *y'* **where** *y'* = *nth-mset* (*t*−*1*) {#*tr-hash x* ω. *x* ∈# *mset-set* (*set as*)#}

```
  have rank-t-lb: u ≤ y
    unfolding y-def using True t-gt-0 lb
      by (intro nth-mset-bound-left, simp-all add:count-less-def swap-filter-image
Q-def)

  have rank-t-ub: y ≤ v − 1
    unfolding y-def using True t-gt-0 ub
   by (intro nth-mset-bound-right, simp-all add:Q-def swap-filter-image count-le-def)

  have y-ge-0: real-of-int y ≥ 0 using rank-t-lb a-ge-0 by linarith

  have mono (λx. truncate-down r (real-of-int x))
    by (metis truncate-down-mono mono-def of-int-le-iff)
  hence y′-eq: y′ = truncate-down r y
    unfolding y-def y′-def  using True t-gt-0
      by (subst nth-mset-commute-mono[where f=(λx. truncate-down r (of-int
x))])
      (simp-all add: multiset.map-comp comp-def tr-hash-def)

  have real-of-int u ∗ (1 − 2 powr −real r) ≤ real-of-int y ∗ (1 − 2 powr (−real
r))
    using rank-t-lb of-int-le-iff two-pow-r-le-1
    by (intro mult-right-mono, auto)
  also have ... ≤ y′
    using y′-eq truncate-down-pos[OF y-ge-0] by simp
  finally have rank-t-lb′: u ∗ (1 − 2 powr −real r) ≤ y′ by simp

  have y′ ≤ real-of-int y
    by (subst y′-eq, rule truncate-down-le, simp)
  also have ... ≤ real-of-int (v−1)
    using rank-t-ub of-int-le-iff by blast
  finally have rank-t-ub′: y′ ≤ v−1
    by simp

  have 0 < u ∗ (1−2 powr −real r)
    using a-ge-1 two-pow-r-le-1 by (intro mult-pos-pos, auto)
  hence y′-pos: y′ > 0 using rank-t-lb′ by linarith

  have no-col′: ⋀x. x ≤ y′ ⟹ count {#tr-hash x ω. x ∈# mset-set (set as)#}
x ≤ 1
    using  rank-t-ub′ no-col
   by (simp add:vimage-def card-le-Suc0-iff-eq count-image-mset has-no-collision-def)
force

  have h-1: Max (sketch-rv′ ω) = y′
    using True t-gt-0 no-col′
    by (simp add:sketch-rv′-def y′-def nth-mset-max)
```

have *card* (*sketch-rv′ ω*) = *card* (*least* ((*t*−*1*)+*1*) (*set-mset* {#*tr-hash x ω*. *x* ∈# *mset-set* (*set as*)#}))
using *t-gt-0* **by** (*simp add*:*sketch-rv′-def*)
also have ... = (*t*−*1*) +*1*
using *True t-gt-0 no-col′* **by** (*intro nth-mset-max*(*2*), *simp-all add*:*y′-def*)
also have ... = *t* **using** *t-gt-0* **by** *simp*
finally have *card* (*sketch-rv′ ω*) = *t* **by** *simp*
hence *h-3*: *estimate′* (*sketch-rv′ ω*) = *real t* ∗ *real p* / *y′*
using *h-1* **by** (*simp add*:*estimate′-def*)

have (*real t*) ∗ *real p* ≤ (*1* + *δ′*) ∗ *real m* ∗ ((*real t*) ∗ *real p* / (*real m* ∗ (*1* + *δ′*)))
using *δ′-lt-1 m-def True t-gt-0 δ′-gt-0* **by** *auto*
also have ... ≤ (*1*+*δ′*) ∗ *m* ∗ *u*
using *δ′-gt-0* **by** (*intro mult-left-mono*, *simp-all add*:*u-def*)
also have ... < ((*1* + *real-of-rat δ*)∗(*1*−*real-of-rat δ*/*8*)) ∗ *m* ∗ *u*
using *True m-def t-gt-0 a-ge-1 δ-range*
by (*intro mult-strict-right-mono*, *auto simp add*:*δ′-def right-diff-distrib*)
also have ... ≤ ((*1* + *real-of-rat δ*)∗(*1*−*2 powr* (−*r*))) ∗ *m* ∗ *u*
using *r-le-δ δ-range a-ge-0* **by** (*intro mult-right-mono mult-left-mono*, *auto*)
also have ... = (*1* + *real-of-rat δ*) ∗ *m* ∗ (*u* ∗ (*1*−*2 powr* −*real r*))
by *simp*
also have ... ≤ (*1* + *real-of-rat δ*) ∗ *m* ∗ *y′*
using *δ-range* **by** (*intro mult-left-mono rank-t-lb′*, *simp*)
finally have *real t* ∗ *real p* < (*1* + *real-of-rat δ*) ∗ *m* ∗ *y′* **by** *simp*
hence *f-1*: *estimate′* (*sketch-rv′ ω*) < (*1* + *real-of-rat δ*) ∗ *m*
using *y′-pos* **by** (*simp add*: *h-3 pos-divide-less-eq*)

have (*1* − *real-of-rat δ*) ∗ *m* ∗ *y′* ≤ (*1* − *real-of-rat δ*) ∗ *m* ∗ *v*
using *δ-range rank-t-ub′ y′-pos* **by** (*intro mult-mono rank-t-ub′*, *simp-all*)
also have ... = (*1*−*real-of-rat δ*) ∗ (*real m* ∗ *v*)
by *simp*
also have ... < (*1*−*δ′*) ∗ (*real m* ∗ *v*)
using *δ-range m-ge-0 v-ge-1*
by (*intro mult-strict-right-mono mult-pos-pos*, *simp-all add*:*δ′-def*)
also have ... ≤ (*1*−*δ′*) ∗ (*real m* ∗ (*real t* ∗ *real p* / (*real m* ∗ (*1*−*δ′*))))
using *δ′-gt-0 δ′-lt-1* **by** (*intro mult-left-mono*, *auto simp add*:*v-def*)
also have ... = *real t* ∗ *real p*
using *δ′-gt-0 δ′-lt-1 t-gt-0 p-gt-0 m-ge-0* **by** *auto*
finally have (*1* − *real-of-rat δ*) ∗ *m* ∗ *y′* < *real t* ∗ *real p* **by** *simp*
hence *f-2*: *estimate′* (*sketch-rv′ ω*) > (*1* − *real-of-rat δ*) ∗ *m*
using *y′-pos* **by** (*simp add*: *h-3 pos-less-divide-eq*)

have *abs* (*estimate′* (*sketch-rv′ ω*) − *real-of-rat* (*F 0 as*)) < *real-of-rat δ* ∗ (*real-of-rat* (*F 0 as*))
using *f-1 f-2* **by** (*simp add*:*abs-less-iff algebra-simps m-eq-F-0*)
thus *False* **using** *est* **by** *linarith*
qed
also have ... ≤ *1*/*9* + (*1*/*9* + *1*/*9*)

by (*intro pmf-add-2*[*OF M-def*] *case-1 case-2 case-3*)
also have ... = *1/3* **by** *simp*
finally show *?thesis* **by** *simp*
next
case *False*
have *prob* {ω. *real-of-rat* δ $*$ *of-rat* (*F 0 as*) < |*estimate'* (*sketch-rv'* ω) − *of-rat* (*F 0 as*)|} ≤
prob {ω. ∃ *x* ∈ *set as*. ∃ *y* ∈ *set as*. *x* ≠ *y* ∧ *tr-hash x* ω ≤ *real p* ∧ *tr-hash x* ω = *tr-hash y* ω}
proof (*rule pmf-mono*[*OF M-def*])
fix ω
assume *a*:ω ∈ {ω. *real-of-rat* δ $*$ *real-of-rat* (*F 0 as*) < |*estimate'* (*sketch-rv'* ω) − *real-of-rat* (*F 0 as*)|}
assume *b*:ω ∈ *set-pmf* (*pmf-of-set space*)
have *c*: *card* (*set as*) < *t* **using** *False* **by** *auto*
hence *card* ((λ*x*. *tr-hash x* ω) ' *set as*) < *t*
using *card-image-le order-le-less-trans* **by** *blast*
hence *d*:*card* (*sketch-rv'* ω) = *card* ((λ*x*. *tr-hash x* ω) ' (*set as*))
by (*simp add*:*sketch-rv'-def card-least*)
have *card* (*sketch-rv'* ω) < *t*
by (*metis List.finite-set c d card-image-le order-le-less-trans*)
hence *estimate'* (*sketch-rv'* ω) = *card* (*sketch-rv'* ω) **by** (*simp add*:*estimate'-def*)
hence *card* (*sketch-rv'* ω) ≠ *real-of-rat* (*F 0 as*)
using *a* δ*-range* **by** *simp*
(*metis abs-zero cancel-comm-monoid-add-class.diff-cancel of-nat-less-0-iff pos-prod-lt zero-less-of-rat-iff*)
hence *card* (*sketch-rv'* ω) ≠ *card* (*set as*)
using *m-def m-eq-F-0* **by** *linarith*
hence ¬*inj-on* (λ*x*. *tr-hash x* ω) (*set as*)
using *card-image d* **by** *auto*
moreover have *tr-hash x* ω ≤ *real p* **if** *a*:*x* ∈ *set as* **for** *x*
proof −
have *hash x* ω < *p*
using *hash-range as-lt-p a b* **by** (*simp add*:*mod-ring-carr M-def*)
thus *tr-hash x* ω ≤ *real p* **using** *truncate-down-le* **by** (*simp add*:*tr-hash-def*)
qed
ultimately show ω ∈ {ω. ∃ *x* ∈ *set as*. ∃ *y* ∈ *set as*. *x* ≠ *y* ∧ *tr-hash x* ω ≤ *real p* ∧ *tr-hash x* ω = *tr-hash y* ω}
by (*simp add*:*inj-on-def*, *blast*)
qed
also have ... ≤ (*5/2*) $*$ (*real* (*card* (*set as*)))^2 $*$ (*real p*)^2 $*$ *2 powr* − *real r* / (*real p*)^2 + *1* / *real p*
using *p-gt-0* **by** (*intro collision-prob*, *auto*)
also have ... = (*5/2*) $*$ (*real* (*card* (*set as*)))^2 $*$ *2 powr* (− *real r*) + *1* / *real p*
using *p-gt-0* **by** (*simp add*:*ac-simps power2-eq-square*)
also have ... ≤ (*5/2*) $*$ (*real t*)^2 $*$ *2 powr* (−*real r*) + *1* / *real p*
using *False* **by** (*intro add-mono mult-right-mono mult-left-mono power-mono*, *auto*)
also have ... ≤ *1/6* + *1/6*

using *t-r-bound p-ge-18* **by** (*intro add-mono*, *simp-all*)
also have ... ≤ *1/3* **by** *simp*
finally show *?thesis* **by** *simp*
qed

private lemma *median-bounds*:
$\mathcal{P}(\omega$ *in measure-pmf* Ω_0. |*median s* (λ*i*. *estimate* (*sketch-rv* (ω *i*))) − *F 0 as*| ≤ δ ∗ *F 0 as*) ≥ *1* − *real-of-rat* ε
proof −
have *strict-mono-on A real-of-float* **for** *A* **by** (*meson less-float.rep-eq strict-mono-onI*)
hence *real-g-2*: ⋀ω. *sketch-rv′* ω = *real-of-float* ‘ *sketch-rv* ω
by (*simp add*: *sketch-rv′-def sketch-rv-def tr-hash-def least-mono-commute image-comp*)

moreover have *inj-on real-of-float A* **for** *A*
using *real-of-float-inject* **by** (*simp add*:*inj-on-def*)
ultimately have *card-eq*: ⋀ω. *card* (*sketch-rv* ω) = *card* (*sketch-rv′* ω)
using *real-g-2* **by** (*auto intro*!: *card-image*[*symmetric*])

have *Max* (*sketch-rv′* ω) = *real-of-float* (*Max* (*sketch-rv* ω)) **if** *a*:*card* (*sketch-rv′* ω) ≥ *t* **for** ω
proof −
have *mono real-of-float*
using *less-eq-float.rep-eq mono-def* **by** *blast*
moreover have *finite* (*sketch-rv* ω)
by (*simp add*:*sketch-rv-def least-def*)
moreover have *sketch-rv* ω ≠ {}
using *card-eq*[*symmetric*] *card-gt-0-iff t-gt-0 a* **by** (*simp*, *force*)
ultimately show *?thesis*
by (*subst mono-Max-commute*[**where** *f*=*real-of-float*], *simp-all add*:*real-g-2*)
qed
hence *real-g*: ⋀ω. *estimate′* (*sketch-rv′* ω) = *real-of-rat* (*estimate* (*sketch-rv* ω))
by (*simp add*:*estimate-def estimate′-def card-eq of-rat-divide of-rat-mult of-rat-add real-of-rat-of-float*)

have *indep*: *prob-space.indep-vars* (*measure-pmf* Ω_0) (λ-. *borel*) (λ*i* ω. *estimate′* (*sketch-rv′* (ω *i*))) {*0*..<*s*}
unfolding Ω_0*-def*
by (*rule indep-vars-restrict-intro′*, *auto simp add*:*restrict-dfl-def lessThan-atLeast0*)

moreover have − (*18* ∗ *ln* (*real-of-rat* ε)) ≤ *real s*
using *of-nat-ceiling* **by** (*simp add*:*s-def*) *blast*

moreover have *i* < *s* ⟹ *measure* Ω_0 {ω. *of-rat* δ ∗ *of-rat* (*F 0 as*) < |*estimate′* (*sketch-rv′* (ω *i*)) − *of-rat* (*F 0 as*)|} ≤ *1/3*
for *i*
using *estimate′-bounds* **unfolding** Ω_0*-def M-def*
by (*subst prob-prod-pmf-slice*, *simp-all*)

ultimately have *1−real-of-rat* $\varepsilon \le \mathcal{P}(\omega$ *in measure-pmf* Ω_0.
|*median s* (λi. *estimate′* (*sketch-rv′* (ω *i*))) − *real-of-rat* (*F 0 as*)| ≤ *real-of-rat* δ ∗ *real-of-rat* (*F 0 as*))
using ε*-range prob-space-measure-pmf*
by (*intro prob-space.median-bound-2*) *auto*
also have ... = $\mathcal{P}(\omega$ *in measure-pmf* Ω_0.
|*median s* (λi. *estimate* (*sketch-rv* (ω *i*))) − *F 0 as*| ≤ δ ∗ *F 0 as*)
using *s-gt-0 median-rat*[*symmetric*] *real-g* **by** (*intro arg-cong2*[**where** *f*=*measure*])

(*simp-all add*:*of-rat-diff*[*symmetric*] *of-rat-mult*[*symmetric*] *of-rat-less-eq*)
finally show $\mathcal{P}(\omega$ *in measure-pmf* Ω_0. |*median s* (λi. *estimate* (*sketch-rv* (ω *i*))) − *F 0 as*| ≤ δ ∗ *F 0 as*) ≥ *1−real-of-rat* ε
by *blast*
qed

lemma *f0-alg-correct′*:
$\mathcal{P}(\omega$ *in measure-pmf result*. |ω − *F 0 as*| ≤ δ ∗ *F 0 as*) ≥ *1* − *of-rat* ε
proof −
have *f0-result-elim*: ⋀*x*. *f0-result* (*s*, *t*, *p*, *r*, *x*, λi∈{..<*s*}. *sketch-rv* (*x i*)) =
return-pmf (*median s* (λi. *estimate* (*sketch-rv* (*x i*))))
by (*simp add*:*estimate-def*, *rule median-cong*, *simp*)

have *result* = *map-pmf* (λx. (*s*, *t*, *p*, *r*, *x*, λi∈{..<*s*}. *sketch-rv* (*x i*))) Ω_0 ⤜ *f0-result*
by (*subst result-def*, *subst f0-alg-sketch*, *simp*)
also have ... = Ω_0 ⤜ (λx. *return-pmf* (*s*, *t*, *p*, *r*, *x*, λi∈{..<*s*}. *sketch-rv* (*x i*))) ⤜ *f0-result*
by (*simp add*:*t-def p-def r-def s-def map-pmf-def*)
also have ... = Ω_0 ⤜ (λx. *return-pmf* (*median s* (λi. *estimate* (*sketch-rv* (*x i*)))))
by (*subst bind-assoc-pmf*, *subst bind-return-pmf*, *subst f0-result-elim*) *simp*
finally have *a*:*result* = Ω_0 ⤜ (λx. *return-pmf* (*median s* (λi. *estimate* (*sketch-rv* (*x i*)))))
by *simp*

show *?thesis*
using *median-bounds* **by** (*simp add*: *a map-pmf-def*[*symmetric*])
qed

private lemma *f-subset*:
assumes *g ‘ A* ⊆ *h ‘ B*
shows (λx. *f* (*g x*)) *‘ A* ⊆ (λx. *f* (*h x*)) *‘ B*
using *assms* **by** *auto*

lemma *f0-exact-space-usage′*:
defines Ω ≡ *fold* (λa *state*. *state* ⤜ *f0-update a*) *as* (*f0-init* δ ε *n*)
shows *AE* ω *in* Ω. *bit-count* (*encode-f0-state* ω) ≤ *f0-space-usage* (*n*, ε, δ)
proof −

have *log-2-4*: *log 2 4* = *2*
by (*metis log2-of-power-eq mult-2 numeral-Bit0 of-nat-numeral power2-eq-square*)

have *a*: *bit-count* (F_e (*float-of* (*truncate-down r y*))) ≤
ereal (*12* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*n*+*13*))) **if** *a-1*:*y* ∈ {..<*p*} **for** *y*
proof (*cases y* ≥ *1*)
case *True*

have *aux-1*: *0* < *2* + *log 2* (*real y*)
using *True* **by** (*intro add-pos-nonneg*, *auto*)
have *aux-2*: *0* < *2* + *log 2* (*real p*)
using *p-gt-1* **by** (*intro add-pos-nonneg*, *auto*)

have *bit-count* (F_e (*float-of* (*truncate-down r y*))) ≤
ereal (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*2* + |*log 2* |*real y*||))
by (*rule truncate-float-bit-count*)
also have ... = *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*2* + (*log 2* (*real y*))))
using *True* **by** *simp*
also have ... ≤ *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*2* + *log 2 p*))
using *aux-1 aux-2 True p-gt-0 a-1* **by** *simp*
also have ... ≤ *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2 4* + *log 2* (*2* ∗ *n* +
40)))
using *log-2-4 p-le-n p-gt-0*
by (*intro ereal-mono add-mono mult-left-mono log-mono of-nat-mono add-pos-nonneg*,
auto)
also have ... = *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*8* ∗ *n* + *160*)))
by (*simp add*:*log-mult*[*symmetric*])
also have ... ≤ *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* ((*n*+*13*) *powr 2*)))
by (*intro ereal-mono add-mono mult-left-mono log-mono of-nat-mono add-pos-nonneg*)
(*auto simp add*:*power2-eq-square algebra-simps*)
also have ... = *ereal* (*10* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2 4* ∗ *log 2* (*n* + *13*)))
by (*subst log-powr*, *simp-all add*:*log-2-4*)
also have ... = *ereal* (*12* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*n* + *13*)))
by (*subst log-mult*, *simp-all add*:*log-2-4*)
finally show *?thesis* **by** *simp*
next
case *False*
hence *y* = *0* **using** *a-1* **by** *simp*
then show *?thesis* **by** (*simp add*:*float-bit-count-zero*)
qed

have *bit-count* (*encode-f0-state* (*s*, *t*, *p*, *r*, *x*, λ*i*∈{..<*s*}. *sketch-rv* (*x i*))) ≤
f0-space-usage (*n*, ε, δ) **if** *b*: *x* ∈ {..<*s*} $\rightarrow_E$ *space* **for** *x*
proof −
have *c*: *x* ∈ *extensional* {..<*s*} **using** *b* **by** (*simp add*:*PiE-def*)

have *d*: *sketch-rv* (*x y*) ⊆ (λ*k*. *float-of* (*truncate-down r k*)) ' {..<*p*}
if *d-1*: *y* < *s* **for** *y*
proof −

have *sketch-rv* (*x y*) ⊆ (λ*xa*. *float-of* (*truncate-down r* (*hash xa* (*x y*)))) ' *set as*
using *least-subset* **by** (*auto simp add*:*sketch-rv-def tr-hash-def*)
also have ... ⊆ (λ*k*. *float-of* (*truncate-down r* (*real k*))) ' {..<*p*}
using *b hash-range as-lt-p d-1*
by (*intro f-subset*[**where** *f*=λ*x*. *float-of* (*truncate-down r* (*real x*))] *image-subsetI*)
(*simp add*: *PiE-iff mod-ring-carr*)
finally show *?thesis*
by *simp*
qed

have ⋀*y*. *y* < *s* ⟹ *finite* (*sketch-rv* (*x y*))
unfolding *sketch-rv-def* **by** (*rule finite-subset*[*OF least-subset*], *simp*)
moreover have *card-sketch*: ⋀*y*. *y* < *s* ⟹ *card* (*sketch-rv* (*x y*)) ≤ *t*
by (*simp add*:*sketch-rv-def card-least*)
moreover have ⋀*y z*. *y* < *s* ⟹ *z* ∈ *sketch-rv* (*x y*) ⟹
bit-count (F_e *z*) ≤ *ereal* (*12* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real n* + *13*)))
using *a d* **by** *auto*
ultimately have *e*: ⋀*y*. *y* < *s* ⟹ *bit-count* (S_e F_e (*sketch-rv* (*x y*)))
≤ *ereal* (*real t*) ∗ (*ereal* (*12* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real* (*n* + *13*)))) + *1*) + *1*
using *float-encoding* **by** (*intro set-bit-count-est*, *auto*)

have *f*: ⋀*y*. *y* < *s* ⟹ *bit-count* (P_e *p 2* (*x y*)) ≤ *ereal* (*real 2* ∗ (*log 2* (*real p*) + *1*))
using *p-gt-1 b*
by (*intro bounded-degree-polynomial-bit-count*) (*simp-all add*:*space-def PiE-def Pi-def*)

have *bit-count* (*encode-f0-state* (*s*, *t*, *p*, *r*, *x*, λ*i*∈{..<*s*}. *sketch-rv* (*x i*))) =
bit-count (N_e *s*) + *bit-count* (N_e *t*) + *bit-count* (N_e *p*) + *bit-count* (N_e *r*) +
bit-count (([*0*..<*s*] $\rightarrow_e$ P_e *p 2*) *x*) +
bit-count (([*0*..<*s*] $\rightarrow_e$ S_e F_e) (λ*i*∈{..<*s*}. *sketch-rv* (*x i*)))
by (*simp add*:*encode-f0-state-def dependent-bit-count lessThan-atLeast0*
s-def[*symmetric*] *t-def*[*symmetric*] *p-def*[*symmetric*] *r-def*[*symmetric*] *ac-simps*)
also have ... ≤ *ereal* (*2*∗ *log 2* (*real s* + *1*) + *1*) + *ereal* (*2*∗ *log 2* (*real t* + *1*) + *1*)
+ *ereal* (*2*∗ *log 2* (*real p* + *1*) + *1*) + *ereal* (*2* ∗ *log 2* (*real r* + *1*) + *1*)
+ (*ereal* (*real s*) ∗ (*ereal* (*real 2* ∗ (*log 2* (*real p*) + *1*))))
+ (*ereal* (*real s*) ∗ ((*ereal* (*real t*) ∗
(*ereal* (*12* + *4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real* (*n* + *13*)))) + *1*) + *1*)))
using *c e f*
by (*intro add-mono exp-golomb-bit-count fun-bit-count-est*[**where** *xs*=[*0*..<*s*], *simplified*])
(*simp-all add*:*lessThan-atLeast0*)
also have ... = *ereal* (*4* + *2* ∗ *log 2* (*real s* + *1*) + *2* ∗ *log 2* (*real t* + *1*) +
2 ∗ *log 2* (*real p* + *1*) + *2* ∗ *log 2* (*real r* + *1*) + *real s* ∗ (*3* + *2* ∗ *log 2* (*real p*) +

real t ∗ (*13* + (*4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real n* + *13*))))))
by (*simp add*:*algebra-simps*)
also have ... ≤ *ereal* (*4* + *2* ∗ *log 2* (*real s* + *1*) + *2* ∗ *log 2* (*real t* + *1*) +
2 ∗ *log 2* (*2* ∗ (*21* + *real n*)) + *2* ∗ *log 2* (*real r* + *1*) + *real s* ∗ (*3* + *2* ∗ *log 2* (*2* ∗ (*21* + *real n*)) +
real t ∗ (*13* + (*4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real n* + *13*))))))
using *p-le-n p-gt-0*
by (*intro ereal-mono add-mono mult-left-mono*, *auto*)
also have ... = *ereal* (*6* + *2* ∗ *log 2* (*real s* + *1*) + *2* ∗ *log 2* (*real t* + *1*) +
2 ∗ *log 2* (*21* + *real n*) + *2* ∗ *log 2* (*real r* + *1*) + *real s* ∗ (*5* + *2* ∗ *log 2* (*21* + *real n*) +
real t ∗ (*13* + (*4* ∗ *real r* + *2* ∗ *log 2* (*log 2* (*real n* + *13*))))))
by (*subst* (*1 2*) *log-mult*, *auto*)
also have ... ≤ *f0-space-usage* (*n*, *ε*, *δ*)
by (*simp add*:*s-def*[*symmetric*] *r-def*[*symmetric*] *t-def*[*symmetric*] *Let-def*)
(*simp add*:*algebra-simps*)
finally show *bit-count* (*encode-f0-state* (*s*, *t*, *p*, *r*, *x*, *λi*∈{..<*s*}. *sketch-rv* (*x i*))) ≤
f0-space-usage (*n*, *ε*, *δ*) **by** *simp*
qed
hence ⋀*x*. *x* ∈ *set-pmf* Ω_0 ⟹
bit-count (*encode-f0-state* (*s*, *t*, *p*, *r*, *x*, *λi*∈{..<*s*}. *sketch-rv* (*x i*))) ≤ *ereal* (*f0-space-usage* (*n*, *ε*, *δ*))
by (*simp add*:Ω_0*-def set-prod-pmf del*:*f0-space-usage.simps*)
hence ⋀*y*. *y* ∈ *set-pmf* Ω ⟹ *bit-count* (*encode-f0-state y*) ≤ *ereal* (*f0-space-usage* (*n*, *ε*, *δ*))
by (*simp add*: *Ω-def f0-alg-sketch del*:*f0-space-usage.simps f0-init.simps*)
(*metis* (*no-types*, *lifting*) *image-iff pmf.set-map*)
thus *?thesis*
by (*simp add*: *AE-measure-pmf-iff del*:*f0-space-usage.simps*)
qed

end

Main results of this section:

theorem *f0-alg-correct*:
assumes *ε* ∈ {*0*<..<*1*}
assumes *δ* ∈ {*0*<..<*1*}
assumes *set as* ⊆ {..<*n*}
defines Ω ≡ *fold* (*λa state*. *state* ≫= *f0-update a*) *as* (*f0-init δ ε n*) ≫= *f0-result*
shows 𝒫(*ω in measure-pmf* Ω. |*ω* − *F 0 as*| ≤ *δ* ∗ *F 0 as*) ≥ *1* − *of-rat ε*
using *f0-alg-correct'*[*OF assms*(*1*−*3*)] **unfolding** *Ω-def* **by** *blast*

theorem *f0-exact-space-usage*:
assumes *ε* ∈ {*0*<..<*1*}
assumes *δ* ∈ {*0*<..<*1*}
assumes *set as* ⊆ {..<*n*}
defines Ω ≡ *fold* (*λa state*. *state* ≫= *f0-update a*) *as* (*f0-init δ ε n*)
shows *AE ω in* Ω. *bit-count* (*encode-f0-state ω*) ≤ *f0-space-usage* (*n*, *ε*, *δ*)

using *f0-exact-space-usage′*[*OF assms*(*1−3*)] **unfolding** Ω*-def* **by** *blast*

theorem *f0-asymptotic-space-complexity*:
f0-space-usage ∈ *O*[*at-top* $\times_F$ *at-right 0* $\times_F$ *at-right 0*](λ(*n*, ε, δ). *ln* (*1* / *of-rat* ε) ∗
(*ln* (*real n*) + *1* / (*of-rat* δ)^2 ∗ (*ln* (*ln* (*real n*)) + *ln* (*1* / *of-rat* δ))))
(**is** - ∈ *O*[*?F*](*?rhs*))
proof −
define *n-of* :: *nat* × *rat* × *rat* ⇒ *nat* **where** *n-of* = (λ(*n*, ε, δ). *n*)
define ε*-of* :: *nat* × *rat* × *rat* ⇒ *rat* **where** ε*-of* = (λ(*n*, ε, δ). ε)
define δ*-of* :: *nat* × *rat* × *rat* ⇒ *rat* **where** δ*-of* = (λ(*n*, ε, δ). δ)
define *t-of* **where** *t-of* = (λ*x*. *nat* ⌈*80* / (*real-of-rat* (δ*-of x*))^2⌉)
define *s-of* **where** *s-of* = (λ*x*. *nat* ⌈−(*18* ∗ *ln* (*real-of-rat* (ε*-of x*)))⌉)
define *r-of* **where** *r-of* = (λ*x*. *nat* (*4* ∗ ⌈*log 2* (*1* / *real-of-rat* (δ*-of x*))⌉ + *23*))

define *g* **where** *g* = (λ*x*. *ln* (*1* / *of-rat* (ε*-of x*)) ∗ (*ln* (*real* (*n-of x*)) +
1 / (*of-rat* (δ*-of x*))^2 ∗ (*ln* (*ln* (*real* (*n-of x*))) + *ln* (*1* / *of-rat* (δ*-of x*)))))

have *evt*: (⋀*x*.
0 < *real-of-rat* (δ*-of x*) ∧ *0* < *real-of-rat* (ε*-of x*) ∧
1/*real-of-rat* (δ*-of x*) ≥ δ ∧ *1*/*real-of-rat* (ε*-of x*) ≥ ε ∧
real (*n-of x*) ≥ *n* ⟹ *P x*) ⟹ *eventually P ?F* (**is** (⋀*x*. *?prem x* ⟹ -) ⟹ -)
for δ ε *n P*
apply (*rule eventually-mono*[**where** *P*=*?prem* **and** *Q*=*P*])
apply (*simp add*:ε*-of-def case-prod-beta′* δ*-of-def n-of-def*)
apply (*intro eventually-conj eventually-prod1′ eventually-prod2′
sequentially-inf eventually-at-right-less inv-at-right-0-inf*)
by (*auto simp add*:*prod-filter-eq-bot*)

have *exp-pos*: *exp k* ≤ *real x* ⟹ *x* > *0* **for** *k x*
using *exp-gt-zero gr0I* **by** *force*

have *exp-gt-1*: *exp 1* ≥ (*1*::*real*)
by *simp*

have *1*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
by (*auto intro*!:*landau-o.big-mono evt*[**where** ε=*exp 1*] *iffD2*[*OF ln-ge-iff*] *simp add*:*abs-ge-iff*)

have *2*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (δ*-of x*)))
by (*auto intro*!:*landau-o.big-mono evt*[**where** δ=*exp 1*] *iffD2*[*OF ln-ge-iff*] *simp add*:*abs-ge-iff*)

have *3*: (λ*x*. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*ln* (*real* (*n-of x*))) + *ln* (*1* / *real-of-rat* (δ*-of x*)))
using *exp-pos*
by (*intro landau-sum-2 2 evt*[**where** *n*=*exp 1* **and** δ=*1*] *ln-ge-zero iffD2*[*OF ln-ge-iff*], *auto*)

have *4*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2)
  using *one-le-power*
  by (*intro landau-o.big-mono evt*[**where** δ=*1*], *auto simp add*:*power-one-over*[*symmetric*])

have (λ*x*. *80* ∗ (*1* / (*real-of-rat* (δ*-of x*))^2)) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2)
  by (*subst landau-o.big.cmult-in-iff*, *auto*)
hence *5*: (λ*x*. *real* (*t-of x*)) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2)
  unfolding *t-of-def*
  by (*intro landau-real-nat landau-ceil 4*, *auto*)

have (λ*x*. *ln* (*real-of-rat* (ε*-of x*))) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
  by (*intro landau-o.big-mono evt*[**where** ε=*1*], *auto simp add*:*ln-div*)
hence *6*: (λ*x*. *real* (*s-of x*)) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
  unfolding *s-of-def* **by** (*intro landau-nat-ceil 1*, *simp*)

have *7*: (λ*x*. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)))
  using *exp-pos* **by** (*auto intro*!: *landau-o.big-mono evt*[**where** *n*=*exp 1*] *iffD2*[*OF ln-ge-iff*] *simp*: *abs-ge-iff*)

have *8*: (λ-. *1*) ∈
  O[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *1* / (*real-of-rat* (δ*-of x*))^2 ∗ (*ln* (*ln* (*real* (*n-of x*))) + *ln* (*1* / *real-of-rat* (δ*-of x*))))
  using *order-trans*[*OF exp-gt-1*] *exp-pos*
  by (*intro landau-sum-1 7 evt*[**where** *n*=*exp 1* **and** δ=*1*] *ln-ge-zero iffD2*[*OF ln-ge-iff*]
    mult-nonneg-nonneg add-nonneg-nonneg) *auto*

have (λ*x*. *ln* (*real* (*s-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
  by (*intro landau-ln-3 sum-in-bigo 6 1*, *simp*)

hence *9*: (λ*x*. *log 2* (*real* (*s-of x*) + *1*)) ∈ *O*[*?F*](*g*)
  unfolding *g-def* **by** (*intro landau-o.big-mult-1 8*, *auto simp*:*log-def*)
have *10*: (λ*x*. *1*) ∈ *O*[*?F*](*g*)
  unfolding *g-def* **by** (*intro landau-o.big-mult-1 8 1*)

have (λ*x*. *ln* (*real* (*t-of x*) + *1*)) ∈
  O[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2 ∗ (*ln* (*ln* (*real* (*n-of x*))) + *ln* (*1* / *real-of-rat* (δ*-of x*))))
  using *5* **by** (*intro landau-o.big-mult-1 3 landau-ln-3 sum-in-bigo 4*, *simp-all*)
hence (λ*x*. *log 2* (*real* (*t-of x*) + *1*)) ∈
  O[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *1* / (*real-of-rat* (δ*-of x*))^2 ∗ (*ln* (*ln* (*real* (*n-of x*))) + *ln* (*1* / *real-of-rat* (δ*-of x*))))
  using *order-trans*[*OF exp-gt-1*] *exp-pos*
  by (*intro landau-sum-2 evt*[**where** *n*=*exp 1* **and** δ=*1*] *ln-ge-zero iffD2*[*OF ln-ge-iff*]
    mult-nonneg-nonneg add-nonneg-nonneg) (*auto simp add*:*log-def*)
hence *11*: (λ*x*. *log 2* (*real* (*t-of x*) + *1*)) ∈ *O*[*?F*](*g*)
  unfolding *g-def* **by** (*intro landau-o.big-mult-1′ 1*, *auto*)

have $(\lambda x.\ 1) \in O[?F](\lambda x.\ real\ (n\text{-}of\ x))$
by (*intro landau-o.big-mono evt*[**where** *n=1*], *auto*)
hence $(\lambda x.\ ln\ (real\ (n\text{-}of\ x) + 21)) \in O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)))$
by (*intro landau-ln-2*[**where** *a=2*] *evt*[**where** *n=2*] *sum-in-bigo*, *auto*)

hence *12*: $(\lambda x.\ log\ 2\ (real\ (n\text{-}of\ x) + 21)) \in O[?F](g)$
unfolding *g-def* **using** *exp-pos order-trans*[*OF exp-gt-1*]
by (*intro landau-o.big-mult-1' 1 landau-sum-1 evt*[**where** *n=exp 1* **and** *δ=1*]
ln-ge-zero iffD2[*OF ln-ge-iff*] *mult-nonneg-nonneg add-nonneg-nonneg*)
(*auto simp add*:*log-def*)

have $(\lambda x.\ ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))) \in O[?F](\lambda x.\ 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2)$
by (*intro landau-ln-3 evt*[**where** *δ=1*] *landau-o.big-mono*)
(*auto simp add*:*power-one-over*[*symmetric*] *self-le-power*)
hence $(\lambda x.\ real\ (nat\ (4*\lceil log\ 2\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))\rceil+23))) \in O[?F](\lambda x.\ 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2)$
using *4* **by** (*auto intro*!: *landau-real-nat sum-in-bigo landau-ceil simp*:*log-def*)
hence $(\lambda x.\ ln\ (real\ (r\text{-}of\ x) + 1)) \in O[?F](\lambda x.\ 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2)$
unfolding *r-of-def*
by (*intro landau-ln-3 sum-in-bigo 4*, *auto*)
hence $(\lambda x.\ log\ 2\ (real\ (r\text{-}of\ x) + 1)) \in$
$O[?F](\lambda x.\ (1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2) * (ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))))$
by (*intro landau-o.big-mult-1 3*, *simp add*:*log-def*)
hence $(\lambda x.\ log\ 2\ (real\ (r\text{-}of\ x) + 1)) \in$
$O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)) + 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2 * (ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))))$
using *exp-pos order-trans*[*OF exp-gt-1*]
by (*intro landau-sum-2 evt*[**where** *n=exp 1* **and** *δ=1*] *ln-ge-zero*
iffD2[*OF ln-ge-iff*] *add-nonneg-nonneg mult-nonneg-nonneg*) (*auto*)
hence *13*: $(\lambda x.\ log\ 2\ (real\ (r\text{-}of\ x) + 1)) \in O[?F](g)$
unfolding *g-def* **by** (*intro landau-o.big-mult-1' 1*, *auto*)
have *14*: $(\lambda x.\ 1) \in O[?F](\lambda x.\ real\ (n\text{-}of\ x))$
by (*intro landau-o.big-mono evt*[**where** *n=1*], *auto*)

have $(\lambda x.\ ln\ (real\ (n\text{-}of\ x) + 13)) \in O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)))$
using *14* **by** (*intro landau-ln-2*[**where** *a=2*] *evt*[**where** *n=2*] *sum-in-bigo*, *auto*)

hence $(\lambda x.\ ln\ (log\ 2\ (real\ (n\text{-}of\ x) + 13))) \in O[?F](\lambda x.\ ln\ (ln\ (real\ (n\text{-}of\ x))))$
using *exp-pos* **by** (*intro landau-ln-2*[**where** *a=2*] *iffD2*[*OF ln-ge-iff*] *evt*[**where** *n=exp 2*])
(*auto simp add*:*log-def*)

hence $(\lambda x.\ log\ 2\ (log\ 2\ (real\ (n\text{-}of\ x) + 13))) \in O[?F](\lambda x.\ ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x)))$
using *exp-pos* **by** (*intro landau-sum-1 evt*[**where** *n=exp 1* **and** *δ=1*] *ln-ge-zero iffD2*[*OF ln-ge-iff*])
(*auto simp add*:*log-def*)

moreover have $(\lambda x.\ real\ (r\text{-}of\ x)) \in O[?F](\lambda x.\ ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x)))$
unfolding *r-of-def* **using** *2*
by (*auto intro*!: *landau-real-nat sum-in-bigo landau-ceil simp*:*log-def*)
hence $(\lambda x.\ real\ (r\text{-}of\ x)) \in O[?F](\lambda x.\ ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x)))$
using *exp-pos*
by (*intro landau-sum-2 evt*[**where** *n*=*exp 1* **and** δ=*1*] *ln-ge-zero iffD2*[*OF ln-ge-iff*], *auto*)

ultimately have *15*: $(\lambda x.\ real\ (t\text{-}of\ x) * (13 + 4 * real\ (r\text{-}of\ x) + 2 * log\ 2\ (log\ 2\ (real\ (n\text{-}of\ x) + 13))))$
$\in O[?F](\lambda x.\ 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2 * (ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))))$
using *5 3*
by (*intro landau-o.mult sum-in-bigo*, *auto*)

have $(\lambda x.\ 5 + 2 * log\ 2\ (21 + real\ (n\text{-}of\ x)) + real\ (t\text{-}of\ x) * (13 + 4 * real\ (r\text{-}of\ x) + 2 * log\ 2\ (log\ 2\ (real\ (n\text{-}of\ x) + 13))))$
$\in O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)) + 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2 * (ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))))$
proof −
have $\forall_F\ x\ in\ ?F.\ 0 \le ln\ (real\ (n\text{-}of\ x))$
by (*intro evt*[**where** *n*=*1*] *ln-ge-zero*, *auto*)
moreover have $\forall_F\ x\ in\ ?F.\ 0 \le 1\ /\ (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x))^2 * (ln\ (ln\ (real\ (n\text{-}of\ x))) + ln\ (1\ /\ real\text{-}of\text{-}rat\ (\delta\text{-}of\ x)))$
using *exp-pos*
by (*intro evt*[**where** *n*=*exp 1* **and** δ=*1*] *mult-nonneg-nonneg add-nonneg-nonneg*

ln-ge-zero iffD2[*OF ln-ge-iff*]) *auto*
moreover have $(\lambda x.\ ln\ (21 + real\ (n\text{-}of\ x))) \in O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)))$
using *14* **by** (*intro landau-ln-2*[**where** *a*=*2*] *sum-in-bigo evt*[**where** *n*=*2*], *auto*)
hence $(\lambda x.\ 5 + 2 * log\ 2\ (21 + real\ (n\text{-}of\ x))) \in O[?F](\lambda x.\ ln\ (real\ (n\text{-}of\ x)))$
using *7* **by** (*intro sum-in-bigo*, *auto simp add*:*log-def*)
ultimately show *?thesis*
using *15* **by** (*rule landau-sum*)
qed

hence *16*: $(\lambda x.\ real\ (s\text{-}of\ x) * (5 + 2 * log\ 2\ (21 + real\ (n\text{-}of\ x)) + real\ (t\text{-}of\ x) *$
$(13 + 4 * real\ (r\text{-}of\ x) + 2 * log\ 2\ (log\ 2\ (real\ (n\text{-}of\ x) + 13))))) \in O[?F](g)$
unfolding *g-def*
by (*intro landau-o.mult 6*, *auto*)

have *f0-space-usage* = $(\lambda x.\ f0\text{-}space\text{-}usage\ (n\text{-}of\ x, \varepsilon\text{-}of\ x, \delta\text{-}of\ x))$
by (*simp add*:*case-prod-beta′ n-of-def ε-of-def δ-of-def*)
also have ... $\in O[?F](g)$
using *9 10 11 12 13 16*

by (*simp add:fun-cong*[*OF s-of-def*[*symmetric*]] *fun-cong*[*OF t-of-def*[*symmetric*]]

fun-cong[*OF r-of-def*[*symmetric*]] *Let-def*) (*intro sum-in-bigo*, *auto*)
also have ... = *O*[*?F*](*?rhs*)
by (*simp add:case-prod-beta′ g-def n-of-def ε-of-def δ-of-def*)
finally show *?thesis*
by *simp*
qed

end

8 Frequency Moment 2

theory *Frequency-Moment-2*
imports
Universal-Hash-Families.Carter-Wegman-Hash-Family
Equivalence-Relation-Enumeration.Equivalence-Relation-Enumeration
Landau-Ext
Median-Method.Median
Product-PMF-Ext
Universal-Hash-Families.Field
Frequency-Moments
begin

This section contains a formalization of the algorithm for the second frequency moment. It is based on the algorithm described in [1, §2.2]. The only difference is that the algorithm is adapted to work with prime field of odd order, which greatly reduces the implementation complexity.

fun *f2-hash* **where**
f2-hash p h k = (*if even* (*ring.hash* (*mod-ring p*) *k h*) *then int p* − *1 else* − *int p* − *1*)

type-synonym *f2-state* = *nat* × *nat* × *nat* × (*nat* × *nat* ⇒ *nat list*) × (*nat* × *nat* ⇒ *int*)

fun *f2-init* :: *rat* ⇒ *rat* ⇒ *nat* ⇒ *f2-state pmf* **where**
f2-init δ ε n =
do {
let s_1 = *nat* ⌈*6* / δ^2⌉;
let s_2 = *nat* ⌈−(*18* ∗ *ln* (*real-of-rat ε*))⌉;
let p = *prime-above* (*max n 3*);
h ← *prod-pmf* ({..<s_1} × {..<s_2}) (λ-. *pmf-of-set* (*bounded-degree-polynomials* (*mod-ring p*) *4*));
return-pmf (s_1, s_2, *p*, *h*, (λ- ∈ {..<s_1} × {..<s_2}. (*0* :: *int*)))
}

fun *f2-update* :: *nat* ⇒ *f2-state* ⇒ *f2-state pmf* **where**
f2-update x (s_1, s_2, *p*, *h*, *sketch*) =

return-pmf $(s_1, s_2, p, h, \lambda i \in \{..<s_1\} \times \{..<s_2\}.$ *f2-hash p* $(h\ i)$ x + *sketch i*)

fun *f2-result* :: *f2-state* ⇒ *rat pmf* **where**
f2-result $(s_1, s_2, p, h,$ *sketch*) =
return-pmf (*median* s_2 $(\lambda i_2 \in \{..<s_2\}.$
$(\sum i_1 \in \{..<s_1\}$. (*rat-of-int* (*sketch* $(i_1, i_2)))^2)$ / (((*rat-of-nat* $p)^2-1)$ ∗ *rat-of-nat* s_1)))

fun *f2-space-usage* :: (*nat* × *nat* × *rat* × *rat*) ⇒ *real* **where**
f2-space-usage $(n, m, \varepsilon, \delta)$ = (
let s_1 = *nat* $\lceil 6 / \delta^2 \rceil$ *in*
let s_2 = *nat* $\lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ *in*
3 +
2 ∗ *log 2* $(s_1 + 1)$ +
2 ∗ *log 2* $(s_2 + 1)$ +
2 ∗ *log 2* (*9* + *2* ∗ *real n*) +
$s_1 * s_2$ ∗ (*5* + *4*∗*log 2* (*8* + *2* ∗ *real n*) + *2* ∗ *log 2* (*real m* ∗ (*18* + *4* ∗ *real n*) + *1*)))

definition *encode-f2-state* :: *f2-state* ⇒ *bool list option* **where**
encode-f2-state =
$N_e \bowtie_e (\lambda s_1.$
$N_e \bowtie_e (\lambda s_2.$
$N_e \bowtie_e (\lambda p.$
(*List.product* $[0..<s_1]$ $[0..<s_2]$ $\rightarrow_e P_e\ p\ 4) \times_e$
(*List.product* $[0..<s_1]$ $[0..<s_2]$ $\rightarrow_e I_e))))$

lemma *inj-on encode-f2-state* (*dom encode-f2-state*)
proof −
have *is-encoding encode-f2-state*
unfolding *encode-f2-state-def*
by (*intro dependent-encoding exp-golomb-encoding fun-encoding list-encoding int-encoding poly-encoding*)

thus *?thesis*
by (*rule encoding-imp-inj*)
qed

context
fixes ε δ :: *rat*
fixes *n* :: *nat*
fixes *as* :: *nat list*
fixes *result*
assumes *ε-range*: $\varepsilon \in \{0<..<1\}$
assumes *δ-range*: $\delta > 0$
assumes *as-range*: *set as* ⊆ {..<*n*}
defines *result* ≡ *fold* (λ*a state. state* ≫= *f2-update a*) *as* (*f2-init* δ ε *n*) ≫= *f2-result*
begin

private definition s_1 **where** $s_1 = nat \lceil 6 / \delta^2 \rceil$

lemma *s1-gt-0*: $s_1 > 0$
  using *δ-range* **by** (*simp add*:s_1*-def*)

private definition s_2 **where** $s_2 = nat \lceil -(18* \ln (\text{real-of-rat } \varepsilon)) \rceil$

lemma *s2-gt-0*: $s_2 > 0$
  using *ε-range* **by** (*simp add*:s_2*-def*)

private definition *p* **where** *p* = *prime-above* (*max n 3*)

lemma *p-prime*: *Factorial-Ring.prime p*
 unfolding *p-def* **using** *prime-above-prime* **by** *blast*

lemma *p-ge-3*: $p \geq 3$
  unfolding *p-def* **by** (*meson max.boundedE prime-above-lower-bound*)

lemma *p-gt-0*: $p > 0$ **using** *p-ge-3* **by** *linarith*

lemma *p-gt-1*: $p > 1$ **using** *p-ge-3* **by** *simp*

lemma *p-ge-n*: $p \geq n$ **unfolding** *p-def*
 by (*meson max.boundedE prime-above-lower-bound*)

interpretation *carter-wegman-hash-family mod-ring p 4*
 using *carter-wegman-hash-familyI*[*OF mod-ring-is-field mod-ring-finite*]
 using *p-prime* **by** *auto*

definition *sketch* **where** *sketch* = *fold* (λa *state. state* ≫= *f2-update a*) *as* (*f2-init* δ ε *n*)
private definition Ω **where**Ω = *prod-pmf* ({..<s_1} × {..<s_2}) (λ-. *pmf-of-set space*)
private definition Ω_p **where**Ω_p = *measure-pmf* Ω
private definition *sketch-rv* **where** *sketch-rv* ω = *of-int* (*sum-list* (*map* (*f2-hash p* ω) *as*))^*2*
private definition *mean-rv* **where** *mean-rv* ω = (λi_2. ($\sum i_1 = 0..<s_1$. *sketch-rv* (ω (i_1, i_2))) / (((*of-nat p*)^2 − *1*) ∗ *of-nat* s_1))
private definition *result-rv* **where** *result-rv* ω = *median* s_2 ($\lambda i_2 \in \{..<s_2\}$. *mean-rv* ω i_2)

lemma *mean-rv-alg-sketch*:
 sketch = Ω ≫= (λω. *return-pmf* (s_1, s_2, *p*, ω, λ*i* ∈ {..<s_1} × {..<s_2}. *sum-list* (*map* (*f2-hash p* (ω *i*)) *as*)))
proof −
 have *sketch* = *fold* (λ*a state. state* ≫= *f2-update a*) *as* (*f2-init* δ ε *n*)
  by (*simp add*:*sketch-def*)
 also have ... = Ω ≫= (λω. *return-pmf* (s_1, s_2, *p*, ω,

$\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (ω *i*)) *as*)))
proof (*induction as rule*:*rev-induct*)
case *Nil*
then show *?case*
by (*simp add*:s_1*-def* s_2*-def space-def p-def*[*symmetric*] Ω*-def restrict-def Let-def*)
next
case (*snoc a as*)
have *fold* (λ*a state. state* $\gg=$ *f2-update a*) (*as* @ [*a*]) (*f2-init* δ ε *n*) = Ω $\gg=$
($\lambda\omega$. *return-pmf* (s_1, s_2, *p*, ω, $\lambda s \in \{..<s_1\} \times \{..<s_2\}$. ($\sum x \leftarrow as$. *f2-hash p* ($\omega$ *s*) *x*)) $\gg=$ *f2-update a*)
using *snoc* **by** (*simp add*: *bind-assoc-pmf restrict-def del*:*f2-hash.simps f2-init.simps*)
also have ... = Ω $\gg=$ ($\lambda\omega$. *return-pmf* (s_1, s_2, *p*, ω, $\lambda i \in \{..<s_1\} \times \{..<s_2\}$. ($\sum x \leftarrow as@[a]$. *f2-hash p* ($\omega$ *i*) *x*)))
by (*subst bind-return-pmf*) (*simp add*: *add.commute del*:*f2-hash.simps cong*:*restrict-cong*)
finally show *?case* **by** *blast*
qed
finally show *?thesis* **by** *auto*
qed

lemma *distr*: *result* = *map-pmf result-rv* Ω
proof −
have *result* = *sketch* $\gg=$ *f2-result*
by (*simp add*:*result-def sketch-def*)
also have ... = Ω $\gg=$ (λx. *f2-result* (s_1, s_2, *p*, *x*, $\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*x i*)) *as*)))
by (*simp add*: *mean-rv-alg-sketch bind-assoc-pmf bind-return-pmf*)
also have ... = *map-pmf result-rv* Ω
by (*simp add*:*map-pmf-def result-rv-def mean-rv-def sketch-rv-def lessThan-atLeast0 cong*:*restrict-cong*)
finally show *?thesis* **by** *simp*
qed

private lemma *f2-hash-pow-exp*:
assumes $k < p$
shows
expectation ($\lambda\omega$. *real-of-int* (*f2-hash p* ω *k*) ^*m*) =
((*real p* − *1*) ^ *m* ∗ (*real p* + *1*) + (− *real p* − *1*) ^ *m* ∗ (*real p* − *1*)) / (*2* ∗ *real p*)
proof −

have *odd p* **using** *p-prime p-ge-3 prime-odd-nat assms* **by** *simp*
then obtain *t* **where** *t-def*: *p=2∗t+1*
using *oddE* **by** *blast*

have *Collect even* ∩ {..<*2* ∗ *t* + *1*} ⊆ (∗) *2* ‘ {..<*t* + *1*}
by (*rule in-image-by-witness*[**where** *g*=λx. *x div 2*], *simp*, *linarith*)
moreover have (∗) *2* ‘ {..<*t* + *1*} ⊆ *Collect even* ∩ {..<*2* ∗ *t* + *1*}
by (*rule image-subsetI*, *simp*)

ultimately have card ({k. even k} ∩ {..<p}) = card ((λx. 2∗x) ‘ {..<t+1})
  unfolding t-def **using** order-antisym **by** metis
also have ... = card {..<t+1}
  by (rule card-image, simp add: inj-on-mult)
also have ... = t+1 **by** simp
finally have card-even: card ({k. even k} ∩ {..<p}) = t+1 **by** simp
hence card ({k. even k} ∩ {..<p}) ∗ 2 = (p+1) **by** (simp add:t-def)
hence prob-even: prob {ω. hash k ω ∈ Collect even} = (real p + 1)/(2∗real p)
  using assms **by** (subst prob-range, auto simp:frac-eq-eq p-gt-0 mod-ring-def)

have p = card {..<p} **by** simp
also have ... = card (({k. odd k} ∩ {..<p}) ∪ ({k. even k} ∩ {..<p}))
  by (rule arg-cong[**where** f=card], auto)
also have ... = card ({k. odd k} ∩ {..<p}) + card ({k. even k} ∩ {..<p})
  by (rule card-Un-disjoint, simp, simp, blast)
also have ... = card ({k. odd k} ∩ {..<p}) + t+1
  by (simp add:card-even)
finally have p = card ({k. odd k} ∩ {..<p}) + t+1
  by simp
hence card ({k. odd k} ∩ {..<p}) ∗ 2 = (p−1)
  by (simp add:t-def)
hence prob-odd: prob {ω. hash k ω ∈ Collect odd} = (real p − 1)/(2∗real p)
  using assms **by** (subst prob-range, auto simp add: frac-eq-eq mod-ring-def)

have expectation (λx. real-of-int (f2-hash p x k) ^ m) =
  expectation (λω. indicator {ω. even (hash k ω)} ω ∗ (real p − 1)^m +
    indicator {ω. odd (hash k ω)} ω ∗ (−real p − 1)^m)
  by (rule Bochner-Integration.integral-cong, simp, simp)
also have ... =
   prob {ω. hash k ω ∈ Collect even} ∗ (real p − 1) ^ m +
   prob {ω. hash k ω ∈ Collect odd} ∗ (−real p − 1) ^ m
  by (simp, simp add:M-def)
also have ... = (real p + 1) ∗ (real p − 1) ^ m / (2 ∗ real p) + (real p − 1) ∗ (− real p − 1) ^ m / (2 ∗ real p)
  by (subst prob-even, subst prob-odd, simp)
also have ... =
  ((real p − 1) ^ m ∗ (real p + 1) + (− real p − 1) ^ m ∗ (real p − 1)) / (2 ∗ real p)
  by (simp add:add-divide-distrib ac-simps)
finally show expectation (λx. real-of-int (f2-hash p x k) ^ m) =
  ((real p − 1) ^ m ∗ (real p + 1) + (− real p − 1) ^ m ∗ (real p − 1)) / (2 ∗ real p) **by** simp
qed

lemma
 shows var-sketch-rv:variance sketch-rv ≤ 2∗(real-of-rat (F 2 as)^2) ∗ ((real p)^2−1)^2 (**is** ?A)
 and exp-sketch-rv:expectation sketch-rv = real-of-rat (F 2 as) ∗ ((real p)^2−1) (**is** ?B)

proof −
define *h* **where** *h* = (λω *x*. *real-of-int* (*f2-hash p* ω *x*))
define *c* **where** *c* = (λ*x*. *real* (*count-list as x*))
define *r* **where** *r* = (λ(*m*::*nat*). ((*real p* − *1*) ^ *m* ∗ (*real p* + *1*) + (− *real p* − *1*) ^ *m* ∗ (*real p* − *1*)) / (*2* ∗ *real p*))
define *h-prod* **where** *h-prod* = (λ*as* ω. *prod-list* (*map* (*h* ω) *as*))

define *exp-h-prod* :: *nat list* ⇒ *real* **where** *exp-h-prod* = (λ*as*. (∏ *i* ∈ *set as*. *r* (*count-list as i*)))

have *f-eq*: *sketch-rv* = (λω. (∑ *x* ∈ *set as*. *c x* ∗ *h* ω *x*)^*2*)
by (*rule ext*, *simp add*:*sketch-rv-def c-def h-def sum-list-eval del*:*f2-hash.simps*)

have *r-one*: *r* (*Suc 0*) = *0*
by (*simp add*:*r-def algebra-simps*)

have *r-two*: *r 2* = (*real p*^*2*−*1*)
using *p-gt-0* **unfolding** *r-def power2-eq-square*
by (*simp add*:*nonzero-divide-eq-eq*, *simp add*:*algebra-simps*)

have(*real p*)^*2* ≥ *2*^*2*
by (*rule power-mono*, *use p-gt-1* **in** *linarith*, *simp*)
hence *p-square-ge-4*: (*real p*)^2 ≥ *4* **by** *simp*

have *r 4* = (*real p*)^*4*+*2*∗(*real p*)^2 − *3*
using *p-gt-0* **unfolding** *r-def*
by (*subst nonzero-divide-eq-eq*, *auto simp*:*power4-eq-xxxx power2-eq-square algebra-simps*)
also have ... ≤ (*real p*)^*4*+*2*∗(*real p*)^2 + *3*
by *simp*
also have ... ≤ *3* ∗ *r 2* ∗ *r 2*
using *p-square-ge-4*
by (*simp add*:*r-two power4-eq-xxxx power2-eq-square algebra-simps mult-left-mono*)
finally have *r-four-est*: *r 4* ≤ *3* ∗ *r 2* ∗ *r 2* **by** *simp*

have *exp-h-prod-elim*: *exp-h-prod* = (λ*as*. *prod-list* (*map* (*r* ∘ *count-list as*) (*remdups as*)))
by (*simp add*:*exp-h-prod-def prod.set-conv-list*[*symmetric*])

have *exp-h-prod*: ⋀*x*. *set x* ⊆ *set as* ⟹ *length x* ≤ *4* ⟹ *expectation* (*h-prod x*) = *exp-h-prod x*
proof −
fix *x*
assume *set x* ⊆ *set as*
hence *x-sub-p*: *set x* ⊆ {..<*p*} **using** *as-range p-ge-n* **by** *auto*
hence *x-le-p*: ⋀*k*. *k* ∈ *set x* ⟹ *k* < *p* **by** *auto*
assume *length x* ≤ *4*
hence *card-x*: *card* (*set x*) ≤ *4* **using** *card-length dual-order.trans* **by** *blast*

have *set x* ⊆ *carrier* (*mod-ring p*)
using *x-sub-p* **by** (*simp add*:*mod-ring-def*)

hence *h-indep*: *indep-vars* (λ-. *borel*) (λ*i* ω. *h* ω *i* ^ *count-list x i*) (*set x*)
using *k-wise-indep-vars-subset*[*OF k-wise-indep*] *card-x as-range h-def*
by (*auto intro*:*indep-vars-compose2*[**where** *X*=*hash* **and** *M'*= (λ-. *discrete*)])

have *expectation* (*h-prod x*) = *expectation* (λω. ∏ *i* ∈ *set x*. *h* ω *i*^(*count-list x i*))
by (*simp add*:*h-prod-def prod-list-eval*)
also have ... = (∏ *i* ∈ *set x*. *expectation* (λω. *h* ω *i*^(*count-list x i*)))
by (*simp add*: *indep-vars-lebesgue-integral*[*OF - h-indep*])
also have ... = (∏ *i* ∈ *set x*. *r* (*count-list x i*))
using *f2-hash-pow-exp x-le-p*
by (*simp add*:*h-def r-def M-def*[*symmetric*] *del*:*f2-hash.simps*)
also have ... = *exp-h-prod x*
by (*simp add*:*exp-h-prod-def*)
finally show *expectation* (*h-prod x*) = *exp-h-prod x* **by** *simp*
qed

have ⋀*x y*. *kernel-of x* = *kernel-of y* ⟹ *exp-h-prod x* = *exp-h-prod y*
proof −
fix *x y* :: *nat list*
assume *a*:*kernel-of x* = *kernel-of y*
then obtain *f* **where** *b*:*bij-betw f* (*set x*) (*set y*) **and** *c*:⋀*z*. *z* ∈ *set x* ⟹ *count-list x z* = *count-list y* (*f z*)
using *kernel-of-eq-imp-bij* **by** *blast*
have *exp-h-prod x* = *prod* ((λ*i*. *r*(*count-list y i*)) ∘ *f*) (*set x*)
by (*simp add*:*exp-h-prod-def c*)
also have ... = (∏ *i* ∈ *f* ‘ (*set x*). *r*(*count-list y i*))
by (*metis b bij-betw-def prod.reindex*)
also have ... = *exp-h-prod y*
unfolding *exp-h-prod-def*
by (*rule prod.cong*, *metis b bij-betw-def*) *simp*
finally show *exp-h-prod x* = *exp-h-prod y* **by** *simp*
qed

hence *exp-h-prod-cong*: ⋀*p x*. *of-bool* (*kernel-of x* = *kernel-of p*) ∗ *exp-h-prod p* =
of-bool (*kernel-of x* = *kernel-of p*) ∗ *exp-h-prod x*
by (*metis* (*full-types*) *of-bool-eq-0-iff vector-space-over-itself.scale-zero-left*)

have *c*:(∑ *p*←*enum-rgfs n*. *of-bool* (*kernel-of xs* = *kernel-of p*) ∗ *r*) = *r*
if *a*:*length xs* = *n* **for** *xs* :: *nat list* **and** *n* **and** *r* :: *real*
proof −
have (∑ *p*←*enum-rgfs n*. *of-bool* (*kernel-of xs* = *kernel-of p*) ∗ *1*) = (*1*::*real*)
using *equiv-rels-2*[*OF a*[*symmetric*]] **by** (*simp add*:*equiv-rels-def comp-def*)
thus (∑ *p*←*enum-rgfs n*. *of-bool* (*kernel-of xs* = *kernel-of p*) ∗ *r*) = (*r*::*real*)
by (*simp add*:*sum-list-mult-const*)

qed

have *expectation sketch-rv* = (∑ *i*∈*set as*. (∑ *j*∈*set as*. *c i* ∗ *c j* ∗ *expectation* (*h-prod* [*i*,*j*])))
by (*simp add*:*f-eq h-prod-def power2-eq-square sum-distrib-left sum-distrib-right Bochner-Integration*.*integral-sum algebra-simps*)
also have ... = (∑ *i*∈*set as*. (∑ *j*∈*set as*. *c i* ∗ *c j* ∗ *exp-h-prod* [*i*,*j*]))
by (*simp add*:*exp-h-prod*)
also have ... = (∑ *i* ∈ *set as*. (∑ *j* ∈ *set as*.
c i ∗ *c j* ∗ (*sum-list* (*map* (λ*p*. *of-bool* (*kernel-of* [*i*,*j*] = *kernel-of p*) ∗ *exp-h-prod p*) (*enum-rgfs 2*)))))
by (*subst exp-h-prod-cong*, *simp add*:*c*)
also have ... = (∑ *i*∈*set as*. *c i* ∗ *c i* ∗ *r 2*)
by (*simp add*: *numeral-eq-Suc kernel-of-eq All-less-Suc exp-h-prod-elim r-one distrib-left sum*.*distrib sum-collapse*)
also have ... = *real-of-rat* (*F 2 as*) ∗ ((*real p*)^*2*−*1*)
by (*simp add*: *sum-distrib-right*[*symmetric*] *c-def F-def power2-eq-square of-rat-sum of-rat-mult r-two*)
finally show *b*:*?B* **by** *simp*

have *expectation* (λ*x*. (*sketch-rv x*)^2) = (∑ *i1* ∈ *set as*. (∑ *i2* ∈ *set as*. (∑ *i3* ∈ *set as*. (∑ *i4* ∈ *set as*.
c i1 ∗ *c i2* ∗ *c i3* ∗ *c i4* ∗ *expectation* (*h-prod* [*i1*, *i2*, *i3*, *i4*])))))
by (*simp add*:*f-eq h-prod-def power4-eq-xxxx sum-distrib-left sum-distrib-right Bochner-Integration*.*integral-sum algebra-simps*)
also have ... = (∑ *i1* ∈ *set as*. (∑ *i2* ∈ *set as*. (∑ *i3* ∈ *set as*. (∑ *i4* ∈ *set as*.
c i1 ∗ *c i2* ∗ *c i3* ∗ *c i4* ∗ *exp-h-prod* [*i1*,*i2*,*i3*,*i4*]))))
by (*simp add*:*exp-h-prod*)
also have ... = (∑ *i1* ∈ *set as*. (∑ *i2* ∈ *set as*. (∑ *i3* ∈ *set as*. (∑ *i4* ∈ *set as*.
c i1 ∗ *c i2* ∗ *c i3* ∗ *c i4* ∗
(*sum-list* (*map* (λ*p*. *of-bool* (*kernel-of* [*i1*,*i2*,*i3*,*i4*] = *kernel-of p*) ∗ *exp-h-prod p*) (*enum-rgfs 4*)))))))
by (*subst exp-h-prod-cong*, *simp add*:*c*)
also have ... =
3 ∗ (∑ *i* ∈ *set as*. (∑ *j* ∈ *set as*. *c i*^*2* ∗ *c j*^*2* ∗ *r 2* ∗ *r 2*)) + ((∑ *i* ∈ *set as*. *c i*^*4* ∗ *r 4*) − *3* ∗ (∑ *i* ∈ *set as*. *c i* ^ *4* ∗ *r 2* ∗ *r 2*))
apply (*simp add*: *numeral-eq-Suc exp-h-prod-elim r-one*)
apply (*simp add*: *kernel-of-eq All-less-Suc numeral-eq-Suc distrib-left sum*.*distrib sum-collapse neq-commute of-bool-not-iff*)
apply (*simp add*: *algebra-simps sum-subtractf sum-collapse*)
apply (*simp add*: *sum-distrib-left algebra-simps*)
done
also have ... = *3* ∗ (∑ *i* ∈ *set as*. *c i*^*2* ∗ *r 2*)^*2* + (∑ *i* ∈ *set as*. *c i* ^ *4* ∗ (*r 4* − *3* ∗ *r 2* ∗ *r 2*))
by (*simp add*:*power2-eq-square sum-distrib-left algebra-simps sum-subtractf*)
also have ... = *3* ∗ (∑ *i* ∈ *set as*. *c i*^*2*)^*2* ∗ (*r 2*)^*2* + (∑ *i* ∈ *set as*. *c i* ^ *4* ∗ (*r 4* − *3* ∗ *r 2* ∗ *r 2*))
by (*simp add*:*power-mult-distrib sum-distrib-right*[*symmetric*])
also have ... ≤ *3* ∗ (∑ *i* ∈ *set as*. *c i*^*2*)^*2* ∗ (*r 2*)^*2* + (∑ *i* ∈ *set as*. *c i* ^ *4*

$* \ 0$)
  using *r-four-est*
  by (*auto intro*!: *sum-nonpos simp add*:*mult-nonneg-nonpos*)
 also have ... $= 3 * (\textit{real-of-rat} \ (F \ 2 \ as)\hat{\ }2) * ((\textit{real} \ p)^2{-}1)^2$
  by (*simp add*:*c-def r-two F-def of-rat-sum of-rat-power*)
 finally have *expectation* $(\lambda x. \ (\textit{sketch-rv} \ x)^2) \leq 3 * (\textit{real-of-rat} \ (F \ 2 \ as)\hat{\ }2) * ((\textit{real} \ p)^2{-}1)^2$
  by *simp*

 thus *variance sketch-rv* $\leq 2{*}(\textit{real-of-rat} \ (F \ 2 \ as)\hat{\ }2) * ((\textit{real} \ p)^2{-}1)^2$
  by (*simp add*: *variance-eq*, *simp add*:*power-mult-distrib b*)
qed

lemma *space-omega-1* [*simp*]: *Sigma-Algebra.space* Ω_p = *UNIV*
  by (*simp add*:Ω_p*-def*)

interpretation Ω: *prob-space* Ω_p
 by (*simp add*:Ω_p*-def prob-space-measure-pmf*)

lemma *integrable*-Ω:
 fixes f :: $((\textit{nat} \times \textit{nat}) \Rightarrow (\textit{nat list})) \Rightarrow \textit{real}$
 shows *integrable* Ω_p f
 unfolding Ω_p*-def* Ω*-def*
 by (*rule integrable-measure-pmf-finite*, *auto intro*:*finite-PiE simp*:*set-prod-pmf*)

lemma *sketch-rv-exp*:
 assumes $i_2 < s_2$
 assumes $i_1 \in \{0..{<}s_1\}$
 shows Ω.*expectation* $(\lambda\omega. \ \textit{sketch-rv} \ (\omega \ (i_1, \ i_2))) = \textit{real-of-rat} \ (F \ 2 \ as) * ((\textit{real} \ p)^2 - 1)$
proof −
 have Ω.*expectation* $(\lambda\omega. \ (\textit{sketch-rv} \ (\omega \ (i_1, \ i_2))) :: \textit{real}) = \textit{expectation sketch-rv}$
  using *integrable*-Ω *integrable-M assms*
  unfolding Ω*-def* Ω_p*-def M-def*
  by (*subst expectation-Pi-pmf-slice*, *auto*)
 also have ... $= (\textit{real-of-rat} \ (F \ 2 \ as)) * ((\textit{real} \ p)^2 - 1)$
  using *exp-sketch-rv* **by** *simp*
 finally show *?thesis* **by** *simp*
qed

lemma *sketch-rv-var*:
 assumes $i_2 < s_2$
 assumes $i_1 \in \{0..{<}s_1\}$
 shows Ω.*variance* $(\lambda\omega. \ \textit{sketch-rv} \ (\omega \ (i_1, \ i_2))) \leq 2 * (\textit{real-of-rat} \ (F \ 2 \ as))^2 * ((\textit{real} \ p)^2 - 1)^2$
proof −
 have Ω.*variance* $(\lambda\omega. \ (\textit{sketch-rv} \ (\omega \ (i_1, \ i_2)) :: \textit{real})) = \textit{variance sketch-rv}$
  using *integrable*-Ω *integrable-M assms*
  unfolding Ω*-def* Ω_p*-def M-def*

by (*subst variance-prod-pmf-slice*, *auto*)
also have ... $\leq$ *2* $*$ (*real-of-rat* (*F 2 as*))^2 $*$ ((*real p*)^2 $-$ *1*)^2
using *var-sketch-rv* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *mean-rv-exp*:
assumes $i < s_2$
shows Ω.*expectation* ($\lambda\omega$. *mean-rv* ω *i*) = *real-of-rat* (*F 2 as*)
proof $-$
have *a*:(*real p*)^2 > *1* **using** *p-gt-1* **by** *simp*

have Ω.*expectation* ($\lambda\omega$. *mean-rv* ω *i*) = ($\sum i_1$ = *0*..<s_1. Ω.*expectation* ($\lambda\omega$. *sketch-rv* (ω (i_1, *i*)))) / (((*real p*)^2 $-$ *1*) $*$ *real* s_1)
using *assms integrable-*Ω **by** (*simp add*:*mean-rv-def*)
also have ... = ($\sum i_1$ = *0*..<s_1. *real-of-rat* (*F 2 as*) $*$ ((*real p*)^2 $-$ *1*)) / (((*real p*)^2 $-$ *1*) $*$ *real* s_1)
using *sketch-rv-exp*[*OF assms*] **by** *simp*
also have ... = *real-of-rat* (*F 2 as*)
using *s1-gt-0 a* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *mean-rv-var*:
assumes $i < s_2$
shows Ω.*variance* ($\lambda\omega$. *mean-rv* ω *i*) $\leq$ (*real-of-rat* (δ $*$ *F 2 as*))^2 / *3*
proof $-$
have *a*: Ω.*indep-vars* (λ-. *borel*) (λi_1 *x*. *sketch-rv* (*x* (i_1, *i*))) {*0*..<s_1}
using *assms*
unfolding Ω_p*-def* Ω*-def*
by (*intro indep-vars-restrict-intro'*[**where** *f*=*fst*])
(*auto simp add*: *restrict-dfl-def case-prod-beta lessThan-atLeast0*)

have *p-sq-ne-1*: (*real p*)^*2* $\neq$ *1*
by (*metis p-gt-1 less-numeral-extra*(*4*) *of-nat-power one-less-power pos2 semiring-char-0-class.of-nat-eq-1-iff*)

have *s1-bound*: *6* / (*real-of-rat* δ)^2 $\leq$ *real* s_1
unfolding s_1*-def*
by (*metis* (*mono-tags*, *opaque-lifting*) *of-rat-ceiling of-rat-divide of-rat-numeral-eq of-rat-power real-nat-ceiling-ge*)

have Ω.*variance* ($\lambda\omega$. *mean-rv* ω *i*) = Ω.*variance* ($\lambda\omega$. $\sum i_1$ = *0*..<s_1. *sketch-rv* (ω (i_1, *i*))) / (((*real p*)^2 $-$ *1*) $*$ *real* s_1)^2
unfolding *mean-rv-def* **by** (*subst* Ω.*variance-divide*[*OF integrable-*Ω], *simp*)
also have ... = ($\sum i_1$ = *0*..<s_1. Ω.*variance* ($\lambda\omega$. *sketch-rv* (ω (i_1, *i*)))) / (((*real p*)^2 $-$ *1*) $*$ *real* s_1)^2
by (*subst* Ω.*var-sum-all-indep*[*OF - - integrable-*Ω *a*]) (*auto simp*: Ω*-def* Ω_p*-def*)
also have ... $\leq$ ($\sum i_1$ = *0*..<s_1. *2*$*$(*real-of-rat* (*F 2 as*)^*2*) $*$ ((*real p*)^2$-$*1*)^2) /

$(((real\ p)^2 - 1) * real\ s_1)^2$
  by (*rule divide-right-mono*, *rule sum-mono*[*OF sketch-rv-var*[*OF assms*]], *auto*)
  also have ... = $2 *$ (*real-of-rat* (*F 2 as*)^*2*) / *real* s_1
    using *p-sq-ne-1 s1-gt-0* **by** (*subst frac-eq-eq*, *auto simp*:*power2-eq-square*)
  also have ... $\leq 2 *$ (*real-of-rat* (*F 2 as*)^*2*) / (*6* / (*real-of-rat* $\delta)^2$)
    using *s1-gt-0* δ*-range* **by** (*intro divide-left-mono mult-pos-pos s1-bound*) *auto*
  also have ... = (*real-of-rat* ($\delta *$ *F 2 as*))^2 / *3*
    by (*simp add*:*of-rat-mult algebra-simps*)
  finally show *?thesis* **by** *simp*
qed

lemma *mean-rv-bounds*:
  assumes $i < s_2$
  shows Ω*.prob* {ω. *real-of-rat* $\delta *$ *real-of-rat* (*F 2 as*) < |*mean-rv* ω *i* $-$ *real-of-rat* (*F 2 as*)|} $\leq$ *1/3*
proof (*cases as* = [])
  case *True*
  then show *?thesis*
    using *assms* **by** (*subst mean-rv-def*, *subst sketch-rv-def*, *simp add*:*F-def*)
next
  case *False*
  hence *F 2 as* > *0* **using** *F-gr-0* **by** *auto*

  hence *a*: *0* < *real-of-rat* ($\delta *$ *F 2 as*)
    using δ*-range* **by** *simp*
  have [*simp*]: ($\lambda\omega$. *mean-rv* ω *i*) $\in$ *borel-measurable* Ω_p
    by (*simp add*:Ω*-def* Ω_p*-def*)
  have Ω*.prob* {ω. *real-of-rat* $\delta *$ *real-of-rat* (*F 2 as*) < |*mean-rv* ω *i* $-$ *real-of-rat* (*F 2 as*)|} $\leq$
      Ω*.prob* {ω. *real-of-rat* ($\delta *$ *F 2 as*) $\leq$ |*mean-rv* ω *i* $-$ *real-of-rat* (*F 2 as*)|}
    by (*rule* Ω*.pmf-mono*[*OF* Ω_p*-def*], *simp add*:*of-rat-mult*)
  also have ... $\leq$ Ω*.variance* ($\lambda\omega$. *mean-rv* ω *i*) / (*real-of-rat* ($\delta *$ *F 2 as*))^2
     using Ω*.Chebyshev-inequality*[**where** *a*=*real-of-rat* ($\delta *$ *F 2 as*) **and** *f*=$\lambda\omega$. *mean-rv* ω *i*,*simplified*]
       a prob-space-measure-pmf[**where** *p*=Ω] *mean-rv-exp*[*OF assms*] *integrable*-Ω
by *simp*
  also have ... $\leq$ ((*real-of-rat* ($\delta *$ *F 2 as*))^2/*3*) / (*real-of-rat* ($\delta *$ *F 2 as*))^2
    by (*rule divide-right-mono*, *rule mean-rv-var*[*OF assms*], *simp*)
  also have ... = *1/3* **using** *a* **by** *force*
  finally show *?thesis* **by** *blast*
qed

lemma *f2-alg-correct*$'$:
  $\mathcal{P}$(ω *in measure-pmf result*. |ω $-$ *F 2 as*| $\leq \delta *$ *F 2 as*) $\geq$ *1*$-$*of-rat* ε
proof $-$
  have *a*: Ω*.indep-vars* (λ-. *borel*) (λi ω. *mean-rv* ω *i*) {*0*..<s_2}
    using *s1-gt-0* **unfolding** Ω_p*-def* Ω*-def*
    by (*intro indep-vars-restrict-intro*$'$[**where** *f*=*snd*])
      (*auto simp*: Ω_p*-def* Ω*-def mean-rv-def restrict-dfl-def*)

have *b*: $-$ *18* $*$ *ln* (*real-of-rat* ε) $\leq$ *real* s_2
unfolding s_2*-def* **using** *of-nat-ceiling* **by** *auto*

have *1* $-$ *of-rat* $\varepsilon \leq \Omega$*.prob* {ω. |*median* s_2 (*mean-rv* ω) $-$ *real-of-rat* (*F 2 as*) | $\leq$ *of-rat* δ $*$ *of-rat* (*F 2 as*)}
using ε*-range* Ω*.median-bound-2*[*OF - a b*, **where** δ=*real-of-rat* δ $*$ *real-of-rat* (*F 2 as*)
and μ=*real-of-rat* (*F 2 as*)] *mean-rv-bounds*
by *simp*
also have ... = Ω*.prob* {ω. |*real-of-rat* (*result-rv* ω) $-$ *of-rat* (*F 2 as*) | $\leq$ *of-rat* δ $*$ *of-rat* (*F 2 as*)}
by (*simp add:result-rv-def median-restrict lessThan-atLeast0 median-rat*[*OF s2-gt-0*]
mean-rv-def sketch-rv-def of-rat-divide of-rat-sum of-rat-mult of-rat-diff of-rat-power)
also have ... = Ω*.prob* {ω. |*result-rv* ω $-$ *F 2 as*| $\leq$ δ $*$ *F 2 as*}
by (*simp add:of-rat-less-eq of-rat-mult*[*symmetric*] *of-rat-diff*[*symmetric*] *set-eq-iff*)
finally have Ω*.prob* {*y*. |*result-rv y* $-$ *F 2 as*| $\leq$ δ $*$ *F 2 as*} $\geq$ *1*$-$*of-rat* ε **by** *simp*
thus *?thesis* **by** (*simp add*: *distr* Ω_p*-def*)
qed

lemma *f2-exact-space-usage*$'$:
AE ω *in sketch* . *bit-count* (*encode-f2-state* ω) $\leq$ *f2-space-usage* (*n*, *length as*, ε, δ)
proof $-$
have *p* $\leq$ *2* $*$ *max n 3* + *2*
by (*subst p-def*, *rule prime-above-upper-bound*)
also have ... $\leq$ *2* $*$ *n* + *8*
by (*cases n* $\leq$ *2*, *simp-all*)
finally have *p-bound*: *p* $\leq$ *2* $*$ *n* + *8*
by *simp*
have *bit-count* (N_e *p*) $\leq$ *ereal* (*2* $*$ *log 2* (*real p* + *1*) + *1*)
by (*rule exp-golomb-bit-count*)
also have ... $\leq$ *ereal* (*2* $*$ *log 2* (*2* $*$ *real n* + *9*) + *1*)
using *p-bound* **by** *simp*
finally have *p-bit-count*: *bit-count* (N_e *p*) $\leq$ *ereal* (*2* $*$ *log 2* (*2* $*$ *real n* + *9*) + *1*)
by *simp*

have *a*: *bit-count* (*encode-f2-state* (s_1, s_2, *p*, *y*, $\lambda i \in \{..<s_1\} \times \{..<s_2\}$.
sum-list (*map* (*f2-hash p* (*y i*)) *as*))) $\leq$ *ereal* (*f2-space-usage* (*n*, *length as*, ε, δ))
if *a*:$y \in \{..<s_1\} \times \{..<s_2\} \rightarrow_E$ *bounded-degree-polynomials* (*mod-ring p*) *4* **for** *y*
proof $-$
have $y \in$ *extensional* ($\{..<s_1\} \times \{..<s_2\}$) **using** *a PiE-iff* **by** *blast*
hence *y-ext*: $y \in$ *extensional* (*set* (*List.product* [*0*..$<s_1$] [*0*..$<s_2$]))
by (*simp add:lessThan-atLeast0*)

have *h-bit-count-aux*: *bit-count* (P_e *p 4* (*y x*)) ≤ *ereal* (*4* + *4* ∗ *log 2* (*8* + *2* ∗ *real n*))
if *b*:*x* ∈ *set* (*List.product* [*0*..<s_1] [*0*..<s_2]) **for** *x*
proof −
have *y x* ∈ *bounded-degree-polynomials* (*Field.mod-ring p*) *4*
using *b a* **by** *force*
hence *bit-count* (P_e *p 4* (*y x*)) ≤ *ereal* (*real 4* ∗ (*log 2* (*real p*) + *1*))
by (*rule bounded-degree-polynomial-bit-count*[*OF p-gt-1*])
also have ... ≤ *ereal* (*real 4* ∗ (*log 2* (*8* + *2* ∗ *real n*) + *1*))
using *p-gt-0 p-bound* **by** *simp*
also have ... ≤ *ereal* (*4* + *4* ∗ *log 2* (*8* + *2* ∗ *real n*))
by *simp*
finally show *?thesis*
by *blast*
qed

have *h-bit-count*:
bit-count ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ P_e *p 4*) *y*) ≤ *ereal* (*real* s_1 ∗ *real* s_2 ∗ (*4* + *4* ∗ *log 2* (*8* + *2* ∗ *real n*)))
using *fun-bit-count-est*[**where** *e*=P_e *p 4*, *OF y-ext h-bit-count-aux*]
by *simp*

have *sketch-bit-count-aux*:
bit-count (I_e (*sum-list* (*map* (*f2-hash p* (*y x*)) *as*))) ≤ *ereal* (*1* + *2* ∗ *log 2* (*real* (*length as*) ∗ (*18* + *4* ∗ *real n*) + *1*)) (**is** *?lhs* ≤ *?rhs*)
if *x* ∈ {*0*..<s_1} × {*0*..<s_2} **for** *x*
proof −
have |*sum-list* (*map* (*f2-hash p* (*y x*)) *as*)| ≤ *sum-list* (*map* (*abs* ∘ (*f2-hash p* (*y x*))) *as*)
by (*subst map-map*[*symmetric*]) (*rule sum-list-abs*)
also have ... ≤ *sum-list* (*map* (λ-. (*int p*+*1*)) *as*)
by (*rule sum-list-mono*) (*simp add*:*p-gt-0*)
also have ... = *int* (*length as*) ∗ (*int p*+*1*)
by (*simp add*: *sum-list-triv*)
also have ... ≤ *int* (*length as*) ∗ (*9*+*2*∗(*int n*))
using *p-bound* **by** (*intro mult-mono*, *auto*)
finally have |*sum-list* (*map* (*f2-hash p* (*y x*)) *as*)| ≤ *int* (*length as*) ∗ (*9* + *2* ∗ *int n*) **by** *simp*
hence *?lhs* ≤ *ereal* (*2* ∗ *log 2* (*real-of-int* (*2*∗ (*int* (*length as*) ∗ (*9* + *2* ∗ *int n*)) + *1*)) + *1*)
by (*rule int-bit-count-est*)
also have ... = *?rhs* **by** (*simp add*:*algebra-simps*)
finally show *?thesis* **by** *simp*
qed

have
bit-count ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ I_e) (λ*i*∈{..<s_1} × {..<s_2}. *sum-list* (*map* (*f2-hash p* (*y i*)) *as*)))

$\leq$ *ereal* (*real* (*length* (*List.product* [*0*..<s_1] [*0*..<s_2]))) $*$ (*ereal* (*1* + *2* $*$ *log 2* (*real* (*length as*) $*$ (*18* + *4* $*$ *real n*) + *1*)))
by (*intro fun-bit-count-est*)
(*simp-all add*:*extensional-def lessThan-atLeast0 sketch-bit-count-aux del*:*f2-hash.simps*)
also have ... = *ereal* (*real* s_1 $*$ *real* s_2 $*$ (*1* + *2* $*$ *log 2* (*real* (*length as*) $*$ (*18* + *4* $*$ *real n*) + *1*)))
by *simp*
finally have *sketch-bit-count*:
bit-count ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ I_e) ($\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*y i*)) *as*))) $\leq$
ereal (*real* s_1 $*$ *real* s_2 $*$ (*1* + *2* $*$ *log 2* (*real* (*length as*) $*$ (*18* + *4* $*$ *real n*) + *1*))) **by** *simp*

have *bit-count* (*encode-f2-state* (s_1, s_2, *p*, *y*, $\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*y i*)) *as*))) $\leq$
bit-count (N_e s_1) + *bit-count* (N_e s_2) +*bit-count* (N_e *p*) +
bit-count ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ P_e *p 4*) *y*) +
bit-count ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ I_e) ($\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*y i*)) *as*)))
by (*simp add*:*Let-def* s_1*-def* s_2*-def encode-f2-state-def dependent-bit-count add.assoc*)
also have ... $\leq$ *ereal* (*2* $*$ *log 2* (*real* s_1 + *1*) + *1*) + *ereal* (*2* $*$ *log 2* (*real* s_2 + *1*) + *1*) + *ereal* (*2* $*$ *log 2* (*2* $*$ *real n* + *9*) + *1*) +
(*ereal* (*real* s_1 $*$ *real* s_2) $*$ (*4* + *4* $*$ *log 2* (*8* + *2* $*$ *real n*))) +
(*ereal* (*real* s_1 $*$ *real* s_2) $*$ (*1* + *2* $*$ *log 2* (*real* (*length as*) $*$ (*18* + *4* $*$ *real n*) + *1*)))
by (*intro add-mono exp-golomb-bit-count p-bit-count*, *auto intro*: *h-bit-count sketch-bit-count*)
also have ... = *ereal* (*f2-space-usage* (*n*, *length as*, ε, δ))
by (*simp add*:*distrib-left add.commute* s_1*-def*[*symmetric*] s_2*-def*[*symmetric*] *Let-def*)
finally show *bit-count* (*encode-f2-state* (s_1, s_2, *p*, *y*, $\lambda i \in \{..<s_1\} \times \{..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*y i*)) *as*))) $\leq$
ereal (*f2-space-usage* (*n*, *length as*, ε, δ))
by *simp*
qed

have *set-pmf* Ω = $\{..<s_1\} \times \{..<s_2\} \rightarrow_E$ *bounded-degree-polynomials* (*Field.mod-ring p*) *4*
by (*simp add*: Ω*-def set-prod-pmf*) (*simp add*: *space-def*)
thus *?thesis*
by (*simp add*:*mean-rv-alg-sketch AE-measure-pmf-iff del*:*f2-space-usage.simps*, *metis a*)
qed

end

Main results of this section:

theorem *f2-alg-correct*:

assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq$ {..<*n*}
defines $\Omega \equiv$ *fold* (λa *state. state* $\gg=$ *f2-update a*) *as* (*f2-init* δ ε *n*) $\gg=$ *f2-result*
shows $\mathcal{P}$(ω *in measure-pmf* Ω. $|\omega - F\ 2\ as| \leq \delta * F\ 2\ as$) $\geq$ 1$-$*of-rat* ε
using *f2-alg-correct'*[*OF assms*(*1*,*2*,*3*)] Ω*-def* **by** *auto*

theorem *f2-exact-space-usage*:
assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq$ {..<*n*}
defines $M \equiv$ *fold* (λa *state. state* $\gg=$ *f2-update a*) *as* (*f2-init* δ ε *n*)
shows *AE* ω *in M. bit-count* (*encode-f2-state* ω) $\leq$ *f2-space-usage* (*n*, *length as*, ε, δ)
using *f2-exact-space-usage'*[*OF assms*(*1*,*2*,*3*)]
by (*subst* (*asm*) *sketch-def*[*OF assms*(*1*,*2*,*3*)], *subst M-def*, *simp*)

theorem *f2-asymptotic-space-complexity*:
f2-space-usage $\in$ *O*[*at-top* $\times_F$ *at-top* $\times_F$ *at-right 0* $\times_F$ *at-right 0*](λ (*n*, *m*, ε, δ).

(*ln* (*1* / *of-rat* ε)) / (*of-rat* δ)^2 $*$ (*ln* (*real n*) + *ln* (*real m*)))
(**is** - $\in$ *O*[*?F*](*?rhs*))
proof $-$
define *n-of* :: *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *nat* **where** *n-of* = (λ(*n*, *m*, ε, δ). *n*)
define *m-of* :: *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *nat* **where** *m-of* = (λ(*n*, *m*, ε, δ). *m*)
define ε*-of* :: *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *rat* **where** ε*-of* = (λ(*n*, *m*, ε, δ). ε)
define δ*-of* :: *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *rat* **where** δ*-of* = (λ(*n*, *m*, ε, δ). δ)

define *g* **where** *g* = (λx. (*1*/ (*of-rat* (δ*-of x*))^2) $*$ (*ln* (*1* / *of-rat* (ε*-of x*))) $*$ (*ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*))))

have *evt*: ($\bigwedge x$.
0 < *real-of-rat* (δ*-of x*) $\wedge$ *0* < *real-of-rat* (ε*-of x*) $\wedge$
1/*real-of-rat* (δ*-of x*) $\geq \delta \wedge$ *1*/*real-of-rat* (ε*-of x*) $\geq \varepsilon \wedge$
real (*n-of x*) $\geq n \wedge$ *real* (*m-of x*) $\geq m \Longrightarrow P\ x$)
$\Longrightarrow$ *eventually P ?F* (**is** ($\bigwedge x$. *?prem x* $\Longrightarrow$ -) $\Longrightarrow$ -)
for δ ε *n m P*
apply (*rule eventually-mono*[**where** *P*=*?prem* **and** *Q*=*P*])
apply (*simp add*:ε*-of-def case-prod-beta'* δ*-of-def n-of-def m-of-def*)
apply (*intro eventually-conj eventually-prod1' eventually-prod2'*
sequentially-inf eventually-at-right-less inv-at-right-0-inf)
by (*auto simp add*:*prod-filter-eq-bot*)

have *unit-1*: (λ-. *1*) $\in$ *O*[*?F*](λx. *1* / (*real-of-rat* (δ*-of x*))^2)
using *one-le-power*
by (*intro landau-o.big-mono evt*[**where** δ=*1*], *auto simp add*:*power-one-over*[*symmetric*])

have *unit-2*: (λ-. *1*) $\in$ *O*[*?F*](λx. *ln* (*1* / *real-of-rat* (ε*-of x*)))
by (*intro landau-o.big-mono evt*[**where** ε=*exp 1*])

(*auto intro*!:*iffD2*[*OF ln-ge-iff*] *simp add*:*abs-ge-iff*)

have *unit-3*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*n-of x*))
by (*intro landau-o.big-mono evt*, *auto*)

have *unit-4*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*m-of x*))
by (*intro landau-o.big-mono evt*, *auto*)

have *unit-5*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)))
by (*auto intro*!: *landau-o.big-mono evt*[**where** *n*=*exp 1*])
(*metis abs-ge-self linorder-not-le ln-ge-iff not-exp-le-zero order.trans*)

have *unit-6*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro landau-sum-1 evt unit-5 iffD2*[*OF ln-ge-iff*], *auto*)

have *unit-7*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *1* / *real-of-rat* (ε*-of x*))
by (*intro landau-o.big-mono evt*[**where** ε=*1*], *auto*)

have *unit-8*: (λ-. *1*) ∈ *O*[*?F*](*g*)
unfolding *g-def* **by** (*intro landau-o.big-mult-1 unit-1 unit-2 unit-6*)

have *unit-9*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*n-of x*) ∗ *real* (*m-of x*))
by (*intro landau-o.big-mult-1 unit-3 unit-4*)

have (λ*x*. *6* ∗ (*1* / (*real-of-rat* (δ*-of x*))^2)) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2)
by (*subst landau-o.big.cmult-in-iff*, *simp-all*)
hence *l1*: (λ*x*. *real* (*nat* ⌈*6* / (δ*-of x*)^2⌉)) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ*-of x*))^2)
by (*intro landau-real-nat landau-rat-ceil*[*OF unit-1*]) (*simp-all add*:*of-rat-divide of-rat-power*)

have (λ*x*. − (*ln* (*real-of-rat* (ε*-of x*)))) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
by (*intro landau-o.big-mono evt*) (*subst ln-div*, *auto*)
hence *l2*: (λ*x*. *real* (*nat* ⌈− (*18* ∗ *ln* (*real-of-rat* (ε*-of x*)))⌉)) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
by (*intro landau-real-nat landau-ceil*[*OF unit-2*], *simp*)

have *l3-aux*: (λ*x*. *real* (*m-of x*) ∗ (*18* + *4* ∗ *real* (*n-of x*)) + *1*) ∈ *O*[*?F*](λ*x*. *real* (*n-of x*) ∗ *real* (*m-of x*))
by (*rule sum-in-bigo*[*OF -unit-9*], *subst mult.commute*)
(*intro landau-o.mult sum-in-bigo*, *auto simp*:*unit-3*)

have (λ*x*. *ln* (*real* (*m-of x*) ∗ (*18* + *4* ∗ *real* (*n-of x*)) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*) ∗ *real* (*m-of x*)))
apply (*rule landau-ln-2*[**where** *a*=*2*], *simp*, *simp*)
apply (*rule evt*[**where** *m*=*2* **and** *n*=*1*])
apply (*metis dual-order.trans mult-left-mono mult-of-nat-commute of-nat-0-le-iff verit-prod-simplify*(*1*))
using *l3-aux* **by** *simp*

also have (*λx. ln* (*real* (*n-of x*) ∗ *real* (*m-of x*))) ∈ *O*[*?F*](*λx. ln* (*real* (*n-of x*)) + *ln*(*real* (*m-of x*)))
by (*intro landau-o.big-mono evt*[**where** *m=1* **and** *n=1*], *auto simp add*:*ln-mult*)
finally have *l3*: (*λx. ln* (*real* (*m-of x*) ∗ (*18* + *4* ∗ *real* (*n-of x*)) + *1*)) ∈ *O*[*?F*](*λx. ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
using *landau-o.big-trans* **by** *simp*

have *l4*: (*λx. ln* (*8* + *2* ∗ *real* (*n-of x*))) ∈ *O*[*?F*](*λx. ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro landau-sum-1 evt*[**where** *n=2*] *landau-ln-2*[**where** *a=2*] *iffD2*[*OF ln-ge-iff*])
(*auto intro*!: *sum-in-bigo simp add*:*unit-3*)

have *l5*: (*λx. ln* (*9* + *2* ∗ *real* (*n-of x*))) ∈ *O*[*?F*](*λx. ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro landau-sum-1 evt*[**where** *n=2*] *landau-ln-2*[**where** *a=2*] *iffD2*[*OF ln-ge-iff*])
(*auto intro*!: *sum-in-bigo simp add*:*unit-3*)

have *l6*: (*λx. ln* (*real* (*nat* ⌈*6* / (*δ-of x*)^2⌉) + *1*)) ∈ *O*[*?F*](*g*)
unfolding *g-def*
by (*intro landau-o.big-mult-1 landau-ln-3 sum-in-bigo unit-6 unit-2 l1 unit-1*, *simp*)

have *l7*: (*λx. ln* (*9* + *2* ∗ *real* (*n-of x*))) ∈ *O*[*?F*](*g*)
unfolding *g-def*
by (*intro landau-o.big-mult-1′ unit-1 unit-2 l5*)

have *l8*: (*λx. ln* (*real* (*nat* ⌈− (*18* ∗ *ln* (*real-of-rat* (*ε-of x*)))⌉) + *1*)) ∈ *O*[*?F*](*g*)
unfolding *g-def*
by (*intro landau-o.big-mult-1 unit-6 landau-o.big-mult-1′ unit-1 landau-ln-3 sum-in-bigo l2 unit-2*) *simp*

have *l9*: (*λx. 5* + *4* ∗ *ln* (*8* + *2* ∗ *real* (*n-of x*)) / *ln 2* + *2* ∗ *ln* (*real* (*m-of x*) ∗ (*18* + *4* ∗ *real* (*n-of x*)) + *1*) / *ln 2*)
∈ *O*[*?F*](*λx. ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro sum-in-bigo*, *auto simp*: *l3 l4 unit-6*)

have *l10*: (*λx. real* (*nat* ⌈*6* / (*δ-of x*)^2⌉) ∗ *real* (*nat* ⌈− (*18* ∗ *ln* (*real-of-rat* (*ε-of x*)))⌉) ∗
(*5* + *4* ∗ *ln* (*8* + *2* ∗ *real* (*n-of x*)) / *ln 2* + *2* ∗ *ln*(*real* (*m-of x*) ∗ (*18* + *4* ∗ *real* (*n-of x*)) + *1*) / *ln 2*))
∈ *O*[*?F*](*g*)
unfolding *g-def* **by** (*intro landau-o.mult*, *auto simp*: *l1 l2 l9*)

have *f2-space-usage* = (*λx. f2-space-usage* (*n-of x*, *m-of x*, *ε-of x*, *δ-of x*))
by (*simp add*:*case-prod-beta′ n-of-def ε-of-def δ-of-def m-of-def*)
also have ... ∈ *O*[*?F*](*g*)
by (*auto intro*!:*sum-in-bigo simp*:*Let-def log-def l6 l7 l8 l10 unit-8*)

```
  also have ... = O[?F](?rhs)
    by (simp add:case-prod-beta' g-def n-of-def ε-of-def δ-of-def m-of-def)
  finally show ?thesis by simp
qed

end
```

9 Frequency Moment k

```
theory Frequency-Moment-k
  imports
    Frequency-Moments
    Landau-Ext
    Lp.Lp
    Median-Method.Median
    Product-PMF-Ext
begin
```

This section contains a formalization of the algorithm for the k-th frequency moment. It is based on the algorithm described in [1, §2.1].

```
type-synonym fk-state = nat × nat × nat × nat × (nat × nat ⇒ (nat × nat))

fun fk-init :: nat ⇒ rat ⇒ rat ⇒ nat ⇒ fk-state pmf where
  fk-init k δ ε n =
    do {
      let s₁ = nat ⌈3 * real k * n powr (1−1/real k) / (real-of-rat δ)²⌉;
      let s₂ = nat ⌈−18 * ln (real-of-rat ε)⌉;
      return-pmf (s₁, s₂, k, 0, (λ- ∈ {0..<s₁} × {0..<s₂}. (0,0)))
    }

fun fk-update :: nat ⇒ fk-state ⇒ fk-state pmf where
  fk-update a (s₁, s₂, k, m, r) =
    do {
     coins ← prod-pmf ({0..<s₁} × {0..<s₂}) (λ-. bernoulli-pmf (1/(real m+1)));
      return-pmf (s₁, s₂, k, m+1, λi ∈ {0..<s₁} × {0..<s₂}.
        if coins i then
          (a,0)
        else (
          let (x,l) = r i in (x, l + of-bool (x=a))
        )
      )
    }

fun fk-result :: fk-state ⇒ rat pmf where
  fk-result (s₁, s₂, k, m, r) =
    return-pmf (median s₂ (λi₂ ∈ {0..<s₂}.
      (∑ i₁∈{0..<s₁}. rat-of-nat (let t = snd (r (i₁, i₂)) + 1 in m * (t^k − (t −
1)^k))) / (rat-of-nat s₁))
    )
```

lemma *bernoulli-pmf-1*: *bernoulli-pmf 1 = return-pmf True*
by (*rule pmf-eqI*, *simp add*:*indicator-def*)

fun *fk-space-usage* :: (*nat* × *nat* × *nat* × *rat* × *rat*) ⇒ *real* **where**
fk-space-usage (*k*, *n*, *m*, *ε*, *δ*) = (
let s_1 = *nat* ⌈*3*∗*real k*∗ (*real n*) *powr* (*1*−*1*/ *real k*) / (*real-of-rat δ*)^2 ⌉ *in*
let s_2 = *nat* ⌈−(*18* ∗ *ln* (*real-of-rat ε*))⌉ *in*
4 +
2 ∗ *log 2* (s_1 + *1*) +
2 ∗ *log 2* (s_2 + *1*) +
2 ∗ *log 2* (*real k* + *1*) +
2 ∗ *log 2* (*real m* + *1*) +
s_1 ∗ s_2 ∗ (*2* + *2* ∗ *log 2* (*real n*+*1*) + *2* ∗ *log 2* (*real m*+*1*)))

definition *encode-fk-state* :: *fk-state* ⇒ *bool list option* **where**
encode-fk-state =
N_e $\Join_e$ (λs_1.
N_e $\Join_e$ (λs_2.
N_e $\times_e$
N_e $\times_e$
(*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ (N_e $\times_e$ N_e))))

lemma *inj-on encode-fk-state* (*dom encode-fk-state*)
proof −
have *is-encoding encode-fk-state*
by (*simp add*:*encode-fk-state-def*)
(*intro dependent-encoding exp-golomb-encoding fun-encoding*)

thus *?thesis* **by** (*rule encoding-imp-inj*)
qed

This is an intermediate non-parallel form *fk-update* used only in the correctness proof.

fun *fk-update-2* :: *'a* ⇒ (*nat* × *'a* × *nat*) ⇒ (*nat* × *'a* × *nat*) *pmf* **where**
fk-update-2 a (*m*,*x*,*l*) =
do {
coin ← *bernoulli-pmf* (*1*/(*real m*+*1*));
return-pmf (*m*+*1*,*if coin then* (*a*,*0*) *else* (*x*, *l* + *of-bool* (*x*=*a*)))
}

definition *sketch* **where** *sketch as i* = (*as* ! *i*, *count-list* (*drop* (*i*+*1*) *as*) (*as* ! *i*))

lemma *fk-update-2-distr*:
assumes *as* ≠ []
shows *fold* (*λx s. s* ⤜ *fk-update-2 x*) *as* (*return-pmf* (*0*,*0*,*0*)) =
pmf-of-set {..<*length as*} ⤜ (*λk. return-pmf* (*length as*, *sketch as k*))
using *assms*
proof (*induction as rule*:*rev-nonempty-induct*)

case (*single x*)
show *?case* **using** *single*
by (*simp add:bind-return-pmf pmf-of-set-singleton bernoulli-pmf-1 lessThan-def sketch-def*)
next
case (*snoc x xs*)
let *?h* = (λ*xs k. count-list* (*drop* (*Suc k*) *xs*) (*xs* ! *k*))
let *?q* = (λ*xs k.* (*length xs*, *sketch xs k*))

have *non-empty*: {..<*Suc* (*length xs*)} ≠ {} {..<*length xs*} ≠ {} **using** *snoc* **by** *auto*

have *fk-update-2-eta*:*fk-update-2 x* = (λ*a. fk-update-2 x* (*fst a*, *fst* (*snd a*), *snd* (*snd a*)))
by *auto*

have *pmf-of-set* {..<*length xs*} ⤜ (λ*k. bernoulli-pmf* (*1* / (*real* (*length xs*) + *1*)) ⤜
(λ*coin. return-pmf* (*if coin then length xs else k*))) =
bernoulli-pmf (*1* / (*real* (*length xs*) + *1*)) ⤜ (λ*y. pmf-of-set* {..<*length xs*} ⤜
(λ*k. return-pmf* (*if y then length xs else k*)))
by (*subst bind-commute-pmf*, *simp*)
also have ... = *pmf-of-set* {..<*length xs* + *1*}
using *snoc*(*1*) *non-empty*
by (*intro pmf-eqI*, *simp add*: *pmf-bind measure-pmf-of-set*)
(*simp add:indicator-def algebra-simps frac-eq-eq*)
finally have *b*: *pmf-of-set* {..<*length xs*} ⤜ (λ*k. bernoulli-pmf* (*1* / (*real* (*length xs*) + *1*)) ⤜
(λ*coin. return-pmf* (*if coin then length xs else k*))) = *pmf-of-set* {..<*length xs* +*1*} **by** *simp*

have *fold* (λ*x s.* (*s* ⤜ *fk-update-2 x*)) (*xs*@[*x*]) (*return-pmf* (*0*,*0*,*0*)) =
(*pmf-of-set* {..<*length xs*} ⤜ (λ*k. return-pmf* (*length xs*, *sketch xs k*))) ⤜ *fk-update-2 x*
using *snoc* **by** (*simp add:case-prod-beta′*)
also have ... = (*pmf-of-set* {..<*length xs*} ⤜ (λ*k. return-pmf* (*length xs*, *sketch xs k*))) ⤜
(λ(*m*,*a*,*l*). *bernoulli-pmf* (*1* / (*real m* + *1*)) ⤜ (λ*coin.*
return-pmf (*m* + *1*, *if coin then* (*x*, *0*) *else* (*a*, (*l* + *of-bool* (*a* = *x*))))))
by (*subst fk-update-2-eta*, *subst fk-update-2.simps*, *simp add:case-prod-beta′*)
also have ... = *pmf-of-set* {..<*length xs*} ⤜ (λ*k. bernoulli-pmf* (*1* / (*real* (*length xs*) + *1*)) ⤜
(λ*coin. return-pmf* (*length xs* + *1*, *if coin then* (*x*, *0*) *else* (*xs* ! *k*, *?h xs k* + *of-bool* (*xs* ! *k* = *x*)))))
by (*subst bind-assoc-pmf*, *simp add*: *bind-return-pmf sketch-def*)
also have ... = *pmf-of-set* {..<*length xs*} ⤜ (λ*k. bernoulli-pmf* (*1* / (*real* (*length xs*) + *1*)) ⤜
(λ*coin. return-pmf* (*if coin then length xs else k*) ⤜ (λ*k′. return-pmf* (*?q*

$(xs@[x])\ k'))))$
using *non-empty*
by (*intro bind-pmf-cong*, *auto simp add:bind-return-pmf nth-append count-list-append sketch-def*)
also have ... = *pmf-of-set* $\{..<length\ xs\} \gg= (\lambda k.\ bernoulli\text{-}pmf\ (1\ /\ (real\ (length\ xs)\ +\ 1)) \gg=$
$(\lambda coin.\ return\text{-}pmf\ (if\ coin\ then\ length\ xs\ else\ k))) \gg= (\lambda k'.\ return\text{-}pmf\ (?q\ (xs@[x])\ k'))$
by (*subst bind-assoc-pmf*, *subst bind-assoc-pmf*, *simp*)
also have ... = *pmf-of-set* $\{..<length\ (xs@[x])\} \gg= (\lambda k'.\ return\text{-}pmf\ (?q\ (xs@[x])\ k'))$
by (*subst b*, *simp*)
finally show *?case* **by** *simp*
qed

context
fixes $\varepsilon\ \delta$ :: *rat*
fixes $n\ k$ :: *nat*
fixes *as*
assumes *k-ge-1*: $k \geq 1$
assumes ε*-range*: $\varepsilon \in \{0<..<1\}$
assumes δ*-range*: $\delta > 0$
assumes *as-range*: $set\ as \subseteq \{..<n\}$
begin

definition s_1 **where** $s_1 = nat\ \lceil 3 * real\ k * (real\ n)\ powr\ (1-1/real\ k)\ /\ (real\text{-}of\text{-}rat\ \delta)^2 \rceil$
definition s_2 **where** $s_2 = nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon))\rceil$

definition $M_1 = \{(u,\ v).\ v < count\text{-}list\ as\ u\}$
definition $\Omega_1 = measure\text{-}pmf\ (pmf\text{-}of\text{-}set\ M_1)$

definition $M_2 = prod\text{-}pmf\ (\{0..<s_1\} \times \{0..<s_2\})\ (\lambda\text{-}.\ pmf\text{-}of\text{-}set\ M_1)$
definition $\Omega_2 = measure\text{-}pmf\ M_2$

interpretation *prob-space* Ω_1
unfolding Ω_1*-def* **by** (*simp add:prob-space-measure-pmf*)

interpretation Ω_2:*prob-space* Ω_2
unfolding Ω_2*-def* **by** (*simp add:prob-space-measure-pmf*)

lemma *split-space*: $(\sum a \in M_1.\ f\ (snd\ a)) = (\sum u \in set\ as.\ (\sum v \in \{0..<count\text{-}list\ as\ u\}.\ f\ v))$
proof −
define A **where** $A = (\lambda u.\ \{u\} \times \{v.\ v < count\text{-}list\ as\ u\})$

have *a*: *inj-on snd* $(A\ x)$ **for** x
by (*simp add:A-def inj-on-def*)

have $\bigwedge u\ v.\ u < \mathit{count\text{-}list\ as\ v} \Longrightarrow v \in \mathit{set\ as}$
  by (*subst count-list-gr-1*, *force*)
hence $M_1 = \bigcup\ (A\ `\ \mathit{set\ as})$
  by (*auto simp add*:*set-eq-iff A-def* M_1*-def*)
hence $(\sum a{\in}M_1.\ f\ (\mathit{snd}\ a)) = \mathit{sum}\ (f \circ \mathit{snd})\ \ (\bigcup\ (A\ `\ \mathit{set\ as}))$
  by (*intro sum.cong*, *auto*)
also have ... = *sum* (λ*x*. *sum* (*f* ∘ *snd*) (*A x*)) (*set as*)
  by (*rule sum.UNION-disjoint*, *simp*, *simp add*:*A-def*, *simp add*:*A-def*, *blast*)
also have ... = *sum* (λ*x*. *sum f* (*snd* ‘ *A x*)) (*set as*)
  by (*intro sum.cong*, *auto simp add*:*sum.reindex*[*OF a*])
also have ... = $(\sum u \in \mathit{set\ as}.\ (\sum v \in \{0..<\mathit{count\text{-}list\ as\ u}\}.\ f\ v))$
  unfolding *A-def* **by** (*intro sum.cong*, *auto*)
finally show *?thesis* **by** *blast*
qed

lemma
  assumes *as* ≠ []
  shows *fin-space*: *finite* M_1
    and *non-empty-space*: $M_1 \neq \{\}$
    and *card-space*: *card* M_1 = *length as*
proof −
  have $M_1 \subseteq \mathit{set\ as} \times \{k.\ k < \mathit{length\ as}\}$
  proof (*rule subsetI*)
    fix *x*
    assume *a*:$x \in M_1$
    have *fst x* ∈ *set as*
      using *a* **by** (*simp add*:*case-prod-beta count-list-gr-1* M_1*-def*)
    moreover have *snd x* < *length as*
      using *a count-le-length order-less-le-trans*
      by (*simp add*:*case-prod-beta* M_1*-def*) *fast*
    ultimately show *x* ∈ *set as* × {*k*. *k* < *length as*}
      by (*simp add*:*mem-Times-iff*)
  qed
  thus *fin-space*: *finite* M_1
    using *finite-subset* **by** *blast*

  have $(\mathit{as}\ !\ 0,\ 0) \in M_1$
    using *assms*(*1*) **unfolding** M_1*-def*
   by (*simp*, *metis count-list-gr-1 gr0I length-greater-0-conv not-one-le-zero nth-mem*)
  thus $M_1 \neq \{\}$ **by** *blast*

  show *card* M_1 = *length as*
    using *fin-space split-space*[**where** *f*=λ-. (*1*::*nat*)]
    by (*simp add*:*sum-count-set*[**where** *X*=*set as* **and** *xs*=*as*, *simplified*])
qed

lemma
  assumes *as* ≠ []
  shows *integrable-1*: *integrable* Ω_1 (*f* :: - ⇒ *real*) **and**

integrable-2: *integrable* Ω_2 (*g* :: - ⇒ *real*)
proof –
have *fin-omega*: *finite* (*set-pmf* (*pmf-of-set* M_1))
using *fin-space*[*OF assms*] *non-empty-space*[*OF assms*] **by** *auto*
thus *integrable* Ω_1 *f*
unfolding Ω_1*-def*
by (*rule integrable-measure-pmf-finite*)

have *finite* (*set-pmf* M_2)
unfolding M_2*-def* **using** *fin-omega*
by (*subst set-prod-pmf*) (*auto intro*:*finite-PiE*)

thus *integrable* Ω_2 *g*
unfolding Ω_2*-def* **by** (*intro integrable-measure-pmf-finite*)
qed

lemma *sketch-distr*:
assumes *as* ≠ []
shows *pmf-of-set* {..<*length as*} ≫= (λ*k*. *return-pmf* (*sketch as k*)) = *pmf-of-set* M_1
proof –
have *x* < *y* ⟹ *y* < *length as* ⟹
count-list (*drop* (*y*+*1*) *as*) (*as* ! *y*) < *count-list* (*drop* (*x*+*1*) *as*) (*as* ! *y*) **for** *x y*
by (*intro count-list-lt-suffix suffix-drop-drop*, *simp-all*)
(*metis Suc-diff-Suc diff-Suc-Suc diff-add-inverse lessI less-natE*)
hence *a1*: *inj-on* (*sketch as*) {*k*. *k* < *length as*}
unfolding *sketch-def* **by** (*intro inj-onI*) (*metis Pair-inject mem-Collect-eq nat-neq-iff*)

have *x* < *length as* ⟹ *count-list* (*drop* (*x*+*1*) *as*) (*as* ! *x*) < *count-list as* (*as* ! *x*) **for** *x*
by (*rule count-list-lt-suffix*, *auto simp add*:*suffix-drop*)
hence *sketch as* ‘ {*k*. *k* < *length as*} ⊆ M_1
by (*intro image-subsetI*, *simp add*:*sketch-def* M_1*-def*)
moreover have *card* M_1 ≤ *card* (*sketch as* ‘ {*k*. *k* < *length as*})
by (*simp add*: *card-space*[*OF assms*(*1*)] *card-image*[*OF a1*])
ultimately have *sketch as* ‘ {*k*. *k* < *length as*} = M_1
using *fin-space*[*OF assms*(*1*)] **by** (*intro card-seteq*, *simp-all*)
hence *bij-betw* (*sketch as*) {*k*. *k* < *length as*} M_1
using *a1* **by** (*simp add*:*bij-betw-def*)
hence *map-pmf* (*sketch as*) (*pmf-of-set* {*k*. *k* < *length as*}) = *pmf-of-set* M_1
using *assms* **by** (*intro map-pmf-of-set-bij-betw*, *auto*)
thus *?thesis* **by** (*simp add*: *sketch-def map-pmf-def lessThan-def*)
qed

lemma *fk-update-distr*:
fold (λ*x s*. *s* ≫= *fk-update x*) *as* (*fk-init k* δ ε *n*) =
prod-pmf ({*0*..<s_1} × {*0*..<s_2}) (λ-. *fold* (λ*x s*. *s* ≫= *fk-update-2 x*) *as* (*return-pmf* (*0*,*0*,*0*)))

$\gg=$ (λx. *return-pmf* (s_1,s_2,*k*, *length as*, $\lambda i\in\{0..<s_1\}\times\{0..<s_2\}$. *snd* (*x i*)))
proof (*induction as rule*:*rev-induct*)
case *Nil*
then show *?case*
by (*auto simp*:*Let-def* s_1-*def*[*symmetric*] s_2-*def*[*symmetric*] *bind-return-pmf*)
next
case (*snoc x xs*)

have *fk-update-2-eta*:*fk-update-2 x* = (λa. *fk-update-2 x* (*fst a*, *fst* (*snd a*), *snd* (*snd a*)))
by *auto*

have *a*: *fk-update x* (s_1, s_2, *k*, *length xs*, $\lambda i\in\{0..<s_1\} \times \{0..<s_2\}$. *snd* (*f i*)) =
prod-pmf ($\{0..<s_1\} \times \{0..<s_2\}$) ($\lambda i$. *fk-update-2 x* (*f i*)) $\gg=$
(λa. *return-pmf* (s_1,s_2, *k*, *Suc* (*length xs*), $\lambda i\in\{0..<s_1\} \times \{0..<s_2\}$. *snd* (*a i*)))
if *b*: $f \in$ *set-pmf* (*prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$)
(λ-. *fold* (λa *s*. *s* $\gg=$ *fk-update-2 a*) *xs* (*return-pmf* (*0*, *0*, *0*)))) **for** *f*
proof −
have *c*:*fst* (*f i*) = *length xs* **if** *d*:$i \in \{0..<s_1\} \times\{0..<s_2\}$ **for** *i*
proof (*cases xs* = [])
case *True*
then show *?thesis* **using** *b d* **by** (*simp add*: *set-Pi-pmf*)
next
case *False*
hence {..<*length xs*} ≠ {} **by** *force*
thus *?thesis* **using** *b d*
by (*simp add*:*set-Pi-pmf fk-update-2-distr*[*OF False*] *PiE-dflt-def*) *force*
qed
show *?thesis*
apply (*subst fk-update-2-eta*, *subst fk-update-2.simps*, *simp*)
apply (*simp add*: *Pi-pmf-bind-return*[**where** *d'*=*undefined*] *bind-assoc-pmf*)
apply (*rule bind-pmf-cong*, *simp add*:*c cong*:*Pi-pmf-cong*)
by (*auto simp add*:*bind-return-pmf case-prod-beta*)
qed

have *fold* (λx *s*. *s* $\gg=$ *fk-update x*) (*xs* @ [*x*]) (*fk-init k* δ ε *n*) =
prod-pmf ($\{0..<s_1\} \times \{0..<s_2\}$) ($\lambda$-. *fold* ($\lambda x$ *s*. *s* $\gg=$ *fk-update-2 x*) *xs* (*return-pmf* (*0*,*0*,*0*)))
$\gg=$ ($\lambda\omega$. *return-pmf* (s_1,s_2,*k*, *length xs*, $\lambda i\in\{0..<s_1\}\times\{0..<s_2\}$. *snd* ($\omega$ *i*)) $\gg=$ *fk-update x*)
using *snoc*
by (*simp add*:*restrict-def bind-assoc-pmf del*:*fk-init.simps*)
also have ... = *prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$)
(λ-. *fold* (λa *s*. *s* $\gg=$ *fk-update-2 a*) *xs* (*return-pmf* (*0*, *0*, *0*))) $\gg=$
(λf. *prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$) ($\lambda i$. *fk-update-2 x* (*f i*)) $\gg=$
(λa. *return-pmf* (s_1, s_2, *k*, *Suc* (*length xs*), $\lambda i\in\{0..<s_1\} \times \{0..<s_2\}$. *snd* (*a i*))))
using *a*
by (*intro bind-pmf-cong*, *simp-all add*:*bind-return-pmf del*:*fk-update.simps*)

```
  also have ... = prod-pmf ({0..<s₁} × {0..<s₂})
    (λ-. fold (λa s. s ≫= fk-update-2 a) xs (return-pmf (0, 0, 0))) ≫=
    (λf. prod-pmf ({0..<s₁} × {0..<s₂}) (λi. fk-update-2 x (f i))) ≫=
     (λa. return-pmf (s₁, s₂, k, Suc (length xs), λi∈{0..<s₁} × {0..<s₂}. snd (a
i)))
    by (simp add:bind-assoc-pmf)
  also have ... = (prod-pmf ({0..<s₁} × {0..<s₂})
    (λ-. fold (λa s. s ≫= fk-update-2 a) (xs@[x]) (return-pmf (0,0,0)))
     ≫= (λa. return-pmf (s₁,s₂,k, length (xs@[x]), λi∈{0..<s₁}×{0..<s₂}. snd (a
i))))
    by (simp, subst Pi-pmf-bind, auto)

  finally show ?case by blast
qed

lemma power-diff-sum:
  fixes a b :: 'a :: {comm-ring-1,power}
  assumes k > 0
  shows a^k − b^k = (a−b) ∗ (∑ i = 0..<k. a ^ i ∗ b ^ (k − 1 − i)) (is ?lhs =
?rhs)
proof −
  have insert-lb: m < n ⟹ insert m {Suc m..<n} = {m..<n} for m n :: nat
    by auto

  have ?rhs = sum (λi. a ∗ (a^i ∗ b^(k−1−i))) {0..<k} −
    sum (λi. b ∗ (a^i ∗ b^(k−1−i))) {0..<k}
    by (simp add: sum-distrib-left[symmetric] algebra-simps)
  also have ... = sum ((λi. (a^i ∗ b^(k−i))) ∘ (λi. i+1)) {0..<k} −
    sum (λi. (a^i ∗ (b^(1+(k−1−i))))) {0..<k}
    by (simp add:algebra-simps)
  also have ... = sum ((λi. (a^i ∗ b^(k−i))) ∘ (λi. i+1)) {0..<k} −
    sum (λi. (a^i ∗ b^(k−i))) {0..<k}
    by (intro arg-cong2[where f=(−)] sum.cong arg-cong2[where f=(∗)]
        arg-cong2[where f=(λx y. x ^ y)]) auto
  also have ... = sum (λi. (a^i ∗ b^(k−i))) (insert k {1..<k}) −
    sum (λi. (a^i ∗ b^(k−i))) (insert 0 {Suc 0..<k})
    using assms
    by (subst sum.reindex[symmetric], simp, subst insert-lb, auto)
  also have ... = ?lhs
    by simp
  finally show ?thesis by presburger
qed

lemma power-diff-est:
  assumes k > 0
  assumes (a :: real) ≥ b
  assumes b ≥ 0
  shows a^k −b^k ≤ (a−b) ∗ k ∗ a^(k−1)
proof −
```

have $\bigwedge i.\ i < k \Longrightarrow a\ \widehat{}\ i * b\ \widehat{}\ (k - 1 - i) \le a\ \widehat{}\ i * a\ \widehat{}\ (k{-}1{-}i)$
  using *assms* **by** (*intro mult-left-mono power-mono*) *auto*
also have $\bigwedge i.\ i < k \Longrightarrow a\ \widehat{}\ i * a\ \widehat{}\ (k - 1 - i) = a\ \widehat{}\ (k - Suc\ 0)$
  using *assms*(*1*) **by** (*subst power-add*[*symmetric*], *simp*)
finally have *a*: $\bigwedge i.\ i < k \Longrightarrow a\ \widehat{}\ i * b\ \widehat{}\ (k - 1 - i) \le a\ \widehat{}\ (k - Suc\ 0)$
  by *blast*
have $a\widehat{}k - b\widehat{}k = (a{-}b) * (\sum i = 0..{<}k.\ a\ \widehat{}\ i * b\ \widehat{}\ (k - 1 - i))$
  by (*rule power-diff-sum*[*OF assms*(*1*)])
also have ... $\le (a{-}b) * (\sum i = 0..{<}k.\ \ a\widehat{}(k{-}1))$
  using *a assms* **by** (*intro mult-left-mono sum-mono*, *auto*)
also have ... $= (a{-}b) * (k * a\widehat{}(k{-}Suc\ 0))$
  by *simp*
finally show *?thesis* **by** *simp*
qed

Specialization of the Hoelder inquality for sums.

lemma *Holder-inequality-sum*:
  assumes $p > (0{::}real)$ $q > 0$ $1/p + 1/q = 1$
  assumes *finite A*
  shows $|\sum x{\in}A.\ f\ x * g\ x| \le (\sum x{\in}A.\ |f\ x|\ powr\ p)\ powr\ (1/p) * (\sum x{\in}A.\ |g\ x|$
$powr\ q)\ powr\ (1/q)$
proof −
  have $|LINT\ x|count\text{-}space\ A.\ f\ x * g\ x| \le$
    $(LINT\ x|count\text{-}space\ A.\ |f\ x|\ powr\ p)\ powr\ (1\ /\ p) *$
    $(LINT\ x|count\text{-}space\ A.\ |g\ x|\ powr\ q)\ powr\ (1\ /\ q)$
    using *assms integrable-count-space*
    by (*intro Lp.Holder-inequality*, *auto*)
  thus *?thesis*
    using *assms* **by** (*simp add*: *lebesgue-integral-count-space-finite*[*symmetric*])
qed

lemma *real-count-list-pos*:
  assumes $x \in set\ as$
  shows *real* (*count-list as x*) > 0
  using *count-list-gr-1 assms* **by** *force*

lemma *fk-estimate*:
  assumes $as \ne []$
  shows $length\ as * of\text{-}rat\ (F\ (2{*}k{-}1)\ as) \le n\ powr\ (1 - 1\ /\ real\ k) * (of\text{-}rat\ (F$
$k\ as))\widehat{}2$
  (**is** $?lhs \le ?rhs$)
proof (*cases* $k \ge 2$)
  case *True*
  define *M* **where** $M = Max\ (count\text{-}list\ as\ `\ set\ as)$
  have $M \in count\text{-}list\ as\ `\ set\ as$
    unfolding *M-def* **using** *assms* **by** (*intro Max-in*, *auto*)
  then obtain *m* **where** *m-in*: $m \in set\ as$ **and** *m-def*: $M = count\text{-}list\ as\ m$
    by *blast*

```
  have a: real M > 0 using m-in count-list-gr-1 by (simp add:m-def, force)
  have b: 2∗k−1 = (k−1) + k by simp

  have 0 < real (count-list as m)
    using m-in count-list-gr-1 by force
  hence M powr k = real (count-list as m) ^ k
    by (simp add: powr-realpow m-def)
  also have ... ≤ (∑x∈set as. real (count-list as x) ^ k)
    using m-in by (intro member-le-sum, simp-all)
  also have ... ≤ real-of-rat (F k as)
    by (simp add:F-def of-rat-sum of-rat-power)
  finally have d: M powr k ≤ real-of-rat (F k as) by simp

  have e: 0 ≤ real-of-rat (F k as)
    using F-gr-0[OF assms(1)] by (simp add: order-le-less)

  have real (k − 1) / real k + 1 = real (k − 1) / real k + real k / real k
    using assms True by simp
  also have ... = real (2 ∗ k − 1) / real k
    using b by (subst add-divide-distrib[symmetric], force)
  finally have f: real (k − 1) / real k + 1 = real (2 ∗ k − 1) / real k
    by blast

  have real-of-rat (F (2∗k−1) as) =
    (∑x∈set as. real (count-list as x) ^(k − 1) ∗ real (count-list as x) ^ k)
     using b by (simp add:F-def of-rat-sum sum-distrib-left of-rat-mult power-add
of-rat-power)
  also have ... ≤ (∑x∈set as. real M ^(k − 1) ∗ real (count-list as x) ^ k)
   by (intro sum-mono mult-right-mono power-mono of-nat-mono) (auto simp:M-def)
  also have ... = M powr (k−1) ∗ of-rat (F k as) using a
   by (simp add:sum-distrib-left F-def of-rat-mult of-rat-sum of-rat-power powr-realpow)
  also have ... = (M powr k) powr (real (k − 1) / real k) ∗ of-rat (F k as) powr 1
    using e by (simp add:powr-powr)
   also have ... ≤ (real-of-rat (F k as)) powr ((k−1)/k) ∗ (real-of-rat (F k as)
powr 1)
    using d by (intro mult-right-mono powr-mono2, auto)
  also have ... = (real-of-rat (F k as)) powr ((2∗k−1) / k)
    by (subst powr-add[symmetric], subst f, simp)
 finally have a: real-of-rat (F (2∗k−1) as) ≤ (real-of-rat (F k as)) powr ((2∗k−1)
/ k)
    by blast

  have g: card (set as) ≤ n
    using card-mono[OF - as-range] by simp

  have length as = abs (sum (λx. real (count-list as x)) (set as))
    by (subst of-nat-sum[symmetric], simp add: sum-count-set)
  also have ... ≤ card (set as) powr ((k−Suc 0)/k) ∗
    (sum (λx. |real (count-list as x)| powr k) (set as)) powr (1/k)
```

using *assms True*
by (*intro Holder-inequality-sum*[**where** *p=k/(k−1)* **and** *q=k* **and** *f=λ-.1*, *simplified*])
(*auto simp add:algebra-simps add-divide-distrib*[*symmetric*])
also have ... = (*card* (*set as*)) *powr* ((*k−1*) / *real k*) ∗ *of-rat* (*F k as*) *powr* (*1*/ *k*)
using *real-count-list-pos*
by (*simp add:F-def of-rat-sum of-rat-power powr-realpow*)
also have ... = (*card* (*set as*)) *powr* (*1* − *1* / *real k*) ∗ *of-rat* (*F k as*) *powr* (*1*/ *k*)
using *k-ge-1*
by (*subst of-nat-diff*[*OF k-ge-1*], *subst diff-divide-distrib*, *simp*)
also have ... ≤ *n powr* (*1* − *1* / *real k*) ∗ *of-rat* (*F k as*) *powr* (*1*/ *k*)
using *k-ge-1 g*
by (*intro mult-right-mono powr-mono2*, *auto*)
finally have *h*: *length as* ≤ *n powr* (*1* − *1* / *real k*) ∗ *of-rat* (*F k as*) *powr* (*1*/*real k*)
by *blast*

have *i*:*1* / *real k* + *real* (*2* ∗ *k* − *1*) / *real k* = *real 2*
using *True* **by** (*subst add-divide-distrib*[*symmetric*], *simp-all add:of-nat-diff*)

have *?lhs* ≤ *n powr* (*1* − *1*/*k*) ∗ *of-rat* (*F k as*) *powr* (*1*/*k*) ∗ (*of-rat* (*F k as*)) *powr* ((*2*∗*k*−*1*) / *k*)
using *a h F-ge-0* **by** (*intro mult-mono mult-nonneg-nonneg*, *auto*)
also have ... = *?rhs*
using *i F-gr-0*[*OF assms*] **by** (*simp add:powr-add*[*symmetric*] *powr-realpow*[*symmetric*])
finally show *?thesis*
by *blast*
next
case *False*
have *n* = *0* ⟹ *False*
using *as-range assms* **by** *auto*
hence *n* > *0*
by *auto*
moreover have *k* = *1*
using *assms k-ge-1 False* **by** *linarith*
moreover have *length as* = *real-of-rat* (*F* (*Suc 0*) *as*)
by (*simp add:F-def sum-count-set of-nat-sum*[*symmetric*] *del:of-nat-sum*)
ultimately show *?thesis*
by (*simp add:power2-eq-square*)
qed

definition *result*
where *result a* = *of-nat* (*length as*) ∗ *of-nat* (*Suc* (*snd a*) ^ *k* − *snd a* ^ *k*)

lemma *result-exp-1*:
assumes *as* ≠ []
shows *expectation result* = *real-of-rat* (*F k as*)

proof −
  have *expectation result* = ($\sum$ *a*∈M_1. *result a* ∗ *pmf* (*pmf-of-set* M_1) *a*)
    unfolding Ω_1-*def* **using** *non-empty-space assms fin-space*
    by (*subst integral-measure-pmf-real*) *auto*
  also have ... = ($\sum$ *a*∈M_1. *result a* / *real* (*length as*))
   using *non-empty-space assms fin-space card-space* **by** *simp*
  also have ... = ($\sum$ *a*∈M_1. *real* (*Suc* (*snd a*) ^ *k* − *snd a* ^ *k*))
    using *assms* **by** (*simp add*:*result-def*)
  also have ... = ($\sum$ *u*∈*set as*. $\sum$ *v* = *0*..<*count-list as u*. *real* (*Suc v* ^ *k*) − *real* (*v* ^ *k*))
    using *k-ge-1* **by** (*subst split-space*, *simp add*:*of-nat-diff*)
  also have ... = ($\sum$ *u*∈*set as*. *real* (*count-list as u*)^*k*)
    using *k-ge-1* **by** (*subst sum-Suc-diff'*) (*auto simp add*:*zero-power*)
  also have ... = *of-rat* (*F k as*)
    by (*simp add*:*F-def of-rat-sum of-rat-power*)
  finally show *?thesis* **by** *simp*
qed

lemma *result-var-1*:
  assumes *as* ≠ []
  shows *variance result* ≤ (*of-rat* (*F k as*))^2 ∗ *k* ∗ *n powr* (*1* − *1* / *real k*)
proof −
  have *k-gt-0*: *k* > *0* **using** *k-ge-1* **by** *linarith*

  have *c*:*real* (*Suc v* ^ *k*) − *real* (*v* ^ *k*) ≤ *k* ∗ *real* (*count-list as a*) ^ (*k* − *Suc 0*)
    if *c-1*: *v* < *count-list as a* **for** *a v*
  proof −
    have *real* (*Suc v* ^ *k*) − *real* (*v* ^ *k*) ≤ (*real* (*v*+*1*) − *real v*) ∗ *k* ∗ (*1* + *real v*) ^ (*k* − *Suc 0*)
      using *k-gt-0 power-diff-est*[**where** *a*=*Suc v* **and** *b*=*v*] **by** *simp*
    moreover have (*real* (*v*+*1*) − *real v*) = *1* **by** *auto*
    ultimately have *real* (*Suc v* ^ *k*) − *real* (*v* ^ *k*) ≤ *k* ∗ (*1* + *real v*) ^ (*k* − *Suc 0*)
      by *auto*
    also have ... ≤ *k* ∗ *real* (*count-list as a*) ^ (*k*− *Suc 0*)
      using *c-1* **by** (*intro mult-left-mono power-mono*, *auto*)
    finally show *?thesis* **by** *blast*
  qed

  have *length as* ∗ ($\sum$ *a*∈ M_1. (*real* (*Suc* (*snd a*) ^ *k* − (*snd a*) ^ *k*))^2) =
    length as ∗ ($\sum$ *a*∈ *set as*. ($\sum$ *v* ∈ {*0*..<*count-list as a*}.
    real (*Suc v* ^ *k* − *v* ^ *k*) ∗ *real* (*Suc v* ^ *k* − *v*^*k*)))
    by (*subst split-space*, *simp add*:*power2-eq-square*)
  also have ... ≤ *length as* ∗ ($\sum$ *a*∈ *set as*. ($\sum$ *v* ∈ {*0*..<*count-list as a*}.
    k ∗ *real* (*count-list as a*) ^ (*k*−*1*) ∗ *real* (*Suc v* ^ *k* − *v* ^ *k*)))
   using *c* **by** (*intro mult-left-mono sum-mono mult-right-mono*) (*auto simp*:*power-mono of-nat-diff*)
  also have ... = *length as* ∗ *k* ∗ ($\sum$ *a*∈ *set as*. *real* (*count-list as a*) ^ (*k*−*1*) ∗
    ($\sum$ *v* ∈ {*0*..<*count-list as a*}. *real* (*Suc v* ^ *k*) − *real* (*v* ^ *k*)))

by (*simp add:sum-distrib-left ac-simps of-nat-diff power-mono*)
also have ... = *length as* ∗ *k* ∗ ($\sum$ *a*∈ *set as*. *real* (*count-list as a* ^ (*2*∗*k*−*1*)))
using *assms k-ge-1*
by (*subst sum-Suc-diff′*, *auto simp*: *zero-power*[*OF k-gt-0*] *mult-2 power-add*[*symmetric*])
also have ... = *k* ∗ (*length as* ∗ *of-rat* (*F* (*2*∗*k*−*1*) *as*))
by (*simp add:sum-distrib-left*[*symmetric*] *F-def of-rat-sum of-rat-power*)
also have ... ≤ *k* ∗ (*of-rat* (*F k as*)^*2* ∗ *n powr* (*1* − *1* / *real k*))
using *fk-estimate*[*OF assms*] **by** (*intro mult-left-mono*) (*auto simp*: *mult.commute*)
finally have *b*: *real* (*length as*) ∗ ($\sum$ *a*∈ M_1. (*real* (*Suc* (*snd a*) ^ *k* − (*snd a*) ^ *k*))^2) ≤
k ∗ ((*of-rat* (*F k as*))^2 ∗ *n powr* (*1* − *1* / *real k*))
by *blast*

have *expectation* (λω. (*result* ω :: *real*)^*2*) − (*expectation result*)^*2* ≤ *expectation* (λω. *result* ω^*2*)
by *simp*
also have ... = ($\sum$ *a*∈M_1. (*length as* ∗ *real* (*Suc* (*snd a*) ^ *k* − *snd a* ^ *k*))^2 ∗ *pmf* (*pmf-of-set* M_1) *a*)
using *fin-space non-empty-space assms* **unfolding** Ω_1*-def result-def*
by (*subst integral-measure-pmf-real*[**where** *A*=M_1], *auto*)
also have ... = ($\sum$ *a*∈M_1. *length as* ∗ (*real* (*Suc* (*snd a*) ^ *k* − *snd a* ^ *k*))^2)
using *assms non-empty-space fin-space* **by** (*subst pmf-of-set*)
(*simp-all add:card-space power-mult-distrib power2-eq-square ac-simps*)
also have ... ≤ *k* ∗ ((*of-rat* (*F k as*))^2 ∗ *n powr* (*1* − *1* / *real k*))
using *b* **by** (*simp add:sum-distrib-left*[*symmetric*])
also have ... = *of-rat* (*F k as*)^*2* ∗ *k* ∗ *n powr* (*1* − *1* / *real k*)
by (*simp add:ac-simps*)
finally have *expectation* (λω. *result* ω^*2*) − (*expectation result*)^*2* ≤
of-rat (*F k as*)^*2* ∗ *k* ∗ *n powr* (*1* − *1* / *real k*)
by *blast*

thus *?thesis*
using *integrable-1*[*OF assms*] **by** (*simp add:variance-eq*)
qed

theorem *fk-alg-sketch*:
assumes *as* ≠ []
shows *fold* (λ*a state*. *state* ⤜ *fk-update a*) *as* (*fk-init k* δ ε *n*) =
map-pmf (λ*x*. (s_1,s_2,*k*,*length as*, *x*)) M_2 (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$)
(λ-. *fold* (λ*x s*. *s* ⤜ *fk-update-2 x*) *as* (*return-pmf* (*0*, *0*, *0*))) ⤜
(λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, λ*i*∈$\{0..<s_1\} \times \{0..<s_2\}$. *snd* (*x i*)))
by (*subst fk-update-distr*, *simp*)
also have ... = *prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$) (λ-. *pmf-of-set* {..<*length as*} ⤜
(λ*k*. *return-pmf* (*length as*, *sketch as k*))) ⤜
(λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, λ*i*∈$\{0..<s_1\} \times \{0..<s_2\}$. *snd* (*x i*)))
by (*subst fk-update-2-distr*[*OF assms*], *simp*)

also have ... = *prod-pmf* ({*0*..<s_1} × {*0*..<s_2}) (λ-. *pmf-of-set* {..<*length as*}
≫=
  (λ*k*. *return-pmf* (*sketch as k*)) ≫= (λ*s*. *return-pmf* (*length as*, *s*))) ≫=
  (λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, λ*i*∈{*0*..<s_1} × {*0*..<s_2}. *snd* (*x i*)))
  by (*subst bind-assoc-pmf*, *subst bind-return-pmf*, *simp*)
also have ... = *prod-pmf* ({*0*..<s_1} × {*0*..<s_2}) (λ-. *pmf-of-set* {..<*length as*}
≫=
  (λ*k*. *return-pmf* (*sketch as k*))) ≫=
  (λ*x*. *return-pmf* (λ*i* ∈ {*0*..<s_1} × {*0*..<s_2}. (*length as*, *x i*))) ≫=
  (λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, λ*i*∈{*0*..<s_1} × {*0*..<s_2}. *snd* (*x i*)))
 by (*subst Pi-pmf-bind-return*[**where** *d'*=*undefined*], *simp*, *simp add*:*restrict-def*)
also have ... = *prod-pmf* ({*0*..<s_1} × {*0*..<s_2}) (λ-. *pmf-of-set* {..<*length as*}
≫=
  (λ*k*. *return-pmf* (*sketch as k*))) ≫=
  (λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, *restrict x* ({*0*..<s_1} × {*0*..<s_2})))
  by (*subst bind-assoc-pmf*, *simp add*:*bind-return-pmf cong*:*restrict-cong*)
also have ... = M_2 ≫=
  (λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, *restrict x* ({*0*..<s_1} × {*0*..<s_2})))
  by (*subst sketch-distr*[*OF assms*], *simp add*:M_2*-def*)
also have ... = M_2 ≫= (λ*x*. *return-pmf* (s_1, s_2, *k*, *length as*, *x*))
  by (*rule bind-pmf-cong*, *auto simp add*:*PiE-dflt-def* M_2*-def set-Pi-pmf*)
also have ... = *?rhs*
  by (*simp add*:*map-pmf-def*)
finally show *?thesis* **by** *simp*
qed

definition *mean-rv*
  where *mean-rv* ω i_2 = (∑ i_1 = *0*..<s_1. *result* (ω (i_1, i_2))) / *of-nat* s_1

definition *median-rv*
  where *median-rv* ω = *median* s_2 (λi_2. *mean-rv* ω i_2)

lemma *fk-alg-correct'*:
 defines *M* ≡ *fold* (λ*a state*. *state* ≫= *fk-update a*) *as* (*fk-init k* δ ε *n*) ≫= *fk-result*
 shows 𝒫(ω *in measure-pmf M*. |ω − *F k as*| ≤ δ ∗ *F k as*) ≥ *1* − *of-rat* ε
proof (*cases as* = [])
 case *True*
 have *a*: *nat* ⌈− (*18* ∗ *ln* (*real-of-rat* ε))⌉ > *0* **using** ε*-range* **by** *simp*
 show *?thesis* **using** *True* ε*-range*
  by (*simp add*:*F-def M-def bind-return-pmf median-const*[*OF a*] *Let-def*)
next
 case *False*

 have *set as* ≠ {} **using** *assms False* **by** *blast*
 hence *n-nonzero*: *n* > *0* **using** *as-range* **by** *fastforce*

 have *fk-nonzero*: *F k as* > *0*
  using *F-gr-0*[*OF False*] **by** *simp*

have *s1-nonzero*: $s_1 > 0$
using *δ-range k-ge-1 n-nonzero* **by** (*simp add*:s_1-*def*)
have *s2-nonzero*: $s_2 > 0$
using *ε-range* **by** (*simp add*:s_2-*def*)

have *real-of-rat-mean-rv*: $\bigwedge x\ i.$ *mean-rv x* = (λ*i. real-of-rat* (*mean-rv x i*))
by (*rule ext*, *simp add*:*of-rat-divide of-rat-sum of-rat-mult result-def mean-rv-def*)
have *real-of-rat-median-rv*: $\bigwedge x.$ *median-rv x* = *real-of-rat* (*median-rv x*)
unfolding *median-rv-def* **using** *s2-nonzero*
by (*subst real-of-rat-mean-rv*, *simp add*: *median-rat median-restrict*)

have *space-*Ω_2: *space* Ω_2 = *UNIV* **by** (*simp add*:Ω_2-*def*)

have *fk-result-eta*: *fk-result* = (λ(*x*,*y*,*z*,*u*,*v*). *fk-result* (*x*,*y*,*z*,*u*,*v*))
by *auto*

have *a*:*fold* (λ*x state. state* ⤜ *fk-update x*) *as* (*fk-init k δ ε n*) =
map-pmf (λ*x*. (s_1,s_2,*k*,*length as*, *x*)) M_2
by (*subst fk-alg-sketch*[*OF False*]) (*simp add*:s_1-*def*[*symmetric*] s_2-*def*[*symmetric*])

have *M* = *map-pmf* (λ*x*. (s_1, s_2, *k*, *length as*, *x*)) M_2 ⤜ *fk-result*
by (*subst M-def*, *subst a*, *simp*)
also have ... = M_2 ⤜ *return-pmf* ∘ *median-rv*
by (*subst fk-result-eta*)
(*auto simp add*:*map-pmf-def bind-assoc-pmf bind-return-pmf median-rv-def mean-rv-def comp-def*
M_1-*def result-def median-restrict*)
finally have *b*: *M* = M_2 ⤜ *return-pmf* ∘ *median-rv*
by *simp*

have *result-exp*:
$i_1 < s_1 \Longrightarrow i_2 < s_2 \Longrightarrow$ Ω_2.*expectation* (λ*x*. *result* (*x* (i_1, i_2))) = *real-of-rat* (*F k as*)
for i_1 i_2
unfolding Ω_2-*def* M_2-*def*
using *integrable-1*[*OF False*] *result-exp-1*[*OF False*]
by (*subst expectation-Pi-pmf-slice*, *auto simp*:Ω_1-*def*)

have *result-var*: Ω_2.*variance* (λω. *result* (ω (i_1, i_2))) ≤ *of-rat* (δ ∗ *F k as*)^*2* ∗ *real* s_1 / *3*
if *result-var-assms*: $i_1 < s_1$ $i_2 < s_2$ **for** i_1 i_2
proof −
have *3* ∗ *real k* ∗ *n powr* (*1* − *1* / *real k*) =
(*of-rat* δ)^2 ∗ (*3* ∗ *real k* ∗ *n powr* (*1* − *1* / *real k*) / (*of-rat* δ)^2)
using *δ-range* **by** *simp*
also have ... ≤ (*real-of-rat* δ)^2 ∗ (*real* s_1)
unfolding s_1-*def*

by (*intro mult-mono of-nat-ceiling*, *simp-all*)
finally have *f2-var-2*: *3* ∗ *real k* ∗ *n powr* (*1* − *1* / *real k*) ≤ (*of-rat* δ)^2 ∗ (*real* s_1)
by *blast*

have Ω_2.*variance* ($\lambda\omega$. *result* (ω (i_1, i_2)) :: *real*) = *variance result*
using *result-var-assms integrable-1*[*OF False*]
unfolding Ω_2-*def* M_2-*def* Ω_1-*def*
by (*subst variance-prod-pmf-slice*, *auto*)
also have ... ≤ *of-rat* (*F k as*)^*2* ∗ *real k* ∗ *n powr* (*1* − *1* / *real k*)
using *assms False result-var-1* Ω_1-*def* **by** *simp*
also have ... =
of-rat (*F k as*)^*2* ∗ (*real k* ∗ *n powr* (*1* − *1* / *real k*))
by (*simp add*:*ac-simps*)
also have ... ≤ *of-rat* (*F k as*)^*2* ∗ (*of-rat* δ^*2* ∗ (*real* s_1 / *3*))
using *f2-var-2* **by** (*intro mult-left-mono*, *auto*)
also have ... = *of-rat* (*F k as* ∗ δ)^*2* ∗ (*real* s_1 / *3*)
by (*simp add*: *of-rat-mult power-mult-distrib*)
also have ... = *of-rat* (δ ∗ *F k as*)^*2* ∗ *real* s_1 / *3*
by (*simp add*:*ac-simps*)
finally show *?thesis*
by *simp*
qed

have *mean-rv-exp*: Ω_2.*expectation* ($\lambda\omega$. *mean-rv* ω *i*) = *real-of-rat* (*F k as*)
if *mean-rv-exp-assms*: *i* < s_2 **for** *i*
proof −
have Ω_2.*expectation* ($\lambda\omega$. *mean-rv* ω *i*) = Ω_2.*expectation* ($\lambda\omega$. $\sum$ *n* = *0*..<s_1. *result* (ω (*n*, *i*)) / *real* s_1)
by (*simp add*:*mean-rv-def sum-divide-distrib*)
also have ... = ($\sum$ *n* = *0*..<s_1. Ω_2.*expectation* ($\lambda\omega$. *result* (ω (*n*, *i*))) / *real* s_1)
using *integrable-2*[*OF False*]
by (*subst Bochner-Integration.integral-sum*, *auto*)
also have ... = *of-rat* (*F k as*)
using *s1-nonzero mean-rv-exp-assms*
by (*simp add*:*result-exp*)
finally show *?thesis* **by** *simp*
qed

have *mean-rv-var*: Ω_2.*variance* ($\lambda\omega$. *mean-rv* ω *i*) ≤ *real-of-rat* (δ ∗ *F k as*)^*2*/*3*

if *mean-rv-var-assms*: *i* < s_2 **for** *i*
proof −
have *a*:Ω_2.*indep-vars* (λ-. *borel*) (λ*n x*. *result* (*x* (*n*, *i*)) / *real* s_1) {*0*..<s_1}
unfolding Ω_2-*def* M_2-*def* **using** *mean-rv-var-assms*
by (*intro indep-vars-restrict-intro*′[**where** *f*=*fst*], *simp*, *simp add*:*restrict-dfl-def*, *simp*, *simp*)
have Ω_2.*variance* ($\lambda\omega$. *mean-rv* ω *i*) = Ω_2.*variance* ($\lambda\omega$. $\sum$ *j* = *0*..<s_1. *result* (ω (*j*, *i*)) / *real* s_1)

by (*simp add*:*mean-rv-def sum-divide-distrib*)
also have ... = ($\sum j = 0..<s_1$. Ω_2.*variance* ($\lambda\omega$. *result* (ω (*j*, *i*)) / *real* s_1))
using *a integrable-2*[*OF False*]
by (*subst* Ω_2.*var-sum-all-indep*, *auto simp add*:Ω_2-*def*)
also have ... = ($\sum j = 0..<s_1$. Ω_2.*variance* ($\lambda\omega$. *result* (ω (*j*, *i*))) / *real* s_1 ^*2*)
using *integrable-2*[*OF False*]
by (*subst* Ω_2.*variance-divide*, *auto*)
also have ... $\leq$ ($\sum j = 0..<s_1$. ((*real-of-rat* ($\delta * F\ k\ as$))^2 $*$ *real* s_1 / *3*) / (*real* s_1 ^*2*))
using *result-var*[*OF - mean-rv-var-assms*]
by (*intro sum-mono divide-right-mono*, *auto*)
also have ... = *real-of-rat* ($\delta * F\ k\ as$)^*2*/*3*
using *s1-nonzero*
by (*simp add*:*algebra-simps power2-eq-square*)
finally show *?thesis* **by** *simp*
qed

have Ω_2.*prob* {*y*. *of-rat* ($\delta * F\ k\ as$) < |*mean-rv y i* − *real-of-rat* (*F k as*)|} $\leq$ *1*/*3*
(**is** *?lhs* $\leq$ -) **if** *c-assms*: *i* < s_2 **for** *i*
proof −
define *a* **where** *a* = *real-of-rat* ($\delta * F\ k\ as$)
have *c*: *0* < *a* **unfolding** *a-def*
using *assms* δ-*range fk-nonzero*
by (*metis zero-less-of-rat-iff mult-pos-pos*)
have *?lhs* $\leq$ Ω_2.*prob* {*y* $\in$ *space* Ω_2. *a* $\leq$ |*mean-rv y i* − Ω_2.*expectation* ($\lambda\omega$. *mean-rv* ω *i*)|}
by (*intro* Ω_2.*pmf-mono*[*OF* Ω_2-*def*], *simp add*:*a-def mean-rv-exp*[*OF c-assms*] *space*-Ω_2)
also have ... $\leq$ Ω_2.*variance* ($\lambda\omega$. *mean-rv* ω *i*)/*a*^*2*
by (*intro* Ω_2.*Chebyshev-inequality integrable-2 c False*) (*simp add*:Ω_2-*def*)
also have ... $\leq$ *1*/*3* **using** *c*
using *mean-rv-var*[*OF c-assms*]
by (*simp add*:*algebra-simps*, *simp add*:*a-def*)
finally show *?thesis*
by *blast*
qed

moreover have Ω_2.*indep-vars* (λ-. *borel*) ($\lambda i\ \omega$. *mean-rv* ω *i*) {*0*..<s_2}
using *s1-nonzero* **unfolding** Ω_2-*def* M_2-*def*
by (*intro indep-vars-restrict-intro'*[**where** *f*=*snd*] *finite-cartesian-product*)
(*simp-all add*:*mean-rv-def restrict-dfl-def space*-Ω_2)
moreover have − (*18* $*$ *ln* (*real-of-rat* ε)) $\leq$ *real* s_2
by (*simp add*:s_2-*def*, *linarith*)
ultimately have *1* − *of-rat* ε $\leq$
Ω_2.*prob* {*y* $\in$ *space* Ω_2. |*median* s_2 (*mean-rv y*) − *real-of-rat* (*F k as*)| $\leq$ *of-rat* ($\delta * F\ k\ as$)}
using ε-*range*
by (*intro* Ω_2.*median-bound-2*, *simp-all add*:*space*-Ω_2)

also have ... = Ω_2*.prob* {*y.* |*median-rv y* − *real-of-rat* (*F k as*)| ≤ *real-of-rat* (*δ* ∗ *F k as*)}
  by (*simp add*:*median-rv-def space-*Ω_2)
also have ... = Ω_2*.prob* {*y.* |*median-rv y* − *F k as*| ≤ *δ* ∗ *F k as*}
  by (*simp add*:*real-of-rat-median-rv of-rat-less-eq flip*: *of-rat-diff*)
also have ... = $\mathcal{P}$(*ω in measure-pmf M.* |*ω* − *F k as*| ≤ *δ* ∗ *F k as*)
  by (*simp add*: *b comp-def map-pmf-def*[*symmetric*] Ω_2*-def*)
finally show *?thesis* **by** *simp*
qed

lemma *fk-exact-space-usage′*:
defines *M* ≡ *fold* (*λa state. state* ⤜ *fk-update a*) *as* (*fk-init k δ ε n*)
shows *AE ω in M. bit-count* (*encode-fk-state ω*) ≤ *fk-space-usage* (*k*, *n*, *length as*, *ε*, *δ*)
  (**is** *AE ω in M.* (- ≤ *?rhs*))
proof −
 define *H* **where** *H* = (*if as* = [] *then return-pmf* (*λi*∈ {*0..<*s_1}×{*0..<*s_2}. (*0*,*0*)) *else* M_2)

have *a*:*M* = *map-pmf* (*λx.*(s_1,s_2,*k*,*length as*, *x*)) *H*
proof (*cases as* ≠ [])
  case *True*
  then show *?thesis*
    unfolding *M-def fk-alg-sketch*[*OF True*] *H-def*
    by (*simp add*:M_2*-def*)
next
  case *False*
  then show *?thesis*
  by (*simp add*:*H-def M-def* s_1*-def*[*symmetric*] *Let-def* s_2*-def*[*symmetric*] *map-pmf-def bind-return-pmf*)
qed

have *bit-count* (*encode-fk-state* (s_1, s_2, *k*, *length as*, *y*)) ≤ *?rhs*
  if *b*:*y* ∈ *set-pmf H* **for** *y*
proof −
  have *b0*: *as* ≠ [] ⟹ *y* ∈ {*0..<*s_1} × {*0..<*s_2} $\rightarrow_E$ M_1
    using *b non-empty-space fin-space* **by** (*simp add*:*H-def* M_2*-def set-prod-pmf*)

  have *bit-count* ((N_e $\times_e$ N_e) (*y x*)) ≤
   ereal (*2* ∗ *log 2* (*real n* + *1*) + *1*) + *ereal* (*2* ∗ *log 2* (*real* (*length as*) + *1*) + *1*)
    (**is** - ≤ *?rhs1*)
    if *b1-assms*: *x* ∈ {*0..<*s_1}×{*0..<*s_2} **for** *x*
  proof −
    have *fst* (*y x*) ≤ *n*
    proof (*cases as* = [])
      case *True*
      then show *?thesis* **using** *b b1-assms* **by** (*simp add*:*H-def*)
    next

case *False*
hence *1* ≤ *count-list as* (*fst* (*y x*))
using *b0 b1-assms* **by** (*simp add*:*PiE-iff case-prod-beta* M_1*-def*, *fastforce*)
hence *fst* (*y x*) ∈ *set as*
using *count-list-gr-1* **by** *metis*
then show *?thesis*
by (*meson lessThan-iff less-imp-le-nat subsetD as-range*)
qed
moreover have *snd* (*y x*) ≤ *length as*
proof (*cases as* = [])
case *True*
then show *?thesis* **using** *b b1-assms* **by** (*simp add*:*H-def*)
next
case *False*
hence (*y x*) ∈ M_1
using *b0 b1-assms* **by** *auto*
hence *snd* (*y x*) ≤ *count-list as* (*fst* (*y x*))
by (*simp add*:M_1*-def case-prod-beta*)
then show *?thesis* **using** *count-le-length* **by** (*metis order-trans*)
qed
ultimately have *bit-count* (N_e (*fst* (*y x*))) + *bit-count* (N_e (*snd* (*y x*))) ≤ *?rhs1*
using *exp-golomb-bit-count-est* **by** (*intro add-mono*, *auto*)
thus *?thesis*
by (*subst dependent-bit-count-2*, *simp*)
qed

moreover have *y* ∈ *extensional* ({*0*..<s_1} × {*0*..<s_2})
using *b0 b PiE-iff* **by** (*cases as* = [], *auto simp*:*H-def PiE-iff*)

ultimately have *bit-count* ((*List.product* [*0*..<s_1] [*0*..<s_2] $\rightarrow_e$ N_e $\times_e$ N_e) *y*) ≤
ereal (*real* s_1 ∗ *real* s_2) ∗ (*ereal* (*2* ∗ *log 2* (*real n* + *1*) + *1*) +
ereal (*2* ∗ *log 2* (*real* (*length as*) + *1*) + *1*))
by (*intro fun-bit-count-est*[**where** *xs*=(*List.product* [*0*..<s_1] [*0*..<s_2]), *simplified*], *auto*)
hence *bit-count* (*encode-fk-state* (s_1, s_2, *k*, *length as*, *y*)) ≤
ereal (*2* ∗ *log 2* (*real* s_1 + *1*) + *1*) +
(*ereal* (*2* ∗ *log 2* (*real* s_2 + *1*) + *1*) +
(*ereal* (*2* ∗ *log 2* (*real k* + *1*) + *1*) +
(*ereal* (*2* ∗ *log 2* (*real* (*length as*) + *1*) + *1*) +
(*ereal* (*real* s_1 ∗ *real* s_2) ∗ (*ereal* (*2* ∗ *log 2* (*real n*+*1*) + *1*) +
ereal (*2* ∗ *log 2* (*real* (*length as*)+*1*) + *1*))))))
unfolding *encode-fk-state-def dependent-bit-count*
by (*intro add-mono exp-golomb-bit-count*, *auto*)
also have ... ≤ *?rhs*
by (*simp add*: s_1*-def*[*symmetric*] s_2*-def*[*symmetric*] *Let-def*) (*simp add*:*ac-simps*)
finally show *bit-count* (*encode-fk-state* (s_1, s_2, *k*, *length as*, *y*)) ≤ *?rhs*
by *blast*

qed
thus *?thesis*
by (*simp add*: *a AE-measure-pmf-iff del*:*fk-space-usage.simps*)
qed

end

Main results of this section:

theorem *fk-alg-correct*:
assumes $k \geq 1$
assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq \{..<n\}$
defines $M \equiv$ *fold* (λa *state. state* $\gg=$ *fk-update a*) *as* (*fk-init k* δ ε *n*) $\gg=$ *fk-result*
shows $\mathcal{P}$(ω *in measure-pmf M*. $|\omega - F\ k\ as| \leq \delta * F\ k\ as$) $\geq 1 -$ *of-rat* ε
unfolding *M-def* **using** *fk-alg-correct'*[*OF assms*(*1*−*4*)] **by** *blast*

theorem *fk-exact-space-usage*:
assumes $k \geq 1$
assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq \{..<n\}$
defines $M \equiv$ *fold* (λa *state. state* $\gg=$ *fk-update a*) *as* (*fk-init k* δ ε *n*)
shows *AE* ω *in M. bit-count* (*encode-fk-state* ω) $\leq$ *fk-space-usage* (*k*, *n*, *length as*, ε, δ)
unfolding *M-def* **using** *fk-exact-space-usage'*[*OF assms*(*1*−*4*)] **by** *blast*

theorem *fk-asymptotic-space-complexity*:
fk-space-usage $\in$
O[*at-top* $\times_F$ *at-top* $\times_F$ *at-top* $\times_F$ *at-right* (*0*::*rat*) $\times_F$ *at-right* (*0*::*rat*)](λ (*k*, *n*, *m*, ε, δ).
real k $*$ *real n powr* (*1*−*1*/ *real k*) / (*of-rat* δ)^2 $*$ (*ln* (*1* / *of-rat* ε)) $*$ (*ln* (*real n*) + *ln* (*real m*)))
(**is** - $\in$ *O*[*?F*](*?rhs*))
proof −
define *k-of* :: *nat* $\times$ *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *nat* **where** *k-of* = (λ(*k*, *n*, *m*, ε, δ). *k*)
define *n-of* :: *nat* $\times$ *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *nat* **where** *n-of* = (λ(*k*, *n*, *m*, ε, δ). *n*)
define *m-of* :: *nat* $\times$ *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *nat* **where** *m-of* = (λ(*k*, *n*, *m*, ε, δ). *m*)
define ε*-of* :: *nat* $\times$ *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *rat* **where** ε*-of* = (λ(*k*, *n*, *m*, ε, δ). ε)
define δ*-of* :: *nat* $\times$ *nat* $\times$ *nat* $\times$ *rat* $\times$ *rat* $\Rightarrow$ *rat* **where** δ*-of* = (λ(*k*, *n*, *m*, ε, δ). δ)

define *g1* **where**
g1 = (λ*x*. *real* (*k-of x*)$*$(*real* (*n-of x*)) *powr* (*1*−*1*/ *real* (*k-of x*)) $*$ (*1* / *of-rat* (δ*-of x*)^*2*))

define *g* **where**
g = (λ*x*. *g1 x* ∗ (*ln* (*1* / *of-rat* (ε-*of x*))) ∗ (*ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*))))

define *s1-of* **where** *s1-of* = (λ*x*.
nat ⌈*3* ∗ *real* (*k-of x*) ∗ *real* (*n-of x*) *powr* (*1* − *1* / *real* (*k-of x*)) / (*real-of-rat* (δ-*of x*))^2⌉)
define *s2-of* **where** *s2-of* = (λ*x*. *nat* ⌈− (*18* ∗ *ln* (*real-of-rat* (ε-*of x*)))⌉)

have *evt*: (⋀*x*.
0 < *real-of-rat* (δ-*of x*) ∧ *0* < *real-of-rat* (ε-*of x*) ∧
1/*real-of-rat* (δ-*of x*) ≥ δ ∧ *1*/*real-of-rat* (ε-*of x*) ≥ ε ∧
real (*n-of x*) ≥ *n* ∧ *real* (*k-of x*) ≥ *k* ∧ *real* (*m-of x*) ≥ *m*⟹ *P x*)
⟹ *eventually P ?F* (**is** (⋀*x*. *?prem x* ⟹ -) ⟹ -)
for δ ε *n k m P*
apply (*rule eventually-mono*[**where** *P*=*?prem* **and** *Q*=*P*])
apply (*simp add*:ε-*of-def case-prod-beta′* δ-*of-def n-of-def k-of-def m-of-def*)
apply (*intro eventually-conj eventually-prod1′ eventually-prod2′*
sequentially-inf eventually-at-right-less inv-at-right-0-inf)
by (*auto simp add*:*prod-filter-eq-bot*)

have *1*:
(λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*n-of x*))
(λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*m-of x*))
(λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*k-of x*))
by (*intro landau-o.big-mono eventually-mono*[*OF evt*], *auto*)+

have (λ*x*. *ln* (*real* (*m-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*m-of x*)))
by (*intro landau-ln-2*[**where** *a*=*2*] *evt*[**where** *m*=*2*] *sum-in-bigo 1*, *auto*)
hence *2*: (λ*x*. *log 2* (*real* (*m-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro landau-sum-2 eventually-mono*[*OF evt*[**where** *n*=*1* **and** *m*=*1*]])
(*auto simp add*:*log-def*)

have *3*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε-*of x*)))
using *order-less-le-trans*[*OF exp-gt-zero*] *ln-ge-iff*
by (*intro landau-o.big-mono evt*[**where** ε=*exp 1*])
(*simp add*: *abs-ge-iff*, *blast*)

have *4*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *1* / (*real-of-rat* (δ-*of x*))^2)
using *one-le-power*
by (*intro landau-o.big-mono evt*[**where** δ=*1*])
(*simp add*:*power-one-over*[*symmetric*], *blast*)

have (λ*x*. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)))
using *order-less-le-trans*[*OF exp-gt-zero*] *ln-ge-iff*
by (*intro landau-o.big-mono evt*[**where** *n*=*exp 1*])

(*simp add*: *abs-ge-iff*, *blast*)

hence *5*: (λ*x*. *1*) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
by (*intro landau-sum-1 evt*[**where** *n*=*1* **and** *m*=*1*], *auto*)

have (λ*x*. −*ln*(*of-rat* (ε*-of x*))) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
by (*intro landau-o.big-mono evt*) (*auto simp add*:*ln-div*)
hence *6*: (λ*x*. *real* (*s2-of x*)) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
unfolding *s2-of-def*
by (*intro landau-nat-ceil 3*, *simp*)

have *7*: (λ-. *1*) ∈ *O*[*?F*](λ*x*. *real* (*n-of x*) *powr* (*1* − *1* / *real* (*k-of x*)))
by (*intro landau-o.big-mono evt*[**where** *n*=*1* **and** *k*=*1*])
(*auto simp add*: *ge-one-powr-ge-zero*)

have *8*: (λ-. *1*) ∈ *O*[*?F*](*g1*)
unfolding *g1-def* **by** (*intro landau-o.big-mult-1 1 7 4*)

have (λ*x*. *3* ∗ (*real* (*k-of x*) ∗ (*n-of x*) *powr* (*1* − *1* / *real* (*k-of x*)) / (*of-rat* (δ*-of x*))^2))
∈ *O*[*?F*](*g1*)
by (*subst landau-o.big.cmult-in-iff*, *simp*, *simp add*:*g1-def*)
hence *9*: (λ*x*. *real* (*s1-of x*)) ∈ *O*[*?F*](*g1*)
unfolding *s1-of-def* **by** (*intro landau-nat-ceil 8*, *auto simp*:*ac-simps*)

have *10*: (λ-. *1*) ∈ *O*[*?F*](*g*)
unfolding *g-def* **by** (*intro landau-o.big-mult-1 8 3 5*)

have (λ*x*. *real* (*s1-of x*)) ∈ *O*[*?F*](*g*)
unfolding *g-def* **by** (*intro landau-o.big-mult-1 5 3 9*)
hence (λ*x*. *ln* (*real* (*s1-of x*) + *1*)) ∈ *O*[*?F*](*g*)
using *10* **by** (*intro landau-ln-3 sum-in-bigo*, *auto*)
hence *11*: (λ*x*. *log 2* (*real* (*s1-of x*) + *1*)) ∈ *O*[*?F*](*g*)
by (*simp add*:*log-def*)

have *12*: (λ*x*. *ln* (*real* (*s2-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*1* / *real-of-rat* (ε*-of x*)))
using *evt*[**where** ε=*2*] *6 3*
by (*intro landau-ln-3 sum-in-bigo*, *auto*)

have *13*: (λ*x*. *log 2* (*real* (*s2-of x*) + *1*)) ∈ *O*[*?F*](*g*)
unfolding *g-def*
by (*rule landau-o.big-mult-1*, *rule landau-o.big-mult-1'*, *auto simp add*: *8 5 12 log-def*)

have (λ*x*. *real* (*k-of x*)) ∈ *O*[*?F*](*g1*)
unfolding *g1-def* **using** *7 4*
by (*intro landau-o.big-mult-1*, *simp-all*)
hence (λ*x*. *log 2* (*real* (*k-of x*) + *1*)) ∈ *O*[*?F*](*g1*)
by (*simp add*:*log-def*) (*intro landau-ln-3 sum-in-bigo 8*, *auto*)

hence *14*: (λ*x*. *log 2* (*real* (*k-of x*) + *1*)) ∈ *O*[*?F*](*g*)
  unfolding *g-def* **by** (*intro landau-o.big-mult-1 3 5*)

have *15*: (λ*x*. *log 2* (*real* (*m-of x*) + *1*)) ∈ *O*[*?F*](*g*)
  unfolding *g-def* **using** *2 8 3*
  by (*intro landau-o.big-mult-1′*, *simp-all*)

have (λ*x*. *ln* (*real* (*n-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)))
 by (*intro landau-ln-2*[**where** *a*=*2*] *eventually-mono*[*OF evt*[**where** *n*=*2*]] *sum-in-bigo 1*, *auto*)
hence (λ*x*. *log 2* (*real* (*n-of x*) + *1*)) ∈ *O*[*?F*](λ*x*. *ln* (*real* (*n-of x*)) + *ln* (*real* (*m-of x*)))
  by (*intro landau-sum-1 evt*[**where** *n*=*1* **and** *m*=*1*])
   (*auto simp add*:*log-def*)
hence *16*: (λ*x*. *real* (*s1-of x*) ∗ *real* (*s2-of x*) ∗
 (*2* + *2* ∗ *log 2* (*real* (*n-of x*) + *1*) + *2* ∗ *log 2* (*real* (*m-of x*) + *1*))) ∈ *O*[*?F*](*g*)

  unfolding *g-def* **using** *9 6 5 2*
  by (*intro landau-o.mult sum-in-bigo*, *auto*)

have *fk-space-usage* = (λ*x*. *fk-space-usage* (*k-of x*, *n-of x*, *m-of x*, ε*-of x*, δ*-of x*))
  by (*simp add*:*case-prod-beta′ k-of-def n-of-def* ε*-of-def* δ*-of-def m-of-def*)
also have ... ∈ *O*[*?F*](*g*)
  using *10 11 13 14 15 16*
 by (*simp add*:*fun-cong*[*OF s1-of-def*[*symmetric*]] *fun-cong*[*OF s2-of-def*[*symmetric*]] *Let-def*)
   (*intro sum-in-bigo*, *auto*)
also have ... = *O*[*?F*](*?rhs*)
   by (*simp add*:*case-prod-beta′ g1-def g-def n-of-def* ε*-of-def* δ*-of-def m-of-def k-of-def*)
finally show *?thesis* **by** *simp*
qed

end

A Informal proof of correctness for the F_0 algorithm

This appendix contains a detailed informal proof for the new Rounding-KMV algorithm that approximates F_0 introduced in Section 7 for reference. It follows the same reasoning as the formalized proof.

Because of the amplification result about medians (see for example [1, §2.1]) it is enough to show that each of the estimates the median is taken from is within the desired interval with success probability $\frac{2}{3}$. To verify the latter, let $a_1, \ldots, a_m$ be the stream elements, where we assume that the elements are a subset of $\{0, \ldots, n-1\}$ and $0 < \delta < 1$ be the desired relative accuracy.

Let p be the smallest prime such that $p \geq \max(n, 19)$ and let h be a random polynomial over $GF(p)$ with degree strictly less than 2. The algoritm also introduces the internal parameters t, r defined by:

$$t := \lceil 80\delta^{-2} \rceil \qquad\qquad r := 4 \log_2 \lceil \delta^{-1} \rceil + 23$$

The estimate the algorithm obtains is R, defined using:

$$H := \{ \lfloor h(a) \rfloor_r | a \in A \} \qquad R := \begin{cases} tp \left(\min_t(H) \right)^{-1} & \text{if } |H| \geq t \\ |H| & \text{othewise,} \end{cases}$$

where $A := \{a_1, \ldots, a_m\}$, $\min_t(H)$ denotes the t-th smallest element of H and $\lfloor x \rfloor_r$ denotes the largest binary floating point number smaller or equal to x with a mantissa that requires at most r bits to represent.[1] With these definitions, it is possible to state the main theorem as:

$$P(|R - F_0| \leq \delta |F_0|) \geq \frac{2}{3}.$$

which is shown separately in the following two subsections for the cases $F_0 \geq t$ and $F_0 < t$.

A.1 Case $F_0 \geq t$

Let us introduce:

$$H^* := \{h(a)|a \in A\}^{\#} \qquad\qquad R^* := tp \left(\min_t^{\#}(H^*) \right)^{-1}$$

These definitions are modified versions of the definitions for H and R: The set H^* is a multiset, this means that each element also has a multiplicity, counting the number of *distinct* elements of A being mapped by h to the same value. Note that by definition: $|H^*| = |A|$. Similarly the operation $\min_t^{\#}$ obtains the t-th element of the multiset H (taking multiplicities into account). Note also that there is no rounding operation $\lfloor \cdot \rfloor_r$ in the definition of H^*. The key reason for the introduction of these alternative versions of H, R is that it is easier to show probabilistic bounds on the distances $|R^* - F_0|$ and $|R^* - R|$ as opposed to $|R - F_0|$ directly. In particular the plan is to show:

$$P\left(|R^* - F_0| > \delta' F_0 \right) \leq \frac{2}{9}, \text{ and} \tag{1}$$

$$P\left(|R^* - F_0| \leq \delta' F_0 \wedge |R - R^*| > \frac{\delta}{4} F_0 \right) \leq \frac{1}{9} \tag{2}$$

[1] This rounding operation is called *truncate-down* in Isabelle, it is defined in HOL-Library.Float.

where $\delta' := \frac{3}{4}\delta$. I.e. the probability that R^* has not the relative accuracy of $\frac{3}{4}\delta$ is less that $\frac{2}{9}$ and the probability that assuming R^* has the relative accuracy of $\frac{3}{4}\delta$ but that R deviates by more that $\frac{1}{4}\delta F_0$ is at most $\frac{1}{9}$. Hence, the probability that neither of these events happen is at least $\frac{2}{3}$ but in that case:

$$|R - F_0| \le |R - R^*| + |R^* - F_0| \le \frac{\delta}{4}F_0 + \frac{3\delta}{4}F_0 = \delta F_0. \tag{3}$$

Thus we only need to show Equation 1 and 2. For the verification of Equation 1 let

$$Q(u) = |\{h(a) < u \mid a \in A\}|$$

and observe that $\min_t^{\#}(H^*) < u$ if $Q(u) \ge t$ and $\min_t^{\#}(H^*) \ge v$ if $Q(v) \le t - 1$. To see why this is true note that, if at least t elements of A are mapped by h below a certain value, then the t-smallest element must also be within them, and thus also be below that value. And that the opposite direction of this conclusion is also true. Note that this relies on the fact that H^* is a multiset and that multiplicities are being taken into account, when computing the t-th smallest element. Alternatively, it is also possible to write $Q(u) = \sum_{a\in A} 1_{\{h(a)<u\}}$[2], i.e., Q is a sum of pairwise independent $\{0,1\}$-valued random variables, with expectation $\frac{u}{p}$ and variance $\frac{u}{p} - \frac{u^2}{p^2}$. [3] Using lineariy of expectation and Bienaymé's identity, it follows that $\operatorname{Var} Q(u) \le \operatorname{E} Q(u) = |A|up^{-1} = F_0 up^{-1}$ for $u \in \{0,\ldots,p\}$.

For $v = \left\lfloor \frac{tp}{(1-\delta')F_0} \right\rfloor$ it is possible to conclude:

$$t - 1 \le^{4} \frac{t}{(1-\delta')} - 3\sqrt{\frac{t}{(1-\delta')}} - 1 \le \frac{F_0 v}{p} - 3\sqrt{\frac{F_0 v}{p}} \le \operatorname{E}Q(v) - 3\sqrt{\operatorname{Var}Q(v)}$$

and thus using Tchebyshev's inequality:

$$\begin{aligned} P\left(R^* < \left(1-\delta'\right)F_0\right) &= P\left(\operatorname{rank}_t^{\#}(H^*) > \frac{tp}{(1-\delta')F_0}\right) \\ &\le P(\operatorname{rank}_t^{\#}(H^*) \ge v) = P(Q(v) \le t-1) \\ &\le P\left(Q(v) \le \operatorname{E}Q(v) - 3\sqrt{\operatorname{Var}Q(v)}\right) \le \frac{1}{9}. \end{aligned} \tag{4}$$

Similarly for $u = \left\lceil \frac{tp}{(1+\delta')F_0} \right\rceil$ it is possible to conclude:

$$t \ge \frac{t}{(1+\delta')} + 3\sqrt{\frac{t}{(1+\delta')}} + 1 + 1 \ge \frac{F_0 u}{p} + 3\sqrt{\frac{F_0 u}{p}} \ge \operatorname{E}Q(u) + 3\sqrt{\operatorname{Var}Q(v)}$$

[2] The notation 1_A is shorthand for the indicator function of A, i.e., $1_A(x) = 1$ if $x \in A$ and 0 otherwise.

[3] A consequence of h being chosen uniformly from a 2-independent hash family.

[4] The verification of this inequality is a lengthy but straightforward calculcation using the definition of δ' and t.

and thus using Tchebyshev's inequality:

$$\begin{aligned} P\left(R^* > \left(1+\delta'\right)F_0\right) &= P\left(\text{rank}_t^{\#}(H^*) < \frac{tp}{(1+\delta')F_0}\right) \\ &\leq P(\text{rank}_t^{\#}(H^*) < u) = P(Q(u) \geq t) \\ &\leq P\left(Q(u) \geq \mathrm{E}Q(u) + 3\sqrt{\mathrm{Var}Q(u)}\right) \leq \frac{1}{9}. \end{aligned} \tag{5}$$

Note that Equation 4 and 5 confirm Equation 1. To verfiy Equation 2, note that

$$\min_t(H) = \lfloor \min_t^{\#}(H^*) \rfloor_r \tag{6}$$

if there are no collisions, induced by the application of $\lfloor h(\cdot) \rfloor_r$ on the elements of A. Even more carefully, note that the equation would remain true, as long as there are no collision within the smallest t elements of H^*. Because Equation 2 needs to be shown only in the case where $R^* \geq (1-\delta')F_0$, i.e., when $\min_t^{\#}(H^*) \leq v$, it is enough to bound the probability of a collision in the range $[0; v]$. Moreover Equation 6 implies $|\min_t(H) - \min_t^{\#}(H^*)| \leq \max(\min_t^{\#}(H^*), \min_t(H))2^{-r}$ from which it is possible to derive $|R^* - R| \leq \frac{\delta}{4}F_0$. Another important fact is that h is injective with probability $1 - \frac{1}{p}$, this is because h is chosen uniformly from the polynomials of degree less than 2. If it is a degree 1 polynomial it is a linear function on $GF(p)$ and thus injective. Because $p \geq 18$ the probability that h is not injective can be bounded by $1/18$. With these in mind, we can conclude:

$$\begin{aligned} & P\left(|R^* - F_0| \leq \delta' F_0 \wedge |R - R^*| > \frac{\delta}{4}F_0\right) \\ \leq\ & P\left(R^* \geq (1-\delta')F_0 \wedge \min_t^{\#}(H^*) \neq \min_t(H) \wedge h \text{ inj.}\right) + P(\neg h \text{ inj.}) \\ \leq\ & P\left(\exists a \neq b \in A. \lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \leq v \wedge h(a) \neq h(b)\right) + \frac{1}{18} \\ \leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(\lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \leq v \wedge h(a) \neq h(b)\right) \\ \leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(|h(a) - h(b)| \leq v2^{-r} \wedge h(a) \leq v(1+2^{-r}) \wedge h(a) \neq h(b)\right) \\ \leq\ & \frac{1}{18} + \sum_{a \neq b \in A} \sum_{\substack{a',b' \in \{0,\ldots,p-1\} \wedge a' \neq b' \\ |a'-b'| \leq v2^{-r} \wedge a' \leq v(1+2^{-r})}} P(h(a) = a')P(h(b) = b') \\ \leq\ & \frac{1}{18} + \frac{5F_0^2 v^2}{2p^2} 2^{-r} \leq \frac{1}{9}. \end{aligned}$$

which shows that Equation 2 is true.

A.2 Case $F_0 < t$

Note that in this case $|H| \leq F_0 < t$ and thus $R = |H|$, hence the goal is to show that: $P(|H| \neq F_0) \leq \frac{1}{3}$. The latter can only happen, if there is a collision induced by the application of $\lfloor h(\cdot) \rfloor_r$. As before h is not injective with probability at most $\frac{1}{18}$, hence:

$$
\begin{aligned}
& P\left(|R - F_0| > \delta F_0\right) \leq P\left(R \neq F_0\right) \\
\leq \quad & \frac{1}{18} + P\left(R \neq F_0 \wedge h \text{ inj.}\right) \\
\leq \quad & \frac{1}{18} + P\left(\exists a \neq b \in A. \lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \wedge h \text{ inj.}\right) \\
\leq \quad & \frac{1}{18} + \sum_{a \neq b \in A} P\left(\lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \wedge h(a) \neq h(b)\right) \\
\leq \quad & \frac{1}{18} + \sum_{a \neq b \in A} P\left(|h(a) - h(b)| \leq p2^{-r} \wedge h(a) \neq h(b)\right) \\
\leq \quad & \frac{1}{18} + \sum_{a \neq b \in A} \sum_{\substack{a',b' \in \{0,\ldots,p-1\} \\ a' \neq b' \wedge |a'-b'| \leq p2^{-r}}} P(h(a) = a')P(h(b) = b') \\
\leq \quad & \frac{1}{18} + F_0^2 2^{-r+1} \leq \frac{1}{18} + t^2 2^{-r+1} \leq \frac{1}{9}.
\end{aligned}
$$

Which concludes the proof. □

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 J. D. P. Rolim and S. Vadhan, editors, *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002.