Formalization of Randomized Approximation Algorithms for Frequency Moments

Emin Karayel

January 18, 2022

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 spage 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, which is for example critical information for parallel processing. The frequency moment of a data stream $a_1, \ldots, a_m \in U$ can be defined as $F_k := \sum_{u \in U} C(u, a)^k$ where $C(u, a)$ is the count of occurences of u in the stream a. They introduce both lower bounds and upper bounds, which were later improved by newer publications. The algorithms have guaranteed success probability and accuracy, without making any assumptions on the input distribution. They are an interesting use-case for formal verification, because they rely on deep results from both algebra and analysis, require a large body of existing results. This work contains the formal verification of three algorithms for the approximation of F_0, F_2 and F_k for $k \geq 3$. To achieve it, the formalization also includes reusable components common to all algorithms, such as universal hash families, the median method, formal modelling of one-pass data stream algorithms and a generic flexible encoding library for the verification of space complexities.

Contents

1 Encoding

theory *Encoding*
 imports *Main HOL−Library.Sublist HOL−Library.Extended-Real HOL−Library.FuncSet*

HOL.Transcendental
begin

This section contains a flexible library for encoding high level data structures into bit strings. The library defines encoding functions for primitive types, as well as combinators to build encodings for more complex types. It is used to measure the size of the data structures.

fun *is-prefix* **where**

```
  is-prefix (Some x) (Some y) = prefix x y |
  is-prefix - - = False

type-synonym 'a encoding = 'a ⇀ bool list

definition is-encoding :: 'a encoding ⇒ bool
  where is-encoding f = (∀ x y. is-prefix (f x) (f y) ⟶ x = y)

lemma encoding-imp-inj:
  assumes is-encoding f
  shows inj-on f (dom f)
  ⟨proof⟩

definition decode where
  decode f t = (
    if (∃!z. is-prefix (f z) (Some t)) then
      (let z = (THE z. is-prefix (f z) (Some t)) in (z, drop (length (the (f z))) t))
    else
      (undefined, t)
    )

lemma decode-elim:
  assumes is-encoding f
  assumes f x = Some r
  shows decode f (r@r1) = (x,r1)
⟨proof⟩

lemma decode-elim-2:
  assumes is-encoding f
  assumes x ∈ dom f
  shows decode f (the (f x)@r1) = (x,r1)
  ⟨proof⟩

lemma snd-decode-suffix:
  suffix (snd (decode f t)) t
⟨proof⟩

lemma snd-decode-len:
  assumes decode f t = (u,v)
  shows length v ≤ length t
  ⟨proof⟩

lemma encoding-by-witness:
  assumes ⋀x y. x ∈ dom f ⟹ g (the (f x)@y) = (x,y)
  shows is-encoding f
⟨proof⟩

fun bit-count :: bool list option ⇒ ereal where
  bit-count None = ∞ |
```

bit-count (*Some x*) = *ereal* (*length x*)

fun *append-encoding* :: *bool list option* ⇒ *bool list option* ⇒ *bool list option* (**infixr** $@_S$ *65*)
where
append-encoding (*Some x*) (*Some y*) = *Some* (*x@y*) |
append-encoding - - = *None*

lemma *bit-count-append*: *bit-count* (*x1*$@_S$*x2*) = *bit-count x1* + *bit-count x2*
⟨*proof*⟩

Encodings for lists

fun $list_S$ **where**
$list_S$ *f* [] = *Some* [*False*] |
$list_S$ *f* (*x#xs*) = *Some* [*True*]$@_S$*f x*$@_S$$list_S$ *f xs*

function *decode-list* :: (*'a* ⇒ *bool list option*) ⇒ *bool list*
⇒ *'a list* × *bool list*
where
decode-list e (*True#x0*) = (
let (*r1*,*x1*) = *decode e x0 in* (
let (*r2*,*x2*) = *decode-list e x1 in* (*r1#r2*,*x2*))) |
decode-list e (*False#x0*) = ([], *x0*) |
decode-list e [] = *undefined*
⟨*proof*⟩
termination
⟨*proof*⟩

lemma *list-encoding-dom*:
assumes *set l* ⊆ *dom f*
shows *l* ∈ *dom* ($list_S$ *f*)
⟨*proof*⟩

lemma *list-bit-count*:
bit-count ($list_S$ *f xs*) = (∑ *x* ← *xs*. *bit-count* (*f x*) + *1*) + *1*
⟨*proof*⟩

lemma *list-bit-count-est*:
assumes ⋀*x*. *x* ∈ *set xs* ⟹ *bit-count* (*f x*) ≤ *a*
shows *bit-count* ($list_S$ *f xs*) ≤ *ereal* (*length xs*) ∗ (*a*+*1*) + *1*
⟨*proof*⟩

lemma *list-bit-count-estI*:
assumes ⋀*x*. *x* ∈ *set xs* ⟹ *bit-count* (*f x*) ≤ *a*
assumes *ereal* (*real* (*length xs*)) ∗ (*a*+*1*) + *1* ≤ *h*
shows *bit-count* ($list_S$ *f xs*) ≤ *h*
⟨*proof*⟩

lemma *list-encoding-aux*:

assumes *is-encoding f*
shows $x \in dom\ (list_S\ f) \Longrightarrow decode\text{-}list\ f\ (the\ (list_S\ f\ x)\ @\ y) = (x,\ y)$
⟨*proof*⟩

lemma *list-encoding*:
assumes *is-encoding f*
shows *is-encoding* $(list_S\ f)$
⟨*proof*⟩

Encoding for natural numbers

fun *nat-encoding-aux* :: *nat* ⇒ *bool list*
where
nat-encoding-aux 0 = [*False*] |
nat-encoding-aux (*Suc n*) = *True*#(*odd n*)#*nat-encoding-aux* (*n div 2*)

fun N_S **where** $N_S\ n = Some\ (nat\text{-}encoding\text{-}aux\ n)$

fun *decode-nat* :: *bool list* ⇒ *nat* × *bool list*
where
decode-nat (*False*#*y*) = (*0*,*y*) |
decode-nat (*True*#*x*#*xs*) =
(*let* (*n*, *rs*) = *decode-nat xs in* ($n * 2 + 1 +$ (*if x then 1 else 0*), *rs*)) |
decode-nat - = *undefined*

lemma *nat-encoding-aux*:
decode-nat (*nat-encoding-aux x* @ *y*) = (*x*, *y*)
⟨*proof*⟩

lemma *nat-encoding*:
shows *is-encoding* N_S
⟨*proof*⟩

lemma *nat-bit-count*:
$bit\text{-}count\ (N_S\ n) \le 2 * log\ 2\ (real\ n{+}1) + 1$
⟨*proof*⟩

lemma *nat-bit-count-est*:
assumes $n \le m$
shows $bit\text{-}count\ (N_S\ n) \le 2 * log\ 2\ (1{+}real\ m) + 1$
⟨*proof*⟩

Encoding for integers

fun I_S :: *int* ⇒ *bool list option*
where
$I_S\ n = (if\ n \ge 0\ then\ Some\ [True]@_S N_S\ (nat\ n)\ else\ Some\ [False]@_S\ (N_S\ (nat\ (-n-1))))$

fun *decode-int* :: *bool list* ⇒ (*int* × *bool list*)
where

  decode-int (*True#xs*) = (λ(*x*::*nat*,*y*). (*int x*, *y*)) (*decode-nat xs*) |
  decode-int (*False#xs*) = (λ(*x*::*nat*,*y*). ($-$(*int x*)$-$*1*, *y*)) (*decode-nat xs*) |
  decode-int [] = *undefined*

lemma *int-encoding*: *is-encoding* I_S
  ⟨*proof*⟩

lemma *int-bit-count*:
  bit-count (I_S *x*) $\leq$ *2* $*$ *log 2* (|*x*|+*1*) + *2*
⟨*proof*⟩

lemma *int-bit-count-est*:
  assumes *abs n* $\leq$ *m*
  shows *bit-count* (I_S *n*) $\leq$ *2* $*$ *log 2* (*m*+*1*) + *2*
⟨*proof*⟩

Encoding for Cartesian products

fun *encode-prod* :: *'a encoding* $\Rightarrow$ *'b encoding* $\Rightarrow$ (*'a* $\times$ *'b*) *encoding* (**infixr** $\times_S$ *65*)
  where
    encode-prod e1 e2 x = *e1* (*fst x*)$@_S$ *e2* (*snd x*)

fun *decode-prod* :: *'a encoding* $\Rightarrow$ *'b encoding* $\Rightarrow$ *bool list* $\Rightarrow$ (*'a* $\times$ *'b*) $\times$ *bool list*
  where
    decode-prod e1 e2 x0 = (
      let (*r1*,*x1*) = *decode e1 x0 in* (
        let (*r2*,*x2*) = *decode e2 x1 in* ((*r1*,*r2*),*x2*)))

lemma *prod-encoding-dom*:
  x $\in$ *dom* (*e1* $\times_S$ *e2*) = (*fst x* $\in$ *dom e1* $\wedge$ *snd x* $\in$ *dom e2*)
  ⟨*proof*⟩

lemma *prod-encoding*:
  assumes *is-encoding e1*
  assumes *is-encoding e2*
  shows *is-encoding* (*encode-prod e1 e2*)
⟨*proof*⟩

lemma *prod-bit-count*:
  bit-count ((e_1 $\times_S$ e_2) (x_1,x_2)) = *bit-count* (e_1 x_1) + *bit-count* (e_2 x_2)
  ⟨*proof*⟩

lemma *prod-bit-count-2*:
  bit-count ((*e1* $\times_S$ *e2*) *x*) = *bit-count* (*e1* (*fst x*)) + *bit-count* (*e2* (*snd x*))
  ⟨*proof*⟩

Encoding for dependent sums

fun *encode-dependent-sum* :: *'a encoding* $\Rightarrow$ (*'a* $\Rightarrow$ *'b encoding*) $\Rightarrow$ (*'a* $\times$ *'b*) *encoding* (**infixr** $\times_D$ *65*)
  where

encode-dependent-sum e1 e2 x = *e1* (*fst x*)$@_S$ *e2* (*fst x*) (*snd x*)

lemma *dependent-encoding*:
assumes *is-encoding e1*
assumes $\bigwedge x$. *is-encoding* (*e2 x*)
shows *is-encoding* (*encode-dependent-sum e1 e2*)
⟨*proof*⟩

lemma *dependent-bit-count*:
bit-count (($e_1 \times_D e_2$) (x_1,x_2)) = *bit-count* (e_1 x_1) + *bit-count* (e_2 x_1 x_2)
⟨*proof*⟩

This lemma helps derive an encoding on the domain of an injective function using an existing encoding on its image.

lemma *encoding-compose*:
assumes *is-encoding f*
assumes *inj-on g* {*x*. *P x*}
shows *is-encoding* (λ*x*. *if P x then f* (*g x*) *else None*)
⟨*proof*⟩

Encoding for extensional maps defined on an enumerable set.

definition fun_S :: *'a list* ⇒ *'b encoding* ⇒ (*'a* ⇒ *'b*) *encoding* (**infixr** $\rightarrow_S$ *65*)
where
fun_S *xs e f* = (
  if f ∈ *extensional* (*set xs*) *then*
    $list_S$ *e* (*map f xs*)
  else
    None)

lemma *encode-extensional*:
assumes *is-encoding e*
shows *is-encoding* (λ*x*. (*xs* $\rightarrow_S$ *e*) *x*)
⟨*proof*⟩

lemma *extensional-bit-count*:
assumes *f* ∈ *extensional* (*set xs*)
shows *bit-count* ((*xs* $\rightarrow_S$ *e*) *f*) = ($\sum$ *x* ← *xs*. *bit-count* (*e* (*f x*)) + *1*) + *1*
⟨*proof*⟩

Encoding for ordered sets.

fun set_S **where** set_S *e S* = (*if finite S then* $list_S$ *e* (*sorted-list-of-set S*) *else None*)

lemma *encode-set*:
assumes *is-encoding e*
shows *is-encoding* (λ*S*. set_S *e S*)
⟨*proof*⟩

lemma *set-bit-count*:
assumes *finite S*

shows *bit-count* $(set_S\ e\ S) = (\sum x \in S.\ bit\text{-}count\ (e\ x){+}1){+}1$
⟨*proof*⟩

lemma *set-bit-count-est*:
assumes *finite S*
assumes *card* $S \le m$
assumes $0 \le a$
assumes $\bigwedge x.\ x \in S \Longrightarrow bit\text{-}count\ (f\ x) \le a$
shows *bit-count* $(set_S\ f\ S) \le ereal\ (real\ m) * (a{+}1) + 1$
⟨*proof*⟩

end

2 Field

theory *Field*
imports *Main HOL−Algebra.Ring-Divisibility HOL−Algebra.IntRing*
begin

This section contains a proof that the factor ring *ZFact p* for *prime p* is a field. Note that the bulk of the work has already been done in HOL-Algebra, in particular it is established that *ZFact p* is a domain.

However, any domain with a finite carrier is already a field. This can be seen by establishing that multiplication by a non-zero element is an injective map between the elements of the carrier of the domain. But an injective map between sets of the same non-finite cardinality is also surjective. Hence we can find the unit element in the image of such a map.

Additionally the canonical bijection between *ZFact p* and $\{0..{<}p\}$ is introduced, which is useful for hashing natural numbers.

definition *zfact-embed* :: *nat* ⇒ *nat* ⇒ *int set* **where**
zfact-embed p k $= Idl_{\mathcal{Z}}\ \{int\ p\}\ {+}{>}_{\mathcal{Z}}\ (int\ k)$

lemma *zfact-embed-ran*:
assumes $p > 0$
shows *zfact-embed p* ‘ $\{0..{<}p\}$ = *carrier* (*ZFact p*)
⟨*proof*⟩

lemma *zfact-embed-inj*:
assumes $p > 0$
shows *inj-on* (*zfact-embed p*) $\{0..{<}p\}$
⟨*proof*⟩

lemma *zfact-embed-bij*:
assumes $p > 0$
shows *bij-betw* (*zfact-embed p*) $\{0..{<}p\}$ (*carrier* (*ZFact p*))
⟨*proof*⟩

lemma *zfact-card*:
  assumes $(p :: nat) > 0$
  shows *card* (*carrier* (*ZFact* (*int p*))) = *p*
  ⟨*proof*⟩

lemma *zfact-finite*:
  assumes $(p :: nat) > 0$
  shows *finite* (*carrier* (*ZFact* (*int p*)))
  ⟨*proof*⟩

lemma *finite-domains-are-fields*:
  assumes *domain R*
  assumes *finite* (*carrier R*)
  shows *field R*
⟨*proof*⟩

lemma *zfact-prime-is-field*:
  assumes *prime* (*p* :: *nat*)
  shows *field* (*ZFact* (*int p*))
⟨*proof*⟩

end

3 Float

This section contains results about floating point numbers in addition to "HOL-Library.Float"

theory *Float-Ext*
  imports *HOL−Library.Float Encoding*
begin

lemma *round-down-ge*:
  $x \leq$ *round-down prec x* + *2 powr* (−*prec*)
  ⟨*proof*⟩

lemma *truncate-down-ge*:
  $x \leq$ *truncate-down prec x* + *abs x* ∗ *2 powr* (−*prec*)
⟨*proof*⟩

lemma *truncate-down-pos*:
  assumes $x \geq 0$
  shows *x* ∗ (*1* − *2 powr* (−*prec*)) ≤ *truncate-down prec x*
  ⟨*proof*⟩

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*⟩

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*⟩

Definition of an encoding for floating point numbers.

definition F_S **where** F_S *f* = (I_S $\times_S$ I_S) (*mantissa f*,*exponent f*)

lemma *encode-float*:
  is-encoding F_S
⟨*proof*⟩

lemma *truncate-mantissa-bound*:
  abs (⌊*x* ∗ *2 powr* (*real r* − *real-of-int* ⌊*log 2* |*x*|⌋)⌋) ≤ *2* ^ (*r*+*1*) (**is** *?lhs* ≤ -)
⟨*proof*⟩

lemma *suc-n-le-2-pow-n*:
  fixes *n* :: *nat*
  shows *n* + *1* ≤ *2* ^ *n*
  ⟨*proof*⟩

lemma *float-bit-count*:
  fixes *m* :: *int*
  fixes *e* :: *int*
  defines *f* ≡ *float-of* (*m* ∗ *2 powr e*)
  shows *bit-count* (F_S *f*) ≤ *4* + *2* ∗ (*log 2* (|*m*| + *2*) + *log 2* (|*e*| + *1*))
⟨*proof*⟩

lemma *float-bit-count-zero*:
  bit-count (F_S (*float-of 0*)) = *4*
  ⟨*proof*⟩

lemma *log-est*: *log 2* (*real n* + *1*) ≤ *n*
⟨*proof*⟩

lemma *truncate-float-bit-count*:
  bit-count (F_S (*float-of* (*truncate-down r x*))) ≤ *8* + *4* ∗ *real r* + *2*∗*log 2* (*2* + *abs* (*log 2* (*abs x*)))
  (**is** *?lhs* ≤ *?rhs*)
⟨*proof*⟩

end

4 Lists

theory *List-Ext*
  imports *Main HOL.List*
begin

This section contains results about lists in addition to "HOL.List"

lemma *count-list-gr-1*:
  $(x \in \mathit{set}\ xs) = (\mathit{count\text{-}list}\ xs\ x \geq 1)$
  ⟨*proof*⟩

lemma *count-list-append*: $\mathit{count\text{-}list}\ (xs@ys)\ v = \mathit{count\text{-}list}\ xs\ v + \mathit{count\text{-}list}\ ys\ v$
  ⟨*proof*⟩

lemma *count-list-card*: $\mathit{count\text{-}list}\ xs\ x = \mathit{card}\ \{k.\ k < \mathit{length}\ xs \wedge xs\ !\ k = x\}$
⟨*proof*⟩

lemma *card-gr-1-iff*:
  assumes *finite S*
  assumes $x \in S$
  assumes $y \in S$
  assumes $x \neq y$
  shows $\mathit{card}\ S > 1$
  ⟨*proof*⟩

lemma *count-list-ge-2-iff*:
  assumes $y < z$
  assumes $z < \mathit{length}\ xs$
  assumes $xs\ !\ y = xs\ !\ z$
  shows $\mathit{count\text{-}list}\ xs\ (xs\ !\ y) > 1$
  ⟨*proof*⟩

end

5 Frequency Moments

theory *Frequency-Moments*
  imports *Main HOL.List HOL.Rat List-Ext*
begin

This section contains a definition of the frequency moments of a stream.

definition *F* **where**
  $F\ k\ xs = (\sum\ x \in \mathit{set}\ xs.\ (\mathit{rat\text{-}of\text{-}nat}\ (\mathit{count\text{-}list}\ xs\ x)\hat{}k))$

lemma *F-gr-0*:
  assumes $as \neq []$
  shows $F\ k\ as > 0$
⟨*proof*⟩

end

6 Primes

This section introduces a function that finds the smallest primes above a given threshold.

theory *Primes-Ext*
imports *Main HOL−Computational-Algebra.Primes Bertrands-Postulate.Bertrand*

begin

lemma *inf-primes*: $wf\ ((\lambda n.\ (Suc\ n,\ n))\ `\ \{n.\ \neg\ (prime\ n)\})$ (**is** *wf ?S*)
⟨*proof*⟩

function *find-prime-above* :: *nat* ⇒ *nat* **where**
  find-prime-above n = (*if prime n then n else find-prime-above* (*Suc n*))
  ⟨*proof*⟩
termination
  ⟨*proof*⟩

declare *find-prime-above.simps* [*simp del*]

lemma *find-prime-above-is-prime*:
  prime (*find-prime-above n*)
  ⟨*proof*⟩

lemma *find-prime-above-min*:
  find-prime-above n ≥ *2*
  ⟨*proof*⟩

lemma *find-prime-above-lower-bound*:
  find-prime-above n ≥ *n*
  ⟨*proof*⟩

lemma *find-prime-above-upper-boundI*:
  assumes *prime m*
  shows *n* ≤ *m* ⟹ *find-prime-above n* ≤ *m*
⟨*proof*⟩

lemma *find-prime-above-upper-bound*:
  find-prime-above n ≤ *2*∗*n*+*2*
⟨*proof*⟩

end

7 Multisets

theory *Multiset-Ext*
  imports *Main HOL.Real HOL−Library.Multiset*
begin

This section contains results about multisets in addition to "HOL.Multiset"

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 $\bigwedge n\ M\ x.\ P\ M \Longrightarrow \neg(x \in\#\ M) \Longrightarrow n > 0 \Longrightarrow P\ (M + \mathit{replicate\text{-}mset}\ n\ x)$
  shows $P\ M$
⟨*proof*⟩

lemma *prod-mset-conv*:
  fixes $f :: 'a \Rightarrow 'b::\{\mathit{comm\text{-}monoid\text{-}mult}\}$
  shows $\mathit{prod\text{-}mset}\ (\mathit{image\text{-}mset}\ f\ A) = \mathit{prod}\ (\lambda x.\ f\ x\widehat{\ }(\mathit{count}\ A\ x))\ (\mathit{set\text{-}mset}\ A)$
⟨*proof*⟩

lemma *sum-collapse*:
  fixes $f :: 'a \Rightarrow 'b::\{\mathit{comm\text{-}monoid\text{-}add}\}$
  assumes *finite A*
  assumes $z \in A$
  assumes $\bigwedge y.\ y \in A \Longrightarrow y \neq z \Longrightarrow f\ y = 0$
  shows $\mathit{sum}\ f\ A = f\ z$
  ⟨*proof*⟩

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 $f :: 'a \Rightarrow 'b::\{\mathit{ring},\mathit{semiring\text{-}1}\}$
  shows $\mathit{sum\text{-}list}\ (\mathit{map}\ f\ \mathit{xs}) = (\sum x \in \mathit{set}\ \mathit{xs}.\ \mathit{of\text{-}nat}\ (\mathit{count\text{-}list}\ \mathit{xs}\ x) * f\ x)$
⟨*proof*⟩

lemma *prod-list-eval*:
  fixes $f :: 'a \Rightarrow 'b::\{\mathit{ring},\mathit{semiring\text{-}1},\mathit{comm\text{-}monoid\text{-}mult}\}$
  shows $\mathit{prod\text{-}list}\ (\mathit{map}\ f\ \mathit{xs}) = (\prod x \in \mathit{set}\ \mathit{xs}.\ (f\ x)\widehat{\ }(\mathit{count\text{-}list}\ \mathit{xs}\ x))$
⟨*proof*⟩

lemma *sorted-sorted-list-of-multiset*: *sorted* (*sorted-list-of-multiset M*)
  ⟨*proof*⟩

lemma *count-mset*: *count* (*mset xs*) *a* = *count-list xs a*
  ⟨*proof*⟩

lemma *swap-filter-image*: *filter-mset g* (*image-mset f A*) = *image-mset f* (*filter-mset* (*g* ∘ *f*) *A*)
  ⟨*proof*⟩

lemma *list-eq-iff*:
  assumes *mset xs* = *mset ys*
  assumes *sorted xs*
  assumes *sorted ys*
  shows *xs* = *ys*
  ⟨*proof*⟩

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*) (**is** *?A* = *?B*)
  ⟨*proof*⟩

end

8 Probability Spaces

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

theory *Probability-Ext*
  imports *Main HOL−Probability.Independent-Family Multiset-Ext HOL−Probability.Stream-Space HOL−Probability.Probability-Mass-Function*
begin

lemma *measure-inters*: *measure M* (*E* ∩ *space M*) = $\mathcal{P}$(*x in M*. *x* ∈ *E*)
  ⟨*proof*⟩

lemma *set-comp-subsetI*: (⋀*x*. *P x* ⟹ *f x* ∈ *B*) ⟹ {*f x*|*x*. *P x*} ⊆ *B*
  ⟨*proof*⟩

lemma *set-comp-cong*:
  assumes ⋀*x*. *P x* ⟹ *f x* = *h* (*g x*)
  shows {*f x*| *x*. *P x*} = *h* ‘ {*g x*| *x*. *P x*}
  ⟨*proof*⟩

lemma *indep-sets-distr*:
  assumes *f* ∈ *measurable M N*
  assumes *prob-space M*
  assumes *prob-space.indep-sets M* (λ*i*. (λ*a*. *f* −‘ *a* ∩ *space M*) ‘ *A i*) *I*
  assumes ⋀*i*. *i* ∈ *I* ⟹ *A i* ⊆ *sets N*
  shows *prob-space.indep-sets* (*distr M N f*) *A I*
⟨*proof*⟩

lemma *indep-vars-distr*:
  assumes *f* ∈ *measurable M N*

assumes $\bigwedge i.\ i \in I \Longrightarrow X'\ i \in \mathit{measurable}\ N\ (M'\ i)$
assumes $\mathit{prob\text{-}space.indep\text{-}vars}\ M\ M'\ (\lambda i.\ (X'\ i) \circ f)\ I$
assumes $\mathit{prob\text{-}space}\ M$
shows $\mathit{prob\text{-}space.indep\text{-}vars}\ (\mathit{distr}\ M\ N\ f)\ M'\ X'\ I$
$\langle \mathit{proof} \rangle$

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

lemma *make-ext*:
assumes $\bigwedge x.\ P\ x = P\ (\mathit{restrict}\ x\ I)$
shows $(\forall x \in \mathit{Pi}\ I\ A.\ P\ x) = (\forall x \in \mathit{PiE}\ I\ A.\ P\ x)$
$\langle \mathit{proof} \rangle$

lemma *PiE-reindex*:
assumes *inj-on f I*
shows $\mathit{PiE}\ I\ (A \circ f) = (\lambda a.\ \mathit{restrict}\ (a \circ f)\ I)\ `\ \mathit{PiE}\ (f\ `\ I)\ A$ (**is** $?lhs = ?f\ `\ ?rhs$)
$\langle \mathit{proof} \rangle$

lemma (**in** *prob-space*) *indep-sets-reindex*:
assumes *inj-on f I*
shows $\mathit{indep\text{-}sets}\ A\ (f\ `\ I) = \mathit{indep\text{-}sets}\ (\lambda i.\ A\ (f\ i))\ I$
$\langle \mathit{proof} \rangle$

lemma (**in** *prob-space*) *indep-vars-reindex*:
assumes *inj-on f I*
assumes $\mathit{indep\text{-}vars}\ M'\ X'\ (f\ `\ I)$
shows $\mathit{indep\text{-}vars}\ (M' \circ f)\ (\lambda k\ \omega.\ X'\ (f\ k)\ \omega)\ I$
$\langle \mathit{proof} \rangle$

lemma (**in** *prob-space*) *variance-divide*:
fixes $f :: 'a \Rightarrow \mathit{real}$
assumes *integrable M f*
shows $\mathit{variance}\ (\lambda\omega.\ f\ \omega\ /\ r) = \mathit{variance}\ f\ /\ r\hat{}2$
$\langle \mathit{proof} \rangle$

lemma *pmf-eq*:
assumes $\bigwedge x.\ x \in \mathit{set\text{-}pmf}\ \Omega \Longrightarrow (x \in P) = (x \in Q)$
shows $\mathit{measure}\ (\mathit{measure\text{-}pmf}\ \Omega)\ P = \mathit{measure}\ (\mathit{measure\text{-}pmf}\ \Omega)\ Q$
$\langle \mathit{proof} \rangle$

lemma *pmf-mono-1*:
assumes $\bigwedge x.\ x \in P \Longrightarrow x \in \mathit{set\text{-}pmf}\ \Omega \Longrightarrow x \in Q$
shows $\mathit{measure}\ (\mathit{measure\text{-}pmf}\ \Omega)\ P \le \mathit{measure}\ (\mathit{measure\text{-}pmf}\ \Omega)\ Q$
$\langle \mathit{proof} \rangle$

lemma *pmf-mono-2*:
assumes $\bigwedge \omega.\ \omega \in \mathit{set\text{-}pmf}\ M \Longrightarrow P\ \omega \Longrightarrow Q\ \omega$
shows $\mathcal{P}(\omega\ \mathit{in}\ \mathit{measure\text{-}pmf}\ M.\ P\ \omega) \le \mathcal{P}(\omega\ \mathit{in}\ \mathit{measure\text{-}pmf}\ M.\ Q\ \omega)$

⟨*proof*⟩

lemma *pmf-add*:
assumes $\bigwedge x.\ x \in P \Longrightarrow x \in \textit{set-pmf}\ \Omega \Longrightarrow x \in Q \vee x \in R$
shows $\textit{measure}\ (\textit{measure-pmf}\ \Omega)\ P \leq \textit{measure}\ (\textit{measure-pmf}\ \Omega)\ Q + \textit{measure}\ (\textit{measure-pmf}\ \Omega)\ R$
⟨*proof*⟩

lemma *pmf-add-2*:
assumes $\mathcal{P}(\omega\ \textit{in measure-pmf}\ \Omega.\ P\ \omega) \leq r1$
assumes $\mathcal{P}(\omega\ \textit{in measure-pmf}\ \Omega.\ Q\ \omega) \leq r2$
shows $\mathcal{P}(\omega\ \textit{in measure-pmf}\ \Omega.\ P\ \omega \vee Q\ \omega) \leq r1 + r2$
⟨*proof*⟩

definition (**in** *prob-space*) *covariance* **where**
$\textit{covariance}\ f\ g = \textit{expectation}\ (\lambda\omega.\ (f\ \omega - \textit{expectation}\ f) * (g\ \omega - \textit{expectation}\ g))$

lemma (**in** *prob-space*) *real-prod-integrable*:
fixes $f\ g :: {'a} \Rightarrow \textit{real}$
assumes [*measurable*]: $f \in \textit{borel-measurable}\ M\ g \in \textit{borel-measurable}\ M$
assumes *sq-int*: $\textit{integrable}\ M\ (\lambda\omega.\ f\ \omega\hat{}2)\ \textit{integrable}\ M\ (\lambda\omega.\ g\ \omega\hat{}2)$
shows $\textit{integrable}\ M\ (\lambda\omega.\ f\ \omega * g\ \omega)$
⟨*proof*⟩

lemma (**in** *prob-space*) *covariance-eq*:
fixes $f :: {'a} \Rightarrow \textit{real}$
assumes $f \in \textit{borel-measurable}\ M\ g \in \textit{borel-measurable}\ M$
assumes $\textit{integrable}\ M\ (\lambda\omega.\ f\ \omega\hat{}2)\ \textit{integrable}\ M\ (\lambda\omega.\ g\ \omega\hat{}2)$
shows $\textit{covariance}\ f\ g = \textit{expectation}\ (\lambda\omega.\ f\ \omega * g\ \omega) - \textit{expectation}\ f * \textit{expectation}\ g$
⟨*proof*⟩

lemma (**in** *prob-space*) *covar-integrable*:
fixes $f\ g :: {'a} \Rightarrow \textit{real}$
assumes $f \in \textit{borel-measurable}\ M\ g \in \textit{borel-measurable}\ M$
assumes $\textit{integrable}\ M\ (\lambda\omega.\ f\ \omega\hat{}2)\ \textit{integrable}\ M\ (\lambda\omega.\ g\ \omega\hat{}2)$
shows $\textit{integrable}\ M\ (\lambda\omega.\ (f\ \omega - \textit{expectation}\ f) * (g\ \omega - \textit{expectation}\ g))$
⟨*proof*⟩

lemma (**in** *prob-space*) *sum-square-int*:
fixes $f :: {'b} \Rightarrow {'a} \Rightarrow \textit{real}$
assumes *finite* I
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in \textit{borel-measurable}\ M$
assumes $\bigwedge i.\ i \in I \Longrightarrow \textit{integrable}\ M\ (\lambda\omega.\ f\ i\ \omega\hat{}2)$
shows $\textit{integrable}\ M\ (\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)^2)$
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-1*:
fixes $f :: {'b} \Rightarrow {'a} \Rightarrow \textit{real}$

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* $(f\ i)\ (f\ j)))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma (**in** *prob-space*) *covar-self-eq*:
fixes f :: $'a \Rightarrow$ *real*
shows *covariance f f = variance f*
⟨*proof*⟩

lemma (**in** *prob-space*) *covar-indep-eq-zero*:
fixes $f\ g$ :: $'a \Rightarrow$ *real*
assumes *integrable M f*
assumes *integrable M g*
assumes *indep-var borel f borel g*
shows *covariance f g = 0*
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-2*:
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* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) =$
$(\sum i \in I.$ *variance* $(f\ i)) + (\sum i \in I.\ \sum j \in I - \{i\}.$ *covariance* $(f\ i)\ (f\ j))$
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-pairwise-indep*:
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* $(\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
assumes $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow i \neq j \Longrightarrow$ *indep-var borel* $(f\ i)$ *borel* $(f\ j)$
shows *variance* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.$ *variance* $(f\ i))$
⟨*proof*⟩

lemma (**in** *prob-space*) *indep-var-from-indep-vars*:
assumes $i \neq j$
assumes *indep-vars* $(\lambda\text{-}.\ M')\ f\ \{i, j\}$
shows *indep-var* $M'\ (f\ i)\ M'\ (f\ j)$
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-pairwise-indep-2*:
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* $(\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
assumes $\bigwedge J.\ J \subseteq I \Longrightarrow$ *card* $J = 2 \Longrightarrow$ *indep-vars* $(\lambda$ -. *borel*$)$ $f\ J$
shows *variance* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.$ *variance* $(f\ i))$
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-all-indep*:
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* $(\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
assumes *indep-vars* $(\lambda$ -. *borel*$)$ $f\ I$
shows *variance* $(\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.$ *variance* $(f\ i))$
⟨*proof*⟩

end

9 Median

theory *Median*
imports *Main HOL−Probability.Hoeffding HOL−Library.Multiset Probability-Ext HOL.List*
begin

This section includes an amplification result for estimation algorithms using the median method.

fun *sort-primitive* **where**
sort-primitive i j f k = (*if k* = *i then min* (*f i*) (*f j*) *else* (*if k* = *j then max* (*f i*) (*f j*) *else f k*))

fun *sort-map* **where**
sort-map f n = *fold id* [*sort-primitive j i. i* <− [*0*..<*n*], *j* <− [*0*..<*i*]] *f*

lemma *sort-map-ind*:
sort-map f (*Suc n*) = *fold id* [*sort-primitive j n. j* <− [*0*..<*n*]] (*sort-map f n*)
⟨*proof*⟩

lemma *sort-map-strict-mono*:
fixes f :: *nat* $\Rightarrow$ $'b$ :: *linorder*
shows $j < n \Longrightarrow i < j \Longrightarrow$ *sort-map f n i* $\leq$ *sort-map f n j*
⟨*proof*⟩

lemma *sort-map-mono*:
fixes f :: *nat* $\Rightarrow$ $'b$ :: *linorder*
shows $j < n \Longrightarrow i \leq j \Longrightarrow$ *sort-map f n i* $\leq$ *sort-map f n j*
⟨*proof*⟩

lemma *sort-map-perm*:
fixes f :: *nat* $\Rightarrow$ $'b$ :: *linorder*
shows *image-mset* (*sort-map f n*) (*mset* [*0*..<*n*]) = *image-mset f* (*mset* [*0*..<*n*])

⟨*proof*⟩

lemma *sort-map-eq-sort*:
  fixes f :: *nat* ⇒ (*'b* :: *linorder*)
  shows *map* (*sort-map* f n) [*0..<n*] = *sort* (*map* f [*0..<n*]) (**is** *?A* = *?B*)
⟨*proof*⟩

definition *median* **where**
  median n f = *sort* (*map* f [*0..<n*]) ! (n *div* *2*)

lemma *median-alt-def*:
  assumes $n > 0$
  shows *median* n f = (*sort-map* f n) (n *div* *2*)
  ⟨*proof*⟩

definition *up-ray* :: (*'a* :: *linorder*) *set* ⇒ *bool* **where**
  up-ray I = ($\forall x\ y.\ x \in I \longrightarrow x \le y \longrightarrow y \in I$)

lemma *up-ray-borel*:
  assumes *up-ray* (I :: ((*'a* :: *linorder-topology*) *set*))
  shows $I \in$ *borel*
⟨*proof*⟩

definition *down-ray* :: (*'a* :: *linorder*) *set* ⇒ *bool* **where**
  down-ray I = ($\forall x\ y.\ y \in I \longrightarrow x \le y \longrightarrow x \in I$)

lemma *down-ray-borel*:
  assumes *down-ray* (I :: ((*'a* :: *linorder-topology*) *set*))
  shows $I \in$ *borel*
⟨*proof*⟩

definition *interval* :: (*'a* :: *linorder*) *set* ⇒ *bool* **where**
  interval I = ($\forall x\ y\ z.\ x \in I \longrightarrow z \in I \longrightarrow x \le y \longrightarrow y \le z \longrightarrow y \in I$)

lemma *interval-borel*:
  assumes *interval* (I :: ((*'a* :: *linorder-topology*) *set*))
  shows $I \in$ *borel*
⟨*proof*⟩

lemma *interval-rule*:
  assumes *interval* I
  assumes $a \le x$ $x \le b$
  assumes $a \in I$
  assumes $b \in I$
  shows $x \in I$
  ⟨*proof*⟩

lemma *sorted-int*:
  assumes *interval* I

assumes *sorted xs*
assumes $k < \mathit{length}\ xs\ i \leq j\ j \leq k$
assumes $xs\ !\ i \in I\ xs\ !\ k \in I$
shows $xs\ !\ j \in I$
⟨*proof*⟩

lemma *mid-in-interval*:
assumes $2{*}\mathit{length}\ (\mathit{filter}\ (\lambda x.\ x \in I)\ xs) > \mathit{length}\ xs$
assumes *interval I*
assumes *sorted xs*
shows $xs\ !\ (\mathit{length}\ xs\ \mathit{div}\ 2) \in I$
⟨*proof*⟩

lemma *median-est*:
assumes *interval I*
assumes $2{*}\mathit{card}\ \{k.\ k < n \land f\ k \in I\} > n$
shows $\mathit{median}\ n\ f \in I$
⟨*proof*⟩

lemma *median-measurable*:
fixes $X :: \mathit{nat} \Rightarrow {}'a \Rightarrow ({}'b :: \{\mathit{linorder}, \mathit{topological\text{-}space}, \mathit{linorder\text{-}topology}, \mathit{second\text{-}countable\text{-}topology}\})$
assumes $n \geq 1$
assumes $\bigwedge i.\ i < n \Longrightarrow X\ i \in \mathit{measurable}\ M\ \mathit{borel}$
shows $(\lambda x.\ \mathit{median}\ n\ (\lambda i.\ X\ i\ x)) \in \mathit{measurable}\ M\ \mathit{borel}$
⟨*proof*⟩

lemma (**in** *prob-space*) *median-bound*:
fixes $n :: \mathit{nat}$
fixes $I :: ({}'b :: \{\mathit{linorder\text{-}topology}, \mathit{second\text{-}countable\text{-}topology}\})\ \mathit{set}$
assumes *interval I*
assumes $\alpha > 0$
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\mathit{indep\text{-}vars}\ (\lambda\text{-}.\ \mathit{borel})\ X\ \{0..{<}n\}$
assumes $n \geq -\ \mathit{ln}\ \varepsilon\ /\ (2 * \alpha^2)$
assumes $\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ \mathit{in}\ M.\ X\ i\ \omega \in I) \geq 1/2{+}\alpha$
shows $\mathcal{P}(\omega\ \mathit{in}\ M.\ \mathit{median}\ n\ (\lambda i.\ X\ i\ \omega) \in I) \geq 1{-}\varepsilon$ (**is** $\mathcal{P}(\omega\ \mathit{in}\ M.\ ?\mathit{lhs}\ \omega) \geq ?C$)
⟨*proof*⟩

lemma (**in** *prob-space*) *median-bound-1*:
fixes $a\ b :: \mathit{real}$
fixes $n :: \mathit{nat}$
assumes $\alpha > 0$
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\mathit{indep\text{-}vars}\ (\lambda\text{-}.\ \mathit{borel})\ X\ \{0..{<}n\}$
assumes $n \geq -\ \mathit{ln}\ \varepsilon\ /\ (2 * \alpha^2)$
assumes $\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ \mathit{in}\ M.\ X\ i\ \omega \in \{a..b\}) \geq 1/2{+}\alpha$
shows $\mathcal{P}(\omega\ \mathit{in}\ M.\ \mathit{median}\ n\ (\lambda i.\ X\ i\ \omega) \in \{a..b\}) \geq 1{-}\varepsilon$ (**is** $\mathcal{P}(\omega\ \mathit{in}\ M.\ ?\mathit{lhs}\ \omega)$

$\geq$ *?C*)
  ⟨*proof*⟩

lemma (**in** *prob-space*) *median-bound-2*:
  fixes μ :: *real*
  fixes δ :: *real*
  assumes $\varepsilon \in \{0{<}..{<}1\}$
  assumes *indep-vars* (λ-. *borel*) *X* $\{0..{<}n\}$
  assumes $n \geq -18 * \ln \varepsilon$
  assumes $\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ in\ M.\ abs\ (X\ i\ \omega - \mu) > \delta) \leq 1/3$
  shows $\mathcal{P}(\omega\ in\ M.\ abs\ (median\ n\ (\lambda i.\ X\ i\ \omega) - \mu) \leq \delta) \geq 1{-}\varepsilon$
⟨*proof*⟩

lemma *sorted-mono-map*:
  assumes *sorted xs*
  assumes *mono f*
  shows *sorted* (*map f xs*)
  ⟨*proof*⟩

lemma *map-sort*:
  assumes *mono f*
  shows *sort* (*map f xs*) = *map f* (*sort xs*)
  ⟨*proof*⟩

lemma *median-cong*:
  assumes $\bigwedge i.\ i < n \Longrightarrow f\ i = g\ i$
  shows *median n f* = *median n g*
  ⟨*proof*⟩

lemma *median-restrict*:
  assumes $n > 0$
  shows *median n* ($\lambda i \in \{0..{<}n\}.f\ i$) = *median n f*
  ⟨*proof*⟩

lemma *median-rat*:
  assumes $n > 0$
  shows *real-of-rat* (*median n f*) = *median n* (λi. *real-of-rat* (*f i*))
⟨*proof*⟩

lemma *median-const*:
  assumes $k > 0$
  shows *median k* ($\lambda i \in \{0..{<}k\}.\ a$) = *a*
⟨*proof*⟩

end
theory *Set-Ext*
imports *Main*
begin

This is like *card-vimage-inj* but supports *inj-on* instead.

lemma *card-vimage-inj-on*:
  assumes *inj-on f B*
  assumes $A \subseteq f$ ‘ B
  shows *card* $(f$ −‘ $A \cap B) =$ *card* A
⟨*proof*⟩

lemma *card-ordered-pairs*:
  fixes M :: ($'a$ ::*linorder*) *set*
  assumes *finite M*
  shows $2 *$ *card* $\{(x,y) \in M \times M.\ x < y\} =$ *card* $M * ($*card* $M - 1)$
⟨*proof*⟩

end

10 Ranks, k smallest element and elements

theory *K-Smallest*
  imports *Main HOL−Library.Multiset List-Ext Multiset-Ext Set-Ext*
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* ⇒ $'a$ *set* ⇒ *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 $x \leq y \Longrightarrow$ *rank-of* x $S \leq$ *rank-of* y S
  ⟨*proof*⟩

lemma *rank-mono-commute*:
  assumes *finite S*
  assumes $S \subseteq T$
  assumes *strict-mono-on f T*
  assumes $x \in T$
  shows *rank-of* x $S =$ *rank-of* $(f\ x)$ $(f$ ‘ $S)$
⟨*proof*⟩

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

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

lemma *rank-strict-mono*:
  assumes *finite S*
  shows *strict-mono-on* $(\lambda x.$ *rank-of* x $S)$ S
⟨*proof*⟩

lemma *rank-of-image*:
  assumes *finite S*
  shows $(\lambda x.\ \textit{rank-of}\ x\ S)$ ‘ $S = \{0..<\textit{card}\ S\}$
  ⟨*proof*⟩

lemma *card-least*:
  assumes *finite S*
  shows $\textit{card}\ (\textit{least}\ k\ S) = \textit{min}\ k\ (\textit{card}\ S)$
⟨*proof*⟩

lemma *least-subset*: $\textit{least}\ k\ S \subseteq S$
  ⟨*proof*⟩

lemma *preserve-rank*:
  assumes *finite S*
  shows $\textit{rank-of}\ x\ (\textit{least}\ m\ S) = \textit{min}\ m\ (\textit{rank-of}\ x\ S)$
⟨*proof*⟩

lemma *rank-insert*:
  assumes *finite T*
  shows $\textit{rank-of}\ y\ (\textit{insert}\ v\ T) = \textit{of-bool}\ (v < y \land v \notin T) + \textit{rank-of}\ y\ T$
⟨*proof*⟩

lemma *least-mono-commute*:
  assumes *finite S*
  assumes *strict-mono-on f S*
  shows f ‘ $\textit{least}\ k\ S = \textit{least}\ k\ (f$ ‘ $S)$
⟨*proof*⟩

lemma *least-insert*:
  assumes *finite S*
  shows $\textit{least}\ k\ (\textit{insert}\ x\ (\textit{least}\ k\ S)) = \textit{least}\ k\ (\textit{insert}\ x\ S)$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

definition *count-le* **where** $\textit{count-le}\ x\ M = \textit{size}\ \{\#y \in\#\ M.\ y \le x\#\}$
definition *count-less* **where** $\textit{count-less}\ x\ M = \textit{size}\ \{\#y \in\#\ M.\ y < x\#\}$

definition *nth-mset* :: $\textit{nat} \Rightarrow ('a :: \textit{linorder})\ \textit{multiset} \Rightarrow 'a$ **where**
  $\textit{nth-mset}\ k\ M = \textit{sorted-list-of-multiset}\ M\ !\ k$

lemma *nth-mset-bound-left*:
  assumes $k < \textit{size}\ M$
  assumes $\textit{count-less}\ x\ M \le k$
  shows $x \le \textit{nth-mset}\ k\ M$
⟨*proof*⟩

lemma *nth-mset-bound-left-excl*:
  assumes $k < \textit{size}\ M$

assumes *count-le x M* $\leq$ *k*
shows *x* $<$ *nth-mset k M*
⟨*proof*⟩

lemma *nth-mset-bound-right*:
assumes *k* $<$ *size M*
assumes *count-le x M* $>$ *k*
shows *nth-mset k M* $\leq$ *x*
⟨*proof*⟩

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*⟩

lemma *nth-mset-max*:
assumes *size A* $>$ *k*
assumes $\bigwedge$*x*. *x* $\leq$ *nth-mset k A* $\Longrightarrow$ *count A x* $\leq$ *1*
shows *nth-mset k A* = *Max* (*least* (*k+1*) (*set-mset A*)) **and** *card* (*least* (*k+1*) (*set-mset A*)) = *k+1*
⟨*proof*⟩

end

11 Interpolation Polynomial Counts

theory *Interpolation-Polynomial-Counts*
imports *MainHOL−Algebra.Polynomial-Divisibility HOL−Algebra.Polynomials HOL−Library.FuncSet*
Set-Ext
begin

This section contains results about the count of polynomials with a given degree interpolating a certain number of points.

definition *bounded-degree-polynomials*
where *bounded-degree-polynomials F n* = {*x*. *x* $\in$ *carrier* (*poly-ring F*) $\wedge$ (*degree x* $<$ *n* $\vee$ *x* = [])}

lemma *bounded-degree-polynomials-length*:
bounded-degree-polynomials F n = {*x*. *x* $\in$ *carrier* (*poly-ring F*) $\wedge$ *length x* $\leq$ *n*}
⟨*proof*⟩

lemma *fin-degree-bounded*:
assumes *ring F*
assumes *finite* (*carrier F*)
shows *finite* (*bounded-degree-polynomials F n*)
⟨*proof*⟩

lemma *fin-fixed-degree*:
  assumes *ring F*
  assumes *finite* (*carrier F*)
  shows *finite* $\{p.\ p \in$ *carrier* (*poly-ring F*) $\wedge$ *length* $p = n\}$
⟨*proof*⟩

lemma *nonzero-length-polynomials-count*:
  assumes *ring F*
  assumes *finite* (*carrier F*)
  shows *card* $\{p.\ p \in$ *carrier* (*poly-ring F*) $\wedge$ *length* $p =$ *Suc n*$\}$
    $=$ (*card* (*carrier F*) $-$ *1*) $*$ *card* (*carrier F*) ^ *n*
⟨*proof*⟩

lemma *fixed-degree-polynomials-count*:
  assumes *ring F*
  assumes *finite* (*carrier F*)
  shows *card* ($\{p.\ p \in$ *carrier* (*poly-ring F*) $\wedge$ *length* $p = n\}$) $=$
   (*if* $n \geq$ *1 then* (*card* (*carrier F*) $-$ *1*) $*$ (*card* (*carrier F*) ^(*n*−*1*)) *else 1*)
⟨*proof*⟩

lemma *bounded-degree-polynomials-count*:
  assumes *ring F*
  assumes *finite* (*carrier F*)
  shows *card* (*bounded-degree-polynomials F n*) $=$ *card* (*carrier F*) ^ *n*
⟨*proof*⟩

lemma *non-empty-bounded-degree-polynomials*:
  assumes *ring F*
  shows *bounded-degree-polynomials F k* $\neq \{\}$
⟨*proof*⟩

11.1 Interpolation Polynomials

It is well known that over any field there is exactly one polynomial with degree at most $k - 1$ interpolating k points. That there is never more that one such polynomial follow from the fact that a polynomial of degree $k - 1$ cannot have more than $k - 1$ roots. This is already shown in HOL-Algebra in *field.size-roots-le-degree*. Existence is usually shown using Lagrange interpolation.

In the case of finite fields it is actually only necessary to show either that there is at most one such polynomial or at least one - because a function whose domain and co-domain has the same finite cardinality is injective if and only if it is surjective.

In the following a more generic result (over finite fields) is shown, counting the number of polynomials of degree $k + n - 1$ interpolating k points for non-negative n. As it turns out there are (*card* (*carrier F*))^n such polynomials. The trick is to observe that, for a given fix on the coefficients of order k to $k + n - 1$ and the values at k points there is at most one fitting polynomial.

An alternative way of stating the above result is that there is bijection between the polynomials of degree $n + k - 1$ and the product space $F^k \times F^n$ where the first component is the evaluation of the polynomials at k distinct points and the second component are the coefficients of order at least k.

definition *split-poly* **where** *split-poly F K p* =
  (*restrict* (*ring.eval F p*) *K*, λk. *ring.coeff F p* (*k*+*card K*))

The bijection *split-poly* returns the evaluation of the polynomial at the points in K and the coefficients of order at least *card K*.

In the following it is shown that its image is a subset of the product space mentioned above, and that *split-poly* is injective and finally that its image is exactly that product space using cardinalities.

lemma *split-poly-image*:
  assumes *field F*
  assumes $K \subseteq$ *carrier F*
  shows *split-poly F K* ‘ *bounded-degree-polynomials F* (*card K* + *n*) $\subseteq$
      ($K \rightarrow_E$ *carrier F*) $\times$ {*f*. *range f* $\subseteq$ *carrier F* $\wedge$ ($\forall k \geq n$. *f k* = $\mathbf{0}_F$)}
  ⟨*proof*⟩

lemma *poly-neg-coeff*:
  assumes *domain F*
  assumes $x \in$ *carrier* (*poly-ring F*)
  shows *ring.coeff F* ($\ominus_{poly\text{-}ring\ F}$ *x*) *k* = $\ominus_F$ *ring.coeff F x k*
⟨*proof*⟩

lemma *poly-substract-coeff*:
  assumes *domain F*
  assumes $x \in$ *carrier* (*poly-ring F*)
  assumes $y \in$ *carrier* (*poly-ring F*)
  shows *ring.coeff F* ($x \ominus_{poly\text{-}ring\ F} y$) *k* = *ring.coeff F x k* $\ominus_F$ *ring.coeff F y k*
  ⟨*proof*⟩

lemma *poly-substract-eval*:
  assumes *domain F*
  assumes $i \in$ *carrier F*
  assumes $x \in$ *carrier* (*poly-ring F*)
  assumes $y \in$ *carrier* (*poly-ring F*)
  shows *ring.eval F* ($x \ominus_{poly\text{-}ring\ F} y$) *i* = *ring.eval F x i* $\ominus_F$ *ring.eval F y i*
⟨*proof*⟩

lemma *poly-degree-bound-from-coeff*:
  assumes *ring F*
  assumes $x \in$ *carrier* (*poly-ring F*)
  assumes $\bigwedge k.\ k \geq n \Longrightarrow$ *ring.coeff F x k* = $\mathbf{0}_F$
  shows *degree x* $< n \vee x = \mathbf{0}_{poly\text{-}ring\ F}$

⟨*proof*⟩

lemma *max-roots*:
  assumes *field R*
  assumes $p \in$ *carrier* (*poly-ring R*)
  assumes $K \subseteq$ *carrier R*
  assumes *finite K*
  assumes *degree* $p <$ *card K*
  assumes $\bigwedge x.\ x \in K \Longrightarrow$ *ring.eval R p x* $= \mathbf{0}_R$
  shows $p = \mathbf{0}_{\textit{poly-ring } R}$
⟨*proof*⟩

lemma *split-poly-inj*:
  assumes *field F*
  assumes *finite K*
  assumes $K \subseteq$ *carrier F*
  shows *inj-on* (*split-poly F K*) (*carrier* (*poly-ring F*))
⟨*proof*⟩

lemma
  assumes *field F* $\wedge$ *finite* (*carrier F*)
  shows
    poly-count:*card* (*bounded-degree-polynomials F n*) = *card* (*carrier F*)^*n* (**is** *?A*)
and
    finite-poly-count: *finite* (*bounded-degree-polynomials F n*) (**is** *?B*)
⟨*proof*⟩

lemma
  assumes *finite* (*B* :: *'b set*)
  assumes $y \in B$
  shows
    card-mostly-constant-maps:
    card $\{f.\ \textit{range } f \subseteq B \wedge (\forall x.\ x \geq n \longrightarrow f\,x = y)\}$ = *card B* ^ *n* (**is** *card ?A* =
?B) **and**
    finite-mostly-constant-maps:
    finite $\{f.\ \textit{range } f \subseteq B \wedge (\forall x.\ x \geq n \longrightarrow f\,x = y)\}$
⟨*proof*⟩

lemma *split-poly-surj*:
  assumes *field F*
  assumes *finite* (*carrier F*)
  assumes $K \subseteq$ *carrier F*
  shows *split-poly F K* ‘ *bounded-degree-polynomials F* (*card K* + *n*) =
        $(K \rightarrow_E \textit{carrier } F) \times \{f.\ \textit{range } f \subseteq \textit{carrier } F \wedge (\forall k \geq n.\ f\,k = \mathbf{0}_F)\}$
      (**is** *split-poly F K* ‘ *?A* = *?B*)
⟨*proof*⟩

lemma *inv-subsetI*:
  assumes $\bigwedge x.\ x \in A \Longrightarrow f\,x \in B \Longrightarrow x \in C$

shows $f - ` B \cap A \subseteq C$
⟨*proof*⟩

lemma *interpolating-polynomials-count*:
assumes *field F*
assumes *finite* (*carrier F*)
assumes $K \subseteq$ *carrier F*
assumes f ‘ $K \subseteq$ *carrier F*
shows *card* $\{\omega \in$ *bounded-degree-polynomials F* (*card K* + n). ($\forall k \in K$. *ring.eval F* ω $k = f\ k$)$\}$ =
card (*carrier F*)^*n*
(**is** *card ?A* = *?B*)
⟨*proof*⟩

end

12 Indexed Products of Probability Mass Functions

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`

theory *Product-PMF-Ext*
imports *Main Probability-Ext HOL−Probability.Product-PMF*
begin

definition *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* $x \in$ *extensional I then* $\prod i \in I$. (*pmf* (*M i*)) (*x i*) *else 0*)
⟨*proof*⟩

lemma *set-prod-pmf*:
assumes *finite I*
shows *set-pmf* (*prod-pmf I M*) = *PiE I* (*set-pmf* $\circ$ *M*)
⟨*proof*⟩

lemma *set-pmf-iff'*: $x \notin$ *set-pmf M* $\longleftrightarrow$ *pmf M x* = *0*
⟨*proof*⟩

lemma *prob-prod-pmf*:
assumes *finite I*
shows *measure* (*measure-pmf* (*prod-pmf I M*)) (*Pi I A*) = ($\prod$ $i \in I$. *measure* (*M i*) (*A i*))
⟨*proof*⟩

lemma *prob-prod-pmf'*:
  assumes *finite I*
  assumes $J \subseteq I$
  shows *measure* (*measure-pmf* (*prod-pmf I M*)) (*Pi J A*) = ($\prod$ $i \in J$. *measure* (*M i*) (*A i*))
⟨*proof*⟩

lemma *prob-prod-pmf-slice*:
  assumes *finite I*
  assumes $i \in I$
  shows *measure* (*measure-pmf* (*prod-pmf I M*)) {ω. *P* (ω *i*)} = *measure* (*M i*) {ω. *P* ω}
  ⟨*proof*⟩

lemma *range-inter*: *range* ((∩) *F*) = *Pow F*
  ⟨*proof*⟩

On a finite set M the σ-Algebra generated by singletons and the empty set is already the power set of M.

lemma *sigma-sets-singletons-and-empty*:
  assumes *countable M*
  shows *sigma-sets M* (*insert* {} ((λ*k*. {*k*}) ‘ *M*)) = *Pow M*
⟨*proof*⟩

lemma *indep-vars-pmf*:
  assumes ⋀*a J*. $J \subseteq I$ ⟹ *finite J* ⟹
    $\mathcal{P}$(ω *in measure-pmf M*. ∀ $i \in J$. *X i* ω = *a i*) = ($\prod$ $i \in J$. $\mathcal{P}$(ω *in measure-pmf M*. *X i* ω = *a i*))
  shows *prob-space.indep-vars* (*measure-pmf M*) (λ*i*. *measure-pmf* (*M' i*)) *X I*
⟨*proof*⟩

lemma *indep-vars-restrict*:
  fixes *M* :: *'a* ⇒ *'b pmf*
  fixes *J* :: *'c set*
  assumes *disjoint-family-on f J*
  assumes $J \neq$ {}
  assumes ⋀*i*. $i \in J$ ⟹ *f i* ⊆ *I*
  assumes *finite I*
  shows *prob-space.indep-vars* (*measure-pmf* (*prod-pmf I M*)) (λ*i*. *measure-pmf* (*prod-pmf* (*f i*) *M*)) (λ*i* ω. *restrict* ω (*f i*)) *J*
⟨*proof*⟩

lemma *indep-vars-restrict-intro*:
  fixes *M* :: *'a* ⇒ *'b pmf*
  fixes *J* :: *'c set*
  assumes ⋀ω *i*. $i \in J$ ⟹ *X i* ω = *X i* (*restrict* ω (*f i*))
  assumes *disjoint-family-on f J*
  assumes $J \neq$ {}

assumes $\bigwedge i.\ i \in J \Longrightarrow f\ i \subseteq I$
assumes *finite I*
assumes $\bigwedge \omega\ i.\ i \in J \Longrightarrow\ \ X\ i\ \omega \in \mathit{space}\ (M'\ i)$
shows *prob-space.indep-vars* (*measure-pmf* (*prod-pmf I M*)) M' $(\lambda i\ \omega.\ X\ i\ \omega)$ J
⟨*proof*⟩

lemma *has-bochner-integral-prod-pmfI*:
fixes f :: $'a \Rightarrow {'b} \Rightarrow ({'c}$ :: {*second-countable-topology*,*banach*,*real-normed-field*})
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *has-bochner-integral* (*measure-pmf* (*M i*)) (*f i*) (*r i*)
shows *has-bochner-integral* (*prod-pmf I M*) $(\lambda x.\ (\prod i \in I.\ f\ i\ (x\ i)))$ $(\prod i \in I.\ r\ i)$
⟨*proof*⟩

lemma
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 *prod-pmf-integrable*: *integrable* (*prod-pmf I M*) $(\lambda x.\ (\prod i \in I.\ f\ i\ (x\ i)))$ (**is** *?A*) **and**
prod-pmf-integral: $\mathit{integral}^L$ (*prod-pmf I M*) $(\lambda x.\ (\prod i \in I.\ f\ i\ (x\ i))) =$
$(\prod\ i \in I.\ \mathit{integral}^L\ (M\ i)\ (f\ i))$ (**is** *?B*)
⟨*proof*⟩

lemma *has-bochner-integral-prod-pmf-sliceI*:
fixes f :: $'a \Rightarrow ({'b}$ :: {*second-countable-topology*,*banach*,*real-normed-field*})
assumes *finite I*
assumes $i \in I$
assumes *has-bochner-integral* (*measure-pmf* (*M i*)) (*f*) *r*
shows *has-bochner-integral* (*prod-pmf I M*) $(\lambda x.\ (f\ (x\ i)))$ *r*
⟨*proof*⟩

lemma
fixes f :: $'a \Rightarrow ({'b}$ :: {*second-countable-topology*,*banach*,*real-normed-field*})
assumes *finite I*
assumes $i \in I$
assumes *integrable* (*measure-pmf* (*M i*)) *f*
shows *integrable-prod-pmf-slice*: *integrable* (*prod-pmf I M*) $(\lambda x.\ (f\ (x\ i)))$ (**is** *?A*) **and**
integral-prod-pmf-slice: $\mathit{integral}^L$ (*prod-pmf I M*) $(\lambda x.\ (f\ (x\ i))) = \mathit{integral}^L\ (M\ i)\ f$ (**is** *?B*)
⟨*proof*⟩

lemma *variance-prod-pmf-slice*:
fixes f :: $'a \Rightarrow \mathit{real}$
assumes $i \in I$ *finite I*
assumes *integrable* (*measure-pmf* (*M i*)) $(\lambda\omega.\ f\ \omega\hat{\ }2)$
shows *prob-space.variance* (*prod-pmf I M*) $(\lambda\omega.\ f\ (\omega\ i))$ = *prob-space.variance*

$(M\ i)\ f$
⟨*proof*⟩

lemma *PiE-defaut-undefined-eq*: *PiE-dflt I undefined M* = *PiE I M*
  ⟨*proof*⟩

lemma *pmf-of-set-prod*:
  assumes *finite I*
  assumes $\bigwedge x.\ x \in I \Longrightarrow$ *finite* $(M\ x)$
  assumes $\bigwedge x.\ x \in I \Longrightarrow M\ x \neq \{\}$
  shows *pmf-of-set* (*PiE I M*) = *prod-pmf I* (λ*i*. *pmf-of-set* (*M i*))
  ⟨*proof*⟩

lemma *extensionality-iff*:
  assumes $f \in$ *extensional I*
  shows $((\lambda i \in I.\ g\ i) = f) = (\forall\, i \in I.\ g\ i = f\ i)$
  ⟨*proof*⟩

lemma *of-bool-prod*:
  assumes *finite I*
  shows *of-bool* $(\forall\, i \in I.\ P\ i) = (\prod i \in I.$ (*of-bool* $(P\ i)$ :: *'a* :: *field*))
  ⟨*proof*⟩

lemma *map-ptw*:
  fixes *I* :: *'a set*
  fixes *M* :: *'a* ⇒ *'b pmf*
  fixes *f* :: *'b* ⇒ *'c*
  assumes *finite I*
  shows *prod-pmf I M* ⤜ (λ*x*. *return-pmf* (λ*i* ∈ *I*. *f* (*x i*))) = *prod-pmf I* (λ*i*. (*M i* ⤜ (λ*x*. *return-pmf* (*f x*))))
⟨*proof*⟩

lemma *pair-pmfI*:
  A ⤜ (λ*a*. *B* ⤜ (λ*b*. *return-pmf* (*f a b*))) = *pair-pmf A B* ⤜ (λ(*a*,*b*). *return-pmf* (*f a b*))
  ⟨*proof*⟩

lemma *pmf-pair'*:
  pmf (*pair-pmf M N*) *x* = *pmf M* (*fst x*) ∗ *pmf N* (*snd x*)
  ⟨*proof*⟩

lemma *pair-pmf-ptw*:
  assumes *finite I*
  shows *pair-pmf* (*prod-pmf I A* :: ((*'i* ⇒ *'a*) *pmf*)) (*prod-pmf I B* :: ((*'i* ⇒ *'b*) *pmf*)) =
    prod-pmf I (λ*i*. *pair-pmf* (*A i*) (*B i*)) ⤜
      (λ*f*. *return-pmf* (*restrict* (*fst* ∘ *f*) *I*, *restrict* (*snd* ∘ *f*) *I*))

(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

13 Universal Hash Families

theory *Universal-Hash-Families*
imports *Main Interpolation-Polynomial-Counts Product-PMF-Ext*
begin

A k-universal hash family $\mathcal{H}$ is probability space, whose elements are hash functions with domain U and range $i.i < m$ such that:

- For every fixed $x \in U$ and value $y < m$ exactly $\frac{1}{m}$ of the hash functions map x to y: $P_{h \in \mathcal{H}}(h(x) = y) = \frac{1}{m}$.
- For at most k universe elements: $x_1, \cdots, x_m$ the functions $h(x_1), \cdots, h(x_m)$ are independent random variables.

In this section, we construct k-universal hash families following the approach outlined by Wegman and Carter using the polynomials of degree less than k over a finite field.

A hash function is just polynomial evaluation.

definition *hash* :: (*'a*, *'b*) *ring-scheme* ⇒ *'a* ⇒ *'a list* ⇒ *'a*
where *hash F x ω* = *ring.eval F ω x*

lemma *hash-range*:
assumes *ring F*
assumes *ω* ∈ *bounded-degree-polynomials F n*
assumes *x* ∈ *carrier F*
shows *hash F x ω* ∈ *carrier F*
⟨*proof*⟩

lemma *hash-range-2*:
assumes *ring F*
assumes *ω* ∈ *bounded-degree-polynomials F n*
shows (*λx. hash F x ω*) ‘ *carrier F* ⊆ *carrier F*
⟨*proof*⟩

lemma *poly-cards*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes *K* ⊆ *carrier F*
assumes *card K* ≤ *n*
assumes *y* ‘ *K* ⊆ (*carrier F*)
shows *card* {*ω* ∈ *bounded-degree-polynomials F n*. (∀ *k* ∈ *K*. *ring.eval F ω k* = *y k*)} =

card (*carrier F*)$\widehat{\ }$(*n*−*card K*)
⟨*proof*⟩

lemma *poly-cards-single*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes *k* ∈ *carrier F*
assumes *1* ≤ *n*
assumes *y* ∈ *carrier F*
shows *card* {ω ∈ *bounded-degree-polynomials F n*. *ring.eval F* ω *k* = *y*} =
card (*carrier F*)$\widehat{\ }$(*n*−*1*)
⟨*proof*⟩

lemma *expand-subset-filter*: {*x* ∈ *A*. *P x*} = *A* ∩ {*x*. *P x*}
⟨*proof*⟩

lemma *hash-prob*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes *K* ⊆ *carrier F*
assumes *card K* ≤ *n*
assumes *y* ‘ *K* ⊆ *carrier F*
shows 𝒫(ω *in pmf-of-set* (*bounded-degree-polynomials F n*). (∀ *x* ∈ *K*. *hash F x* ω = *y x*)) = *1*/(*real* (*card* (*carrier F*)))$\widehat{\ }$*card K*
⟨*proof*⟩

lemma *hash-prob-single*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes *x* ∈ *carrier F*
assumes *1* ≤ *n*
assumes *y* ∈ *carrier F*
shows 𝒫(ω *in pmf-of-set* (*bounded-degree-polynomials F n*). *hash F x* ω = *y*) = *1*/(*real* (*card* (*carrier F*)))
⟨*proof*⟩

lemma *hash-indep-pmf*:
assumes *field F* ∧ *finite* (*carrier F*)
assumes *J*⊆*carrier F*
assumes *finite J*
assumes *card J* ≤ *n*
assumes *1* ≤ *n*
shows *prob-space.indep-vars* (*pmf-of-set* (*bounded-degree-polynomials F n*))
(λ-. *pmf-of-set* (*carrier F*)) (*hash F*) *J*
⟨*proof*⟩

We introduce k-wise independent random variables using the existing definition of independent random variables.

definition (**in** *prob-space*) *k-wise-indep-vars* ::
nat ⇒ (*'b* ⇒ *'c measure*) ⇒ (*'b* ⇒ *'a* ⇒ *'c*) ⇒ *'b set* ⇒ *bool* **where**
k-wise-indep-vars k M' X' I = (∀ *J* ⊆ *I*. *card J* ≤ *k* ⟶ *finite J* ⟶ *indep-vars M' X' J*)

lemma *hash-k-wise-indep*:
  assumes *field F* ∧ *finite* (*carrier F*)
  assumes *1* ≤ *n*
   shows *prob-space.k-wise-indep-vars* (*pmf-of-set* (*bounded-degree-polynomials F n*)) *n*
    (λ-. *pmf-of-set* (*carrier F*)) (*hash F*) (*carrier F*)
  ⟨*proof*⟩

lemma *hash-inj-if-degree-1*:
  assumes *field F* ∧ *finite* (*carrier F*)
  assumes ω ∈ *bounded-degree-polynomials F n*
  assumes *degree* ω = *1*
  shows *inj-on* (λ*x*. *hash F x* ω) (*carrier F*)
⟨*proof*⟩

lemma (**in** *prob-space*) *k-wise-subset*:
  assumes *k-wise-indep-vars k M′ X′ I*
  assumes *J* ⊆ *I*
  shows *k-wise-indep-vars k M′ X′ J*
  ⟨*proof*⟩

end

14 Universal Hash Family for $\{0..<p\}$

Specialization of universal hash families from arbitrary finite fields to $\{0..<p\}$.

theory *Universal-Hash-Families-Nat*
  imports *Field Universal-Hash-Families Probability-Ext Encoding*
begin

lemma *fin-bounded-degree-polynomials*:
  assumes *p* > *0*
  shows *finite* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)
  ⟨*proof*⟩

lemma *ne-bounded-degree-polynomials*:
  shows *bounded-degree-polynomials* (*ZFact* (*int p*)) *n* ≠ {}
  ⟨*proof*⟩

lemma *card-bounded-degree-polynomials*:
  assumes *p* > *0*
  shows *card* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*) = *p*^*n*
  ⟨*proof*⟩

fun *hash* :: *nat* ⇒ *nat* ⇒ *int set list* ⇒ *nat*
  where *hash p x f* = *the-inv-into* {*0*..<*p*} (*zfact-embed p*) (*Universal-Hash-Families.hash*

(*ZFact p*) (*zfact-embed p x*) *f*)

declare *hash.simps* [*simp del*]

lemma *hash-range*:
  assumes *p* > *0*
  assumes *ω* ∈ *bounded-degree-polynomials* (*ZFact* (*int p*)) *n*
  assumes *x* < *p*
  shows *hash p x ω* < *p*
⟨*proof*⟩

lemma *hash-inj-if-degree-1*:
  assumes *prime p*
  assumes *ω* ∈ *bounded-degree-polynomials* (*ZFact* (*int p*)) *n*
  assumes *degree ω* = *1*
  shows *inj-on* (*λx. hash p x ω*) {*0..<p*}
⟨*proof*⟩

lemma *hash-prob*:
  assumes *prime p*
  assumes *K* ⊆ {*0..<p*}
  assumes *y* ' *K* ⊆ {*0..<p*}
  assumes *card K* ≤ *n*
  shows 𝒫(*ω in measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)).
    (∀ *x* ∈ *K. hash p x ω* = (*y x*))) = *1* / *real p*^*card K*
⟨*proof*⟩

lemma *hash-prob-2*:
  assumes *prime p*
  assumes *inj-on x K*
  assumes *x* ' *K* ⊆ {*0..<p*}
  assumes *y* ' *K* ⊆ {*0..<p*}
  assumes *card K* ≤ *n*
  shows 𝒫(*ω in measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)).
    (∀ *k* ∈ *K. hash p* (*x k*) *ω* = (*y k*))) = *1* / *real p*^*card K* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *hash-prob-range*:
  assumes *prime p*
  assumes *x* < *p*
  assumes *n* > *0*
  shows 𝒫(*ω in measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)).
    hash p x ω ∈ *A*) = *card* (*A* ∩ {*0..<p*}) / *p*
⟨*proof*⟩

lemma *hash-k-wise-indep*:

assumes *prime p*
assumes $1 \leq n$
shows *prob-space.k-wise-indep-vars* (*measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)))
n (λ-. *pmf-of-set* {*0..<p*}) (*hash p*) {*0..<p*}
⟨*proof*⟩

14.1 Encoding

fun $zfact_S$ **where** $zfact_S$ *p x* = (
if x ∈ *zfact-embed p* ‘ {*0..<p*} *then*
N_S (*the-inv-into* {*0..<p*} (*zfact-embed p*) *x*)
else
None
)

lemma *zfact-encoding* :
is-encoding ($zfact_S$ *p*)
⟨*proof*⟩

lemma *bounded-degree-polynomial-bit-count*:
assumes $p > 0$
assumes *x* ∈ *bounded-degree-polynomials* (*ZFact p*) *n*
shows *bit-count* ($list_S$ ($zfact_S$ *p*) *x*) ≤ *ereal* (*real n* ∗ (*2* ∗ *log 2 p* + *2*) + *1*)
⟨*proof*⟩

end

15 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".

The following lemma is an intentional copy of *sum-in-bigo* with order of assumptions reversed *)

lemma *sum-in-bigo-r*:
assumes *f2* ∈ *O*[*F'*](*g*)
assumes *f1* ∈ *O*[*F'*](*g*)
shows (λ*x*. *f1 x* + *f2 x*) ∈ *O*[*F'*](*g*)
⟨*proof*⟩

lemma *landau-sum*:
assumes *eventually* (λ*x*. *g1 x* ≥ (*0*::*real*)) *F'*
assumes *eventually* (λ*x*. *g2 x* ≥ *0*) *F'*
assumes *f1* ∈ *O*[*F'*](*g1*)

  assumes $f2 \in O[F'](g2)$
  shows $(\lambda x.\ f1\ x + f2\ x) \in O[F'](\lambda x.\ g1\ x + g2\ x)$
$\langle proof \rangle$

lemma *landau-sum-1*:
  assumes *eventually* $(\lambda x.\ g1\ x \geq (0{::}real))\ F'$
  assumes *eventually* $(\lambda x.\ g2\ x \geq 0)\ F'$
  assumes $f \in O[F'](g1)$
  shows $f \in O[F'](\lambda x.\ g1\ x + g2\ x)$
$\langle proof \rangle$

lemma *landau-sum-2*:
  assumes *eventually* $(\lambda x.\ g1\ x \geq (0{::}real))\ F'$
  assumes *eventually* $(\lambda x.\ g2\ x \geq 0)\ F'$
  assumes $f \in O[F'](g2)$
  shows $f \in O[F'](\lambda x.\ g1\ x + g2\ x)$
$\langle proof \rangle$

lemma *landau-ln-3*:
  assumes *eventually* $(\lambda x.\ (1{::}real) \leq f\ x)\ F'$
  assumes $f \in O[F'](g)$
  shows $(\lambda x.\ ln\ (f\ x)) \in O[F'](g)$
$\langle proof \rangle$

lemma *landau-ln-2*:
  assumes $a > (1{::}real)$
  assumes *eventually* $(\lambda x.\ 1 \leq f\ x)\ F'$
  assumes *eventually* $(\lambda x.\ a \leq g\ x)\ F'$
  assumes $f \in O[F'](g)$
  shows $(\lambda x.\ ln\ (f\ x)) \in O[F'](\lambda x.\ ln\ (g\ x))$
$\langle proof \rangle$

lemma *landau-real-nat*:
  fixes $f :: 'a \Rightarrow int$
  assumes $(\lambda x.\ of\text{-}int\ (f\ x)) \in O[F'](g)$
  shows $(\lambda x.\ real\ (nat\ (f\ x))) \in O[F'](g)$
$\langle proof \rangle$

lemma *landau-ceil*:
  assumes $(\lambda\text{-}.\ 1) \in O[F'](g)$
  assumes $f \in O[F'](g)$
  shows $(\lambda x.\ real\text{-}of\text{-}int\ \lceil f\ x \rceil) \in O[F'](g)$
  $\langle proof \rangle$

lemma *landau-nat-ceil*:
  assumes $(\lambda\text{-}.\ 1) \in O[F'](g)$
  assumes $f \in O[F'](g)$
  shows $(\lambda x.\ real\ (nat\ \lceil f\ x \rceil)) \in O[F'](g)$
  $\langle proof \rangle$

lemma *landau-const-inv*:
  assumes $c > (0::real)$
  assumes $(\lambda x.\ 1\ /\ f\ x) \in O[F'](g)$
  shows $(\lambda x.\ c\ /\ f\ x) \in O[F'](g)$
⟨*proof*⟩

lemma *eventually-nonneg-div*:
  assumes *eventually* $(\lambda x.\ (0::real) \leq f\ x)\ F'$
  assumes *eventually* $(\lambda x.\ 0 < g\ x)\ F'$
  shows *eventually* $(\lambda x.\ 0 \leq f\ x\ /\ g\ x)\ F'$
  ⟨*proof*⟩

lemma *eventually-nonneg-add*:
  assumes *eventually* $(\lambda x.\ (0::real) \leq f\ x)\ F'$
  assumes *eventually* $(\lambda x.\ 0 \leq g\ x)\ F'$
  shows *eventually* $(\lambda x.\ 0 \leq f\ x + g\ x)\ F'$
  ⟨*proof*⟩

lemma *eventually-ln-ge-iff*:
  assumes *eventually* $(\lambda x.\ (exp\ (c::real)) \leq f\ x)\ F'$
  shows *eventually* $(\lambda x.\ c \leq ln\ (f\ x))\ F'$
  ⟨*proof*⟩

lemma *div-commute*: $(a::real)\ /\ b = (1/b) * a$ ⟨*proof*⟩

lemma *eventually-prod1'*:
  assumes $B \neq bot$
  shows $(\forall_F\ x\ in\ A \times_F B.\ P\ (fst\ x)) \longleftrightarrow (\forall_F\ x\ in\ A.\ P\ x)$
  ⟨*proof*⟩

lemma *eventually-prod2'*:
  assumes $A \neq bot$
  shows $(\forall_F\ x\ in\ A \times_F B.\ P\ (snd\ x)) \longleftrightarrow (\forall_F\ x\ in\ B.\ P\ x)$
  ⟨*proof*⟩

instantiation *rat* :: *linorder-topology*
begin

definition *open-rat* :: *rat set* ⇒ *bool*
  where *open-rat* = *generate-topology* (*range* $(\lambda a.\ \{..< a\}) \cup$ *range* $(\lambda a.\ \{a <..\})$)

instance
  ⟨*proof*⟩
end

lemma *inv-at-right-0-inf*:
  $\forall_F\ x\ in\ at\text{-}right\ 0.\ c \leq 1\ /\ real\text{-}of\text{-}rat\ x$
  ⟨*proof*⟩

end

16 Frequency Moment 0

theory *Frequency-Moment-0*
 imports *Main Primes-Ext Float-Ext Median K-Smallest Universal-Hash-Families-Nat Encoding*
  Frequency-Moments Landau-Ext
begin

This section contains a formalization of the algorithm for the zero-th frequency moment. It is a KMV algorithm with a rounding method to match the space complexity of the best algorithm described in [2].

In addition ot the Isabelle proof here, there is also and informal hand-writtend proof in Appendix A.

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

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 = find-prime-above (max n 19);
      let r = nat (4 ∗ ⌈log 2 (1 / real-of-rat δ)⌉ + 24);
      h ← prod-pmf {0..<s} (λ-. pmf-of-set (bounded-degree-polynomials (ZFact
(int 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 ∈ {0..<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 ∈ {0..<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)))
    ))

definition f0-sketch where
  f0-sketch p r t h xs = least t ((λx. float-of (truncate-down r (hash p x h))) ‘ (set
xs))

lemma f0-alg-sketch:
  fixes n :: nat
  fixes as :: nat list
  assumes ε ∈ {0<..<1}
  assumes δ ∈ {0<..<1}
  defines sketch ≡ fold (λa state. state ≫= f0-update a) as (f0-init δ ε n)
  defines t ≡ nat ⌈80 / (real-of-rat δ)^2⌉
  defines s ≡ nat ⌈−(18 ∗ ln (real-of-rat ε))⌉
  defines p ≡ find-prime-above (max n 19)
  defines r ≡ nat (4 ∗ ⌈log 2 (1 / real-of-rat δ)⌉ + 24)
  shows sketch = map-pmf (λx. (s,t,p,r, x, λi ∈ {0..<s}. f0-sketch p r t (x i) as))
    (prod-pmf {0..<s} (λ-. pmf-of-set (bounded-degree-polynomials (ZFact (int p))
2)))
⟨proof⟩


lemma abs-ge-iff: ((x::real) ≤ abs y) = (x ≤ y ∨ x ≤ −y)
  ⟨proof⟩

lemma two-powr-0: 2 powr (0::real) = 1
  ⟨proof⟩

lemma count-nat-abs-diff-2:
  fixes x :: nat
  fixes q :: real
  assumes q ≥ 0
  defines A ≡ {(k::nat). abs (real x − real k) ≤ q ∧ k ≠ x}
  shows real (card A) ≤ 2 ∗ q and finite A
⟨proof⟩

lemma f0-collision-prob:
  fixes p :: nat
  assumes Factorial-Ring.prime p
  defines Ω ≡ pmf-of-set (bounded-degree-polynomials (ZFact (int p)) 2)
  assumes M ⊆ {0..<p}
  assumes c ≥ 1
  assumes r ≥ 1
  shows 𝒫(ω in measure-pmf Ω.
    ∃x ∈ M. ∃y ∈ M.
    x ≠ y ∧
    truncate-down r (hash p x ω) ≤ c ∧
    truncate-down r (hash p x ω) = truncate-down r (hash p y ω)) ≤
    6 ∗ (real (card M))^2 ∗ c^2 ∗ 2 powr −r / (real p) ^2 + 1/real p (is 𝒫(ω in -. ?l
ω) ≤ ?r1 + ?r2)
⟨proof⟩

lemma inters-compr: A ∩ {x. P x} = {x ∈ A. P x}
  ⟨proof⟩

lemma of-bool-square: (of-bool x)^2 = ((of-bool x)::real)
```

$\langle proof \rangle$

theorem *f0-alg-correct*:
  assumes $\varepsilon \in \{0<..<1\}$
  assumes $\delta \in \{0<..<1\}$
  assumes *set as* $\subseteq \{0..<n\}$
  defines $M \equiv$ *fold* (λa *state. state* $\gg=$ *f0-update a*) *as* (*f0-init* δ ε *n*) $\gg=$ *f0-result*
  shows $\mathcal{P}(\omega$ *in measure-pmf M*. $|\omega - F\ 0\ as| \le \delta * F\ 0\ as) \ge 1 -$ *of-rat* ε
$\langle proof \rangle$

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)$\rceil$ + 24) *in*
    let t = nat $\lceil 80$ / (*real-of-rat* $\delta)^2$ $\rceil$ *in*
    8 +
    2 ∗ *log 2* (*real s* + 1) +
    2 ∗ *log 2* (*real t* + 1) +
    2 ∗ *log 2* (*real n* + 10) +
    2 ∗ *log 2* (*real r* + 1) +
    real s ∗ (12 + 4 ∗ *log 2* (10 + *real n*) +
    real t ∗ (11 + 4 ∗ *r* + 2 ∗ *log 2* (*log 2* (*real n* + 9)))))

definition *encode-state* :: *f0-state* ⇒ *bool list option* **where**
  encode-state =
    $N_S \times_D (\lambda s.$
    $N_S \times_S ($
    $N_S \times_D (\lambda p.$
    $N_S \times_S ($
    $([0..<s] \rightarrow_S (list_S\ (zfact_S\ p))) \times_S$
    $([0..<s] \rightarrow_S (set_S\ F_S))))))$

lemma *inj-on encode-state* (*dom encode-state*)
  $\langle proof \rangle$

lemma *f-subset*:
  assumes *g* ‘ *A* ⊆ *h* ‘ *B*
  shows (λx. *f* (*g x*)) ‘ *A* ⊆ (λx. *f* (*h x*)) ‘ *B*
  $\langle proof \rangle$

theorem *f0-exact-space-usage*:
  assumes $\varepsilon \in \{0<..<1\}$
  assumes $\delta \in \{0<..<1\}$
  assumes *set as* $\subseteq \{0..<n\}$
  defines $M \equiv$ *fold* (λa *state. state* $\gg=$ *f0-update a*) *as* (*f0-init* δ ε *n*)
  shows *AE* ω *in M*. *bit-count* (*encode-state* ω) ≤ *f0-space-usage* (*n*, ε, δ)
$\langle proof \rangle$

lemma *f0-asympotic-space-complexity*:

f0-space-usage $\in$ *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** - $\in$ *O*[*?F*](*?rhs*))
⟨*proof*⟩

end

17 Partitions

theory *Partitions*
  imports *Main HOL−Library.Multiset HOL.Real List-Ext*
begin

This section introduces a function that enumerates all the partitions of {*0..<n*}. The partitions are represented as lists with *n* elements. If the element at index *i* and *j* have the same value, then *i* and *j* are in the same partition.

fun *enum-partitions-aux* :: *nat* $\Rightarrow$ (*nat* $\times$ *nat list*) *list*
  where
    enum-partitions-aux 0 = [(*0*, [])] |
    enum-partitions-aux (*Suc n*) =
      [(*c*+*1*, *c*#*x*). (*c*,*x*) ← *enum-partitions-aux n*]@
      [(*c*, *y*#*x*). (*c*,*x*) ← *enum-partitions-aux n*, *y* ← [*0*..<*c*]]

fun *enum-partitions* **where** *enum-partitions n* = *map snd* (*enum-partitions-aux n*)

definition *has-eq-relation* :: *nat list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool* **where**
  has-eq-relation r xs = (*length xs* = *length r* $\wedge$ ($\forall$ *i* < *length xs*. $\forall$ *j* < *length xs*. (*xs* ! *i* = *xs* ! *j*) = (*r* ! *i* = *r* ! *j*)))

lemma *filter-one-elim*:
  length (*filter p xs*) = *1* $\Longrightarrow$ ($\exists$ *u v w*. *xs* = *u*@*v*#*w* $\wedge$ *p v* $\wedge$ *length* (*filter p u*) = *0* $\wedge$ *length* (*filter p w*) = *0*)
  (**is** *?A xs* $\Longrightarrow$ *?B xs*)
⟨*proof*⟩

lemma *has-eq-elim*:
  has-eq-relation (*r*#*rs*) (*x*#*xs*) = (
    ($\forall$ *i* < *length xs*. (*r* = *rs* ! *i*) = (*x* = *xs* ! *i*)) $\wedge$
    has-eq-relation rs xs)
⟨*proof*⟩

lemma *enum-partitions-aux-range*:
  x $\in$ *set* (*enum-partitions-aux n*) $\Longrightarrow$ *set* (*snd x*) = {*k*. *k* < *fst x*}
  ⟨*proof*⟩

lemma *enum-partitions-aux-len*:
  $x \in set\ (enum\text{-}partitions\text{-}aux\ n) \Longrightarrow length\ (snd\ x) = n$
  ⟨*proof*⟩

lemma *enum-partitions-complete-aux*: $k < n \Longrightarrow length\ (filter\ (\lambda x.\ x = k)\ [0..<n]) = Suc\ 0$
  ⟨*proof*⟩

lemma *enum-partitions-complete*:
  $length\ (filter\ (\lambda p.\ has\text{-}eq\text{-}relation\ p\ x)\ (enum\text{-}partitions\ (length\ x))) = 1$
⟨*proof*⟩

fun *verify* **where**
  $verify\ r\ x\ 0\ \text{-} = True$ |
  $verify\ r\ x\ (Suc\ n)\ 0 = verify\ r\ x\ n\ n$ |
  $verify\ r\ x\ (Suc\ n)\ (Suc\ m) = (((r\ !\ n = r\ !\ m) = (x\ !\ n = x\ !\ m)) \wedge (verify\ r\ x\ (Suc\ n)\ m))$

lemma *verify-elim-1*:
  $verify\ r\ x\ (Suc\ n)\ m = (verify\ r\ x\ n\ n \wedge\ (\forall i < m.\ (r\ !\ n = r\ !\ i) = (x\ !\ n = x\ !\ i)))$
  ⟨*proof*⟩

lemma *verify-elim*:
  $verify\ r\ x\ m\ m = (\forall i < m.\ \forall j < i.\ (r\ !\ i = r\ !\ j) = (x\ !\ i = x\ !\ j))$
  ⟨*proof*⟩

lemma *has-eq-relation-elim*:
  $has\text{-}eq\text{-}relation\ r\ xs = (length\ r = length\ xs \wedge verify\ r\ xs\ (length\ xs)\ (length\ xs))$
  ⟨*proof*⟩

lemma *sum-filter*: $sum\text{-}list\ (map\ (\lambda p.\ if\ f\ p\ then\ (r{::}real)\ else\ 0)\ y) = r*(length\ (filter\ f\ y))$
  ⟨*proof*⟩

lemma *sum-partitions*: $sum\text{-}list\ (map\ (\lambda p.\ if\ has\text{-}eq\text{-}relation\ p\ x\ then\ (r{::}real)\ else\ 0)\ (enum\text{-}partitions\ (length\ x))) = r$
  ⟨*proof*⟩

lemma *sum-partitions′*:
  assumes $n = length\ x$
  shows $sum\text{-}list\ (map\ (\lambda p.\ of\text{-}bool\ (has\text{-}eq\text{-}relation\ p\ x) * (r{::}real))\ (enum\text{-}partitions\ n)) = r$
  ⟨*proof*⟩

lemma *eq-rel-obtain-bij*:
  assumes $has\text{-}eq\text{-}relation\ u\ v$
  obtains f **where** $bij\text{-}betw\ f\ (set\ u)\ (set\ v)\ \bigwedge y.\ y \in set\ u \Longrightarrow count\text{-}list\ u\ y = count\text{-}list\ v\ (f\ y)$

⟨*proof*⟩

end

18 Frequency Moment 2

theory *Frequency-Moment-2*
  imports *Main Median Partitions Primes-Ext Encoding List-Ext*
    Universal-Hash-Families-Nat Frequency-Moments Landau-Ext
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 hash p k h ∈ {k. 2*k < p} then int p − 1 else − int p − 1)

type-synonym f2-state = nat × nat × nat × (nat × nat ⇒ int set 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 / $\delta^2$⌉;
      let $s_2$ = nat ⌈−(18 ∗ ln (real-of-rat ε))⌉;
      let p = find-prime-above (max n 3);
    h ← prod-pmf ({0..<$s_1$} × {0..<$s_2$}) (λ-. pmf-of-set (bounded-degree-polynomials
(ZFact (int p)) 4));
      return-pmf ($s_1$, $s_2$, p, h, (λ- ∈ {0..<$s_1$} × {0..<$s_2$}. (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, λi ∈ {0..<$s_1$} × {0..<$s_2$}. 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 $s_2$ (λ$i_2$ ∈ {0..<$s_2$}.
        (∑ $i_1$∈{0..<$s_1$} . (rat-of-int (sketch ($i_1$, $i_2$)))$^2$) / (((rat-of-nat p)$^2$−1) ∗
rat-of-nat $s_1$)))

lemma f2-hash-exp:
  assumes Factorial-Ring.prime p
  assumes k < p
  assumes p > 2
  shows
```

  prob-space.expectation (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *4*))
  ($\lambda\omega$. *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*⟩

lemma
 assumes *Factorial-Ring.prime p*
 assumes $p > 2$
 assumes $\bigwedge a.\ a \in set\ as \Longrightarrow a < p$
 defines $M \equiv$ *measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *4*))
 defines $f \equiv$ ($\lambda\omega$. *real-of-int* (*sum-list* (*map* (*f2-hash p* ω) *as*))^*2*)
 shows *var-f2*:*prob-space.variance M f* ≤ *2*∗(*real-of-rat* (*F 2 as*)^*2*) ∗ ((*real p*)^2−*1*)^2 (**is** *?A*)
 and *exp-f2*:*prob-space.expectation M f* = *real-of-rat* (*F 2 as*) ∗ ((*real p*)^2−*1*) (**is** *?B*)
⟨*proof*⟩

lemma *f2-alg-sketch*:
 fixes *n* :: *nat*
 fixes *as* :: *nat list*
 assumes $\varepsilon \in \{0 <..< 1\}$
 assumes $\delta > 0$
 defines $s_1 \equiv nat\ \lceil 6\ /\ \delta^2 \rceil$
 defines $s_2 \equiv nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)) \rceil$
 defines $p \equiv$ *find-prime-above* (*max n 3*)
 defines *sketch* ≡ *fold* (λ*a state*. *state* ⤜ *f2-update a*) *as* (*f2-init* δ ε *n*)
 defines $\Omega \equiv$ *prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$) ($\lambda$-. *pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *4*))
 shows *sketch* = Ω ⤜ (λh. *return-pmf* (s_1, s_2, *p*, *h*,
   $\lambda i \in \{0..<s_1\} \times \{0..<s_2\}$. *sum-list* (*map* (*f2-hash p* (*h i*)) *as*)))
⟨*proof*⟩

theorem *f2-alg-correct*:
 assumes $\varepsilon \in \{0 <..< 1\}$
 assumes $\delta > 0$
 assumes *set as* ⊆ $\{0..<n\}$
 defines *M* ≡ *fold* (λ*a state*. *state* ⤜ *f2-update a*) *as* (*f2-init* δ ε *n*) ⤜ *f2-result*
 shows $\mathcal{P}$(ω *in measure-pmf M*. $|\omega - F\ 2\ as| \le \delta * F\ 2\ as$) ≥ *1*−*of-rat* ε
⟨*proof*⟩

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

$2 * log\ 2\ (s_2 + 1) +$
$2 * log\ 2\ (4 + 2 * real\ n) +$
$s_1 * s_2 * (13 + 8 * log\ 2\ (4 + 2 * real\ n) + 2 * log\ 2\ (real\ m * (4 + 2 * real\ n) + 1)))$

definition *encode-state* :: *f2-state* ⇒ *bool list option* **where**
encode-state =
$N_S \times_D (\lambda s_1.$
$N_S \times_D (\lambda s_2.$
$N_S \times_D (\lambda p.$
$(List.product\ [0..<s_1]\ [0..<s_2] \rightarrow_S (list_S\ (zfact_S\ p))) \times_S$
$(List.product\ [0..<s_1]\ [0..<s_2] \rightarrow_S I_S))))$

lemma *inj-on encode-state* (*dom encode-state*)
⟨*proof*⟩

theorem *f2-exact-space-usage*:
assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq \{0..<n\}$
defines $M \equiv fold\ (\lambda a\ state.\ state \gg= f2\text{-}update\ a)\ as\ (f2\text{-}init\ \delta\ \varepsilon\ n)$
shows *AE* ω *in* M. *bit-count* (*encode-state* ω) $\le$ *f2-space-usage* (n, *length as*, ε, δ)
⟨*proof*⟩

theorem *f2-asympotic-space-complexity*:
f2-space-usage $\in O[at\text{-}top \times_F at\text{-}top \times_F at\text{-}right\ 0 \times_F at\text{-}right\ 0](\lambda\ (n, m, \varepsilon, \delta).$

$(ln\ (1\ /\ of\text{-}rat\ \varepsilon))\ /\ (of\text{-}rat\ \delta)^2 * (ln\ (real\ n) + ln\ (real\ m)))$
(**is** - $\in O[?F](?rhs)$)
⟨*proof*⟩

end

19 Frequency Moment k

theory *Frequency-Moment-k*
imports *Main Median Product-PMF-Ext Lp.Lp List-Ext Encoding Frequency-Moments Landau-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 $k\ \delta\ \varepsilon\ n$ =
do {
let $s_1 = nat\ \lceil 3{*}real\ k{*}(real\ n)\ powr\ (1{-}1/\ real\ k)/\ (real\text{-}of\text{-}rat\ \delta)^2\rceil$;

```
    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))
  )

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

}

lemma *bernoulli-pmf-1*: *bernoulli-pmf 1 = return-pmf True*
⟨*proof*⟩

lemma *split-space*:
$(\sum a{\in}\{(u, v).\ v < \textit{count-list as } u\}.\ (f\ (\textit{snd } a))) =$
$(\sum u \in \textit{set as}.\ (\sum v \in\{0..<\textit{count-list as } u\}.\ (f\ v)))$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma
assumes $as \neq []$
shows *fin-space*: *finite* $\{(u, v).\ v < \textit{count-list as } u\}$ **and**
non-empty-space: $\{(u, v).\ v < \textit{count-list as } u\} \neq \{\}$ **and**
card-space: *card* $\{(u, v).\ v < \textit{count-list as } u\} = \textit{length as}$
⟨*proof*⟩

lemma *fk-alg-aux-5*:
assumes $as \neq []$
shows *pmf-of-set* $\{k.\ k < \textit{length as}\} \gg\!= (\lambda k.\ \textit{return-pmf}\ (as\ !\ k, \textit{count-list}\ (\textit{drop}\ (k{+}1)\ as)\ (as\ !\ k)))$
$= \textit{pmf-of-set}\ \{(u,v).\ v < \textit{count-list as } u\}$
⟨*proof*⟩

lemma *fk-alg-aux-4*:
assumes $as \neq []$
shows *fold* $(\lambda x\ (c,\textit{state}).\ (c{+}1, \textit{state} \gg\!= \textit{fk-update}''\ x\ c))\ as\ (0, \textit{return-pmf}\ (0,0)) =$
$(\textit{length as}, \textit{pmf-of-set}\ \{k.\ k < \textit{length as}\} \gg\!= (\lambda k.\ \textit{return-pmf}\ (as\ !\ k, \textit{count-list}\ (\textit{drop}\ (k{+}1)\ as)\ (as\ !\ k))))$
⟨*proof*⟩

definition *if-then-else* **where** *if-then-else p q r* = (*if p then q else r*)

This definition is introduced to be able to temporarily substitute *if p then q else r* with *if-then-else p q r*, which unblocks the simplifier to process *q* and *r*.

lemma *fk-alg-aux-2*:
fold $(\lambda x\ (c, \textit{state}).\ (c{+}1, \textit{state} \gg\!= \textit{fk-update}'\ x\ s_1\ s_2\ c))\ as\ (0, \textit{return-pmf}\ (\lambda i \in \{0..<s_1\} \times \{0..<s_2\}.\ (0,0)))$
$= (\textit{length as}, \textit{prod-pmf}\ (\{0..<s_1\} \times \{0..<s_2\})\ (\lambda\text{-}.\ (\textit{snd}\ (\textit{fold}\ (\lambda x\ (c,\textit{state}).\ (c{+}1, \textit{state} \gg\!= \textit{fk-update}''\ x\ c))\ as\ (0, \textit{return-pmf}\ (0,0))))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *fk-alg-aux-1*:
fixes k :: *nat*
fixes ε :: *rat*
assumes $\delta > 0$

assumes *set as* $\subseteq$ $\{0..<n\}$
assumes *as* $\neq$ []
defines *sketch* $\equiv$ *fold* (λa *state. state* $\gg=$ *fk-update a*) *as* (*fk-init k* δ ε *n*)
defines s_1 $\equiv$ *nat* $\lceil 3*real\ k*(real\ n)\ powr\ (1-1/\ real\ k)/\ (real\text{-}of\text{-}rat\ \delta)^2\rceil$
defines s_2 $\equiv$ *nat* $\lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon))\rceil$
shows *sketch* =
map-pmf ($\lambda x.$ (s_1,s_2,*k*,*length as*, *x*))
(*snd* (*fold* (λx (*c*, *state*). (*c*+*1*, *state* $\gg=$ *fk-update'* *x* s_1 s_2 *c*)) *as* (*0*, *return-pmf* ($\lambda i \in \{0..<s_1\} \times \{0..<s_2\}$. (*0*,*0*)))))
⟨*proof*⟩

lemma *power-diff-sum*:
assumes $k > 0$
shows (a :: $'a$ :: {*comm-ring-1*,*power*})^*k* −*b*^*k* = ($a-b$) ∗ *sum* ($\lambda i.$ *a*^*i* ∗ *b*^($k-1-i$)) $\{0..<k\}$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *power-diff-est*:
assumes $k > 0$
assumes (a :: *real*) $\geq b$
assumes $b \geq 0$
shows *a*^*k* −*b*^*k* $\leq$ ($a-b$) ∗ *k* ∗ *a*^($k-1$)
⟨*proof*⟩

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* ($\lambda x.$ *f x* ∗ *g x*) *A*| $\leq$ (*sum* ($\lambda x.$ |*f x*| *powr p*) *A*) *powr* ($1/p$) ∗ (*sum* ($\lambda x.$ |*g x*| *powr q*) *A*) *powr* ($1/q$)
⟨*proof*⟩

lemma *fk-estimate*:
assumes *as* $\neq$ []
assumes *set as* $\subseteq$ $\{0..<n\}$
assumes $k \geq 1$
shows *real* (*length as*) ∗ *real-of-rat* (*F* ($2*k-1$) *as*) $\leq$ *real n powr* ($1 - 1$ / *real k*) ∗ (*real-of-rat* (*F k as*))^*2*
(**is** *?lhs* $\leq$ *?rhs*)
⟨*proof*⟩

lemma *fk-alg-core-exp*:
assumes *as* $\neq$ []
assumes $k \geq 1$
shows *has-bochner-integral* (*measure-pmf* (*pmf-of-set* {(*u*, *v*). *v* < *count-list as u*}))
($\lambda a.$ *real* (*length as*) ∗ *real* (*Suc* (*snd a*) ^ *k* − *snd a* ^ *k*)) (*real-of-rat* (*F k as*))
⟨*proof*⟩

lemma *fk-alg-core-var*:
  assumes $as \neq []$
  assumes $k \geq 1$
  assumes $set\ as \subseteq \{0..{<}n\}$
  shows $prob\text{-}space.variance\ (measure\text{-}pmf\ (pmf\text{-}of\text{-}set\ \{(u,\ v).\ v < count\text{-}list\ as\ u\}))$
    $(\lambda a.\ real\ (length\ as) * real\ (Suc\ (snd\ a)\ \hat{}\ k - snd\ a\ \hat{}\ k))$
    $\leq (real\text{-}of\text{-}rat\ (F\ k\ as))^2 * real\ k * real\ n\ powr\ (1 - 1\ /\ real\ k)$
$\langle proof \rangle$

theorem *fk-alg-sketch*:
  fixes $\varepsilon :: rat$
  assumes $k \geq 1$
  assumes $\delta > 0$
  assumes $set\ as \subseteq \{0..{<}n\}$
  assumes $as \neq []$
  defines $sketch \equiv fold\ (\lambda a\ state.\ state \gg\!= fk\text{-}update\ a)\ as\ (fk\text{-}init\ k\ \delta\ \varepsilon\ n)$
  defines $s_1 \equiv nat\ \lceil 3{*}real\ k{*}(real\ n)\ powr\ (1{-}1/\ real\ k)/\ (real\text{-}of\text{-}rat\ \delta)^2 \rceil$
  defines $s_2 \equiv nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)) \rceil$
  shows $sketch = map\text{-}pmf\ (\lambda x.\ (s_1,s_2,k,length\ as,\ x))$
    $(prod\text{-}pmf\ (\{0..{<}s_1\} \times \{0..{<}s_2\})\ (\lambda\text{-}.\ pmf\text{-}of\text{-}set\ \{(u,v).\ v < count\text{-}list\ as\ u\}))$
  $\langle proof \rangle$

lemma *fk-alg-correct*:
  assumes $k \geq 1$
  assumes $\varepsilon \in \{0{<}..{<}1\}$
  assumes $\delta > 0$
  assumes $set\ as \subseteq \{0..{<}n\}$
  defines $M \equiv fold\ (\lambda a\ state.\ state \gg\!= fk\text{-}update\ a)\ as\ (fk\text{-}init\ k\ \delta\ \varepsilon\ n) \gg\!= fk\text{-}result$
  shows $\mathcal{P}(\omega\ in\ measure\text{-}pmf\ M.\ |\omega - F\ k\ as| \leq \delta * F\ k\ as) \geq 1 - of\text{-}rat\ \varepsilon$
$\langle proof \rangle$

fun *fk-space-usage* $:: (nat \times nat \times nat \times rat \times rat) \Rightarrow real$ **where**
  $fk\text{-}space\text{-}usage\ (k,\ n,\ m,\ \varepsilon,\ \delta) = ($
    $let\ s_1 = nat\ \lceil 3{*}real\ k{*}(real\ n)\ powr\ (1{-}1/\ real\ k)\ /\ (real\text{-}of\text{-}rat\ \delta)^2\ \rceil\ in$
    $let\ s_2 = nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)) \rceil\ in$
    $5\ +$
    $2 * log\ 2\ (s_1 + 1)\ +$
    $2 * log\ 2\ (s_2 + 1)\ +$
    $2 * log\ 2\ (real\ k + 1)\ +$
    $2 * log\ 2\ (real\ m + 1)\ +$
    $s_1 * s_2 * (3 + 2 * log\ 2\ (real\ n{+}1) + 2 * log\ 2\ (real\ m{+}1)))$

definition *encode-state* $:: fk\text{-}state \Rightarrow bool\ list\ option$ **where**
  $encode\text{-}state =$
    $N_S \times_D (\lambda s_1.$
    $N_S \times_D (\lambda s_2.$
    $N_S \times_S$

$N_S \times_S$
$(List.product\ [0..{<}s_1]\ [0..{<}s_2] \rightarrow_S (N_S \times_S N_S))))$

lemma *inj-on encode-state* (*dom encode-state*)
⟨*proof*⟩

theorem *fk-exact-space-usage*:
assumes $k \geq 1$
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\delta > 0$
assumes *set as* $\subseteq \{0..{<}n\}$
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-state* ω) $\leq$ *fk-space-usage* (*k*, *n*, *length as*, ε, δ) (**is** *AE* ω *in M*. (- $\leq$ *?rhs*))
⟨*proof*⟩

lemma *fk-asympotic-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*⟩

end

A Informal proof of correctness for the F_0 algorithm

This section contains a detailed informal proof for the correctness of the F_0-algorithm. Because of the standard amplification result about medians (see for example [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:

$$\begin{aligned} t &:= \lceil 80\delta^{-2} \rceil \\ r &:= 4 \log_2 \lceil \delta^{-1} \rceil + 24 \end{aligned}$$

The estimate the algorithm obtains is:

$$A := \{a_1, \ldots, a_m\} \qquad\qquad H := \{\lfloor h(a) \rfloor_r | a \in A\}$$

$$R := \begin{cases} tp\,(\min_t(H))^{-1} & \text{if } |H| \geq t \\ |H| & \text{othewise,} \end{cases}$$

Here $\min_t(H)$ denotes the t-th smallest element of H. With these definitions, it is possible to state the goal as:

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

which is shown by 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:

$$\begin{aligned} H^* &:= \{h(a)|a \in A\}^{\#} \\ R^* &:= tp\left(\mathrm{rank}_t^{\#}(H^*)\right)^{-1} \end{aligned}$$

These definitions correspond to the H, R but with a few minor modifications. 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:

$$\delta' := \frac{3}{4}\delta \tag{1}$$

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

$$P\left(|R^* - F_0| \leq \delta' F_0 \wedge |R - R^*| > \frac{\delta}{4} F_0\right) \leq \frac{1}{9} \tag{3}$$

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:

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

For the verification of Equation 2 let us introduce:

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

and observe that $\min_t^{\#}(H^*) < u$ if $Q(u) \geq t$ and $\min_t^{\#}(H^*) \geq v$ if $Q(v) \leq t - 1$. 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 rank t 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 $Q(u) = \sum_{a \in A} 1_{\{h(a)<u\}}$[1], i.e., Q is a sum of pairwise independent $\{0,1\}$-valued random variables, with expectation $\frac{u}{p}$ and variance $\frac{u}{p} - \frac{u^2}{p^2}$. [2] Using lineariy of expectation and Bienaymé's identity, it follows that $\operatorname{Var} Q(u) \leq \operatorname{E} Q(u) = |A|up^{-1} = F_0 up^{-1}$ for $u \in \{0, \ldots, p\}$.

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

$$\begin{aligned} t - 1 \quad &\leq^{3} \quad \frac{t}{(1-\delta')} - 3\sqrt{\frac{t}{(1-\delta')}} - 1 \\ &\leq \quad \frac{F_0 v}{p} - 3\sqrt{\frac{F_0 v}{p}} \leq \mathrm{E}Q(v) - 3\sqrt{\mathrm{Var}Q(v)} \end{aligned}$$

and thus using Tchebyshev's inequality:

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

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

$$\begin{aligned} t \quad &\geq \quad \frac{t}{(1+\delta')} + 3\sqrt{\frac{t}{(1+\delta')} + 1} + 1 \\ &\geq \quad \frac{F_0 u}{p} + 3\sqrt{\frac{F_0 u}{p}} \geq \mathrm{E}Q(u) + 3\sqrt{\mathrm{Var}Q(v)} \end{aligned}$$

[1]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.

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

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

and thus using Tchebyshev's inequality:

$$\begin{aligned}
P\left(R^* > \left(1+\delta'\right)F_0\right) &= P\left(\operatorname{rank}_t^{\#}(H^*) < \frac{tp}{(1+\delta')F_0}\right) \\
&\leq P(\operatorname{rank}_t^{\#}(H^*) < u) = P(Q(u) \geq t) \\
&\leq P\left(Q(u) \geq \mathrm{E}Q(u) + 3\sqrt{\mathrm{Var}Q(u)}\right) \leq \frac{1}{9}.
\end{aligned} \tag{6}$$

To verfiy Equation 3, note that

$$\min_t(H) = \lfloor \min_t^{\#}(H^*) \rfloor_r \tag{7}$$

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 3 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 7 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 choosen 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 $1/18$. However, even if h is injective, there is still a possibility of collision, because of the application of the rounding operation $\lfloor \cdot \rfloor_r$. The plan is to bound that probability by $1/18$ as well to show Equation 3.

$$\begin{aligned}
& P\left(|R^* - F_0| \leq \delta' F_0 \wedge |R - R^*| > \frac{\delta}{4}F_0\right) \\
\leq\ & P\left(R^* \geq (1-\delta')F_0 \wedge \min_t^{\#}(H^*) \neq \min_t(H) \wedge h \text{ inj.}\right) + P(\neg h \text{ inj.}) \\
\leq\ & P\left(\exists a \neq b \in A. \lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \leq v \wedge h(a) \neq h(b)\right) + \frac{1}{18} \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(\lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \leq v \wedge h(a) \neq h(b)\right) \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(|h(a) - h(b)| \leq v2^{-r} \wedge h(a) \leq v(1+2^{-r}) \wedge h(a) \neq h(b)\right) \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} \sum_{\substack{a',b' \in \{0,\ldots,p-1\} \wedge a' \neq b' \\ |a'-b'| \leq v2^{-r} \wedge a' \leq v(1+2^{-r})}} P(h(a) = a')P(h(b) = b') \\
\leq\ & \frac{1}{18} + 6\frac{F_0^2 v^2}{p^2}2^{-r} \leq \frac{1}{9}.
\end{aligned}$$

Which shows that Equation 3 is true and Equation 5 and 6 implies Equation 2, which means the reasoning in Equation 4 confirms:

$$P(|R - F_0| \leq \delta|F_0|) \geq \frac{2}{3} \tag{8}$$

The following subsection confirms that this is also true for the remaining case, if $F_0 < t$, concluding the proof.

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 least $\frac{1}{18}$, hence:

$$\begin{aligned}
& P\left(|R - F_0| > \delta F_0\right) \\
\leq\ & P\left(R \neq F_0\right) \\
\leq\ & \frac{1}{18} + P\left(R \neq F_0 \wedge h \text{ injective}\right) \\
\leq\ & \frac{1}{18} + P\left(\exists a \neq b \in A . \lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r\right) \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(\lfloor h(a) \rfloor_r = \lfloor h(b) \rfloor_r \wedge h(a) \neq h(b)\right) \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} P\left(|h(a) - h(b)| \leq p2^{-r} \wedge h(a) \neq h(b)\right) \\
\leq\ & \frac{1}{18} + \sum_{a \neq b \in A} \sum_{\substack{a', b' \in \{0, \ldots, p-1\} \\ a' \neq b' \wedge |a' - b'| \leq p2^{-r}}} P(h(a) = a')P(h(b) = b') \\
\leq\ & \frac{1}{18} + F_0^2 2^{-r+1} \leq \frac{1}{9}.
\end{aligned}$$

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.