Formalization of Randomized Approximation Algorithms for Frequency Moments

Emin Karayel

January 10, 2022

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 $x \in dom\ f$
  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 $length\ v \leq length\ t$
  ⟨*proof*⟩

lemma *encoding-by-witness*:
  assumes $\bigwedge x\ y.\ x \in dom\ f \Longrightarrow g\ (the\ (f\ x)@y) = (x,y)$
  shows *is-encoding f*
⟨*proof*⟩

fun *bit-count* **where**
  $bit\text{-}count\ None = \infty$ |
  $bit\text{-}count\ (Some\ x) = ereal\ (length\ x)$

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

lemma *bit-count-append*: $bit\text{-}count\ (x1 @_S x2) = bit\text{-}count\ x1 + bit\text{-}count\ x2$
  ⟨*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* :: $('a \Rightarrow bool\ list\ option) \Rightarrow bool\ list$
  $\Rightarrow 'a\ list \times bool\ list$
  where
    $decode\text{-}list\ e\ (True\#x0) = ($
      $let\ (r1,x1) = decode\ e\ x0\ in\ ($
        $let\ (r2,x2) = decode\text{-}list\ e\ x1\ in\ (r1\#r2,x2)))$ |
    $decode\text{-}list\ e\ (False\#x0) = ([],\ x0)$ |
    $decode\text{-}list\ e\ [] = undefined$
  ⟨*proof*⟩

termination
⟨*proof*⟩

lemma *list-encoding-dom*:
assumes *set* $l \subseteq$ *dom f*
shows $l \in$ *dom* (*list*$_S$ *f*)
⟨*proof*⟩

lemma *list-bit-count*:
bit-count (*list*$_S$ *f xs*) = ($\sum x \leftarrow xs.$ *bit-count* (*f x*) + *1*) + *1*
⟨*proof*⟩

lemma *list-bit-count-est*:
assumes $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *bit-count* (*f x*) $\leq a$
shows *bit-count* (*list*$_S$ *f xs*) $\leq$ *ereal* (*length xs*) $*$ (*a+1*) + *1*
⟨*proof*⟩

lemma *list-bit-count-estI*:
assumes $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow$ *bit-count* (*f x*) $\leq a$
assumes *ereal* (*real* (*length xs*)) $*$ (*a+1*) + *1* $\leq h$
shows *bit-count* (*list*$_S$ *f xs*) $\leq h$
⟨*proof*⟩

lemma *list-encoding-aux*:
assumes *is-encoding f*
shows $x \in$ *dom* (*list*$_S$ *f*) $\Longrightarrow$ *decode-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-encoding-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*:

$decode\text{-}nat\ (nat\text{-}encoding\text{-}aux\ x\ @\ y) = (x,\ y)$
⟨*proof*⟩

lemma *nat-encoding*:
shows $is\text{-}encoding\ N_S$
⟨*proof*⟩

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

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

Encoding for integers

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

fun *decode-int* :: $bool\ list \Rightarrow (int \times bool\ list)$
where
$decode\text{-}int\ (True\#xs) = (\lambda(x{::}nat,y).\ (int\ x,\ y))\ (decode\text{-}nat\ xs)\ |$
$decode\text{-}int\ (False\#xs) = (\lambda(x{::}nat,y).\ (-(int\ x)-1,\ y))\ (decode\text{-}nat\ xs)\ |$
$decode\text{-}int\ [] = undefined$

lemma *int-encoding*: $is\text{-}encoding\ I_S$
⟨*proof*⟩

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

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

Encoding for Cartesian products

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

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

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* ⇒ (*'a* ⇒ *'b encoding*) ⇒ (*'a* × *'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* (λx. *if P x then f* (*g x*) *else None*)
⟨*proof*⟩

Encoding for extensional maps defined on an enumerable set.

definition *encode-extensional* :: *'a list* ⇒ *'b encoding* ⇒ (*'a* ⇒ *'b*) *encoding* (**infixr** $\rightarrow_S$ *65*) **where**
encode-extensional 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 $is\text{-}encoding\ (\lambda x.\ (xs \to_S\ e)\ x)$
⟨*proof*⟩

lemma *extensional-bit-count*:
assumes $f \in extensional\ (set\ xs)$
shows $bit\text{-}count\ ((xs \to_S\ e)\ f) = (\sum x \leftarrow xs.\ bit\text{-}count\ (e\ (f\ x)) + 1) + 1$
⟨*proof*⟩

Encoding for ordered sets.

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

lemma *encode-set*:
assumes *is-encoding e*
shows $is\text{-}encoding\ (\lambda S.\ set_S\ e\ S)$
⟨*proof*⟩

lemma *set-bit-count*:
assumes *finite S*
shows $bit\text{-}count\ (set_S\ e\ S) = (\sum x \in S.\ bit\text{-}count\ (e\ x)+1)+1$
⟨*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 $bit\text{-}count\ (set_S\ f\ S) \le ereal\ (real\ m) * (a+1) + 1$
⟨*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* $(-prec)) \leq$ *truncate-down prec* x
  ⟨*proof*⟩

lemma *truncate-down-eq*:
  assumes *truncate-down* r x = *truncate-down* r y
  shows *abs* $(x-y) \leq$ *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* $f \geq 0$ *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 $(\lfloor x * 2$ *powr* (*real* r − *real-of-int* $\lfloor$*log* 2 $|x|\rfloor)\rfloor) \leq 2$ ^ $(r+1)$ (**is** *?lhs* ≤ -)
⟨*proof*⟩

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

lemma *float-bit-count*:
  fixes m :: *int*
  fixes e :: *int*
  defines $f \equiv$ *float-of* $(m * 2 \text{ powr } e)$
  shows *bit-count* $(F_S\ f) \leq 4 + 2 * (\log 2\ (|m| + 2) + \log 2\ (|e| + 1))$
$\langle proof \rangle$

lemma *float-bit-count-zero*:
  bit-count $(F_S$ (*float-of* $0)) = 4$
  $\langle proof \rangle$

lemma *log-est*: $\log 2\ (\text{real } n + 1) \leq n$
$\langle proof \rangle$

lemma *truncate-float-bit-count*:
  bit-count $(F_S$ (*float-of* (*truncate-down* $r\ x))) \leq 8 + 4 * \text{real } r + 2*\log 2\ (2 +$ *abs* $(\log 2$ (*abs* $x)))$
  (**is** *?lhs* $\leq$ *?rhs*)
$\langle proof \rangle$

end

4 Extensions to "HOL.List"

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

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

lemma *count-list-gr-1*:
  $(x \in$ *set xs*$) = ($*count-list xs x* $\geq 1)$
  $\langle proof \rangle$

lemma *count-list-append*: *count-list* $(xs@ys)\ v =$ *count-list xs v* $+$ *count-list ys v*
  $\langle proof \rangle$

lemma *count-list-card*: *count-list xs x* $=$ *card* $\{k.\ k <$ *length xs* $\wedge$ *xs* ! $k = x\}$
$\langle proof \rangle$

lemma *card-gr-1-iff*:
  assumes *finite S*
  assumes $x \in S$
  assumes $y \in S$
  assumes $x \neq y$
  shows *card* $S > 1$
  $\langle proof \rangle$

lemma *count-list-ge-2-iff*:

assumes $y < z$
assumes $z < \textit{length xs}$
assumes $\textit{xs} \,!\, y = \textit{xs} \,!\, z$
shows $\textit{count-list xs} \; (\textit{xs} \,!\, y) > \textit{1}$
⟨*proof*⟩

end

5 Frequency Moments

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

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

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

end

6 Primes

In this section we introduce a function that finds primes above a given threshold.

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

begin

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

function *find-prime-above* :: $\textit{nat} \Rightarrow \textit{nat}$ **where**
find-prime-above $n = (\textit{if prime } n \textit{ then } n \textit{ else find-prime-above } (\textit{Suc}\ n))$
⟨*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 $n \geq 2$
  ⟨*proof*⟩

lemma *find-prime-above-lower-bound*:
  find-prime-above $n \geq n$
  ⟨*proof*⟩

lemma *find-prime-above-upper-boundI*:
  assumes *prime m*
  shows $n \leq m \Longrightarrow$ *find-prime-above* $n \leq m$
⟨*proof*⟩

lemma *find-prime-above-upper-bound*:
  find-prime-above $n \leq 2*n+2$
⟨*proof*⟩

end

7 Extensions to "HOL-Library.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 $\bigwedge n\ M\ x.\ P\ M \Longrightarrow \neg(x \in\#\ M) \Longrightarrow n > 0 \Longrightarrow P\ (M +$ *replicate-mset* $n\ x)$
  shows $P\ M$
⟨*proof*⟩

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

lemma *sum-collapse*:
  fixes $f :: 'a \Rightarrow 'b$::{*comm-monoid-add*}
  assumes *finite A*
  assumes $z \in A$
  assumes $\bigwedge y.\ y \in A \Longrightarrow y \neq z \Longrightarrow f\ y = 0$
  shows *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 :: $'a \Rightarrow 'b$::{*ring*,*semiring-1*}
shows *sum-list* (*map f xs*) = ($\sum$ *x* ∈ *set xs*. *of-nat* (*count-list xs x*) ∗ *f x*)
⟨*proof*⟩

lemma *prod-list-eval*:
fixes f :: $'a \Rightarrow 'b$::{*ring*,*semiring-1*,*comm-monoid-mult*}
shows *prod-list* (*map f xs*) = ($\prod$ *x* ∈ *set xs*. (*f x*)^(*count-list 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 Probabilities and Independent Families

Some additional results about probabilities and independent families.

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*)

$\langle proof \rangle$

lemma *set-comp-subsetI*: $(\bigwedge x.\ P\ x \Longrightarrow f\ x \in B) \Longrightarrow \{f\ x|x.\ P\ x\} \subseteq B$
$\langle proof \rangle$

lemma *set-comp-cong*:
assumes $\bigwedge x.\ P\ x \Longrightarrow f\ x = h\ (g\ x)$
shows $\{f\ x|\ x.\ P\ x\} = h\ `\ \{g\ x|\ x.\ P\ x\}$
$\langle proof \rangle$

lemma *indep-sets-distr*:
assumes $f \in measurable\ M\ N$
assumes *prob-space* M
assumes *prob-space.indep-sets* $M\ (\lambda i.\ (\lambda a.\ f\ -`\ a \cap space\ M)\ `\ A\ i)\ I$
assumes $\bigwedge i.\ i \in I \Longrightarrow A\ i \subseteq sets\ N$
shows *prob-space.indep-sets* $(distr\ M\ N\ f)\ A\ I$
$\langle proof \rangle$

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

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

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

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

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

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

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

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

lemma *pmf-mono-1*:
assumes $\bigwedge x.\ x \in P \Longrightarrow x \in set\text{-}pmf\ \Omega \Longrightarrow x \in Q$
shows *measure* $(measure\text{-}pmf\ \Omega)\ P \leq measure\ (measure\text{-}pmf\ \Omega)\ Q$
⟨*proof*⟩

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

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

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

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

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

lemma (**in** *prob-space*) *var-sum-1*:
  fixes $f :: 'b \Rightarrow 'a \Rightarrow real$
  assumes *finite I*
  assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in borel\text{-}measurable\ M$
  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 $\bigwedge i.\ i \in I \Longrightarrow f\ i \in borel\text{-}measurable\ M$
  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 $\bigwedge i.\ i \in I \Longrightarrow f\ i \in borel\text{-}measurable\ M$
  assumes $\bigwedge i.\ i \in I \Longrightarrow integrable\ M\ (\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
  assumes $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow i \neq j \Longrightarrow indep\text{-}var\ borel\ (f\ i)\ borel\ (f\ j)$
  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 $indep\text{-}vars\ (\lambda\text{-}.\ M')\ f\ \{i, j\}$
  shows $indep\text{-}var\ M'\ (f\ i)\ M'\ (f\ j)$
⟨*proof*⟩

lemma (**in** *prob-space*) *var-sum-pairwise-indep-2*:

fixes $f :: {}'b \Rightarrow {}'a \Rightarrow \mathit{real}$
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in \mathit{borel\text{-}measurable}\ M$
assumes $\bigwedge i.\ i \in I \Longrightarrow \mathit{integrable}\ M\ (\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
assumes $\bigwedge J.\ J \subseteq I \Longrightarrow \mathit{card}\ J = 2 \Longrightarrow \mathit{indep\text{-}vars}\ (\lambda\ \text{-}.\ \mathit{borel})\ f\ J$
shows $\mathit{variance}\ (\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.\ \mathit{variance}\ (f\ i))$
$\langle \mathit{proof} \rangle$

lemma (**in** *prob-space*) *var-sum-all-indep*:
fixes $f :: {}'b \Rightarrow {}'a \Rightarrow \mathit{real}$
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in \mathit{borel\text{-}measurable}\ M$
assumes $\bigwedge i.\ i \in I \Longrightarrow \mathit{integrable}\ M\ (\lambda\omega.\ f\ i\ \omega\hat{\ }2)$
assumes $\mathit{indep\text{-}vars}\ (\lambda\ \text{-}.\ \mathit{borel})\ f\ I$
shows $\mathit{variance}\ (\lambda\omega.\ (\sum i \in I.\ f\ i\ \omega)) = (\sum i \in I.\ \mathit{variance}\ (f\ i))$
$\langle \mathit{proof} \rangle$

end

9 Median

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

fun *sort-primitive* **where**
$\mathit{sort\text{-}primitive}\ i\ j\ f\ k = (\mathit{if}\ k = i\ \mathit{then}\ \mathit{min}\ (f\ i)\ (f\ j)\ \mathit{else}\ (\mathit{if}\ k = j\ \mathit{then}\ \mathit{max}\ (f\ i)\ (f\ j)\ \mathit{else}\ f\ k))$

fun *sort-map* **where**
$\mathit{sort\text{-}map}\ f\ n = \mathit{fold}\ \mathit{id}\ [\mathit{sort\text{-}primitive}\ j\ i.\ i <\!-\ [0..<n],\ j <\!-\ [0..<i]]\ f$

lemma *sort-map-ind*:
$\mathit{sort\text{-}map}\ f\ (\mathit{Suc}\ n) = \mathit{fold}\ \mathit{id}\ [\mathit{sort\text{-}primitive}\ j\ n.\ j <\!-\ [0..<n]]\ (\mathit{sort\text{-}map}\ f\ n)$
$\langle \mathit{proof} \rangle$

lemma *sort-map-strict-mono*:
fixes $f :: \mathit{nat} \Rightarrow {}'b :: \mathit{linorder}$
shows $j < n \Longrightarrow i < j \Longrightarrow \mathit{sort\text{-}map}\ f\ n\ i \le \mathit{sort\text{-}map}\ f\ n\ j$
$\langle \mathit{proof} \rangle$

lemma *sort-map-mono*:
fixes $f :: \mathit{nat} \Rightarrow {}'b :: \mathit{linorder}$
shows $j < n \Longrightarrow i \le j \Longrightarrow \mathit{sort\text{-}map}\ f\ n\ i \le \mathit{sort\text{-}map}\ f\ n\ j$
$\langle \mathit{proof} \rangle$

lemma *sort-map-perm*:
fixes $f :: \mathit{nat} \Rightarrow {}'b :: \mathit{linorder}$

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

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

definition *interval* :: (*'a* :: *linorder*) *set* ⇒ *bool* **where**
interval I = (∀ *x y z*. *x* ∈ *I* ⟶ *z* ∈ *I* ⟶ *x* ≤ *y* ⟶ *y* ≤ *z* ⟶ *y* ∈ *I*)

lemma *interval-rule*:
assumes *interval I*
assumes *a* ≤ *x x* ≤ *b*
assumes *a* ∈ *I*
assumes *b* ∈ *I*
shows *x* ∈ *I*
⟨*proof*⟩

lemma *sorted-int*:
assumes *interval I*
assumes *sorted xs*
assumes *k* < *length xs i* ≤ *j j* ≤ *k*
assumes *xs* ! *i* ∈ *I xs* ! *k* ∈ *I*
shows *xs* ! *j* ∈ *I*
⟨*proof*⟩

lemma *mid-in-interval*:
assumes *2*∗*length* (*filter* (λ*x*. *x* ∈ *I*) *xs*) > *length xs*
assumes *interval I*
assumes *sorted xs*
shows *xs* ! (*length xs div 2*) ∈ *I*
⟨*proof*⟩

lemma *median-est*:
fixes δ :: *real*
assumes *2*∗*card* {*k*. *k* < *n* ∧ *abs* (*f k* − μ) ≤ δ} > *n*
shows *abs* (*median f n* − μ) ≤ δ

⟨*proof*⟩

lemma *median-est-2*:
fixes *a b* :: *real*
assumes $2*card\ \{k.\ k < n \wedge f\ k \in \{a..b\}\} > n$
shows $median\ f\ n \in \{a..b\}$
⟨*proof*⟩

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

lemma (**in** *prob-space*) *median-bound-gen*:
fixes *a b* :: *real*
fixes *n* :: *nat*
assumes $\alpha > 0$
assumes $\varepsilon \in \{0<..<1\}$
assumes *indep-vars* (λ-. *borel*) *X* $\{0..<n\}$
assumes $n \geq -\ ln\ \varepsilon\ /\ (2 * \alpha^2)$
assumes $\bigwedge i.\ i < n \Longrightarrow \mathcal{P}(\omega\ in\ M.\ X\ i\ \omega \in \{a..b\}) \geq 1/2+\alpha$
shows $\mathcal{P}(\omega\ in\ M.\ median\ (\lambda i.\ X\ i\ \omega)\ n \in \{a..b\}) \geq 1-\varepsilon$ (**is** $\mathcal{P}(\omega\ in\ M.\ ?lhs\ \omega) \geq ?C$)
⟨*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 $\mathcal{P}(\omega\ in\ M.\ abs\ (median\ (\lambda i.\ X\ i\ \omega)\ n - \mu) \leq \delta) \geq 1-\varepsilon$
⟨*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 f n = median g n*
  ⟨*proof*⟩

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

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

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

theory *OrderStatistics*
  imports *Main HOL−Library.Multiset List-Ext Multiset-Ext Set-Ext*
begin

Returns the rank of an element within a set.

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

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*⟩

Returns the k smallest elements of a finite set.

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

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 $(\lambda x.$ *rank-of* x $S)$ ` S = $\{0..<$*card* $S\}$
  ⟨*proof*⟩

lemma *card-least*:
  assumes *finite* S
  shows *card* (*least* k S) = *min* k (*card* S)
⟨*proof*⟩

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

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

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

lemma *least-mono-commute*:

assumes *finite S*
assumes *strict-mono-on f S*
shows f ' *least k S* = *least k* (f ' S)
⟨*proof*⟩

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

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

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

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

lemma *nth-mset-bound-left-excl*:
assumes $k <$ *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* ≤ 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 Counting Polynomials

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

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.

Here we are interested in a more generic result (over finite fields). We also want to count 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 we have 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*))

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

We first show that its image is a subset of the product space mentioned above, after that we will show that *split-poly* is injective and finally we will be able to show 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* $(x \ominus_{\textit{poly-ring } F} y)$ 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* $(x \ominus_{\textit{poly-ring } F} y)$ 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* $x < n \vee x = \mathbf{0}_{\textit{poly-ring } F}$
⟨*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:
$card\ \{f.\ range\ f \subseteq B \wedge (\forall x.\ x \geq n \longrightarrow f\ x = y)\} = card\ B\ \hat{}\ n$ (**is** $card\ ?A = ?B$) **and**
finite-mostly-constant-maps:
$finite\ \{f.\ range\ f \subseteq B \wedge (\forall x.\ x \geq n \longrightarrow f\ x = y)\}$
⟨*proof*⟩

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

lemma *inv-subsetI*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \in B \Longrightarrow x \in C$
shows $f\ -\text{`}\ B \cap A \subseteq C$
⟨*proof*⟩

lemma *interpolating-polynomials-count*:
assumes *field* F
assumes *finite* $(carrier\ F)$
assumes $K \subseteq carrier\ F$
assumes $f\ \text{`}\ K \subseteq carrier\ F$
shows $card\ \{\omega \in bounded\text{-}degree\text{-}polynomials\ F\ (card\ K + n).\ (\forall k \in K.\ ring.eval\ F\ \omega\ k = f\ k)\} =$
$card\ (carrier\ F)\hat{}n$
(**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*)) $\{\omega.\ P\ (\omega\ i)\}$ = *measure* (M i) $\{\omega.\ P\ \omega\}$
⟨*proof*⟩

lemma *range-inter*: *range* (($\cap$) 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* {} (($\lambda k.\ \{k\}$) ‘ M)) = *Pow M*
⟨*proof*⟩

lemma *indep-vars-pmf*:
assumes $\bigwedge a\ J.\ J \subseteq I \Longrightarrow$ *finite J* $\Longrightarrow$

$\mathcal{P}(\omega \textit{ in measure-pmf } M.\ \forall\, i \in J.\ X\ i\ \omega = a\ i) = (\prod i \in J.\ \mathcal{P}(\omega \textit{ in measure-pmf } M.\ X\ i\ \omega = a\ i))$
shows *prob-space.indep-vars* (*measure-pmf* M) ($\lambda i.$ *measure-pmf* (M' i)) X I
⟨*proof*⟩

lemma *indep-vars-restrict*:
fixes M :: $'a \Rightarrow 'b$ *pmf*
fixes J :: $'c$ *set*
assumes *disjoint-family-on* f J
assumes $J \neq \{\}$
assumes $\bigwedge i.\ i \in J \Longrightarrow f\ i \subseteq I$
assumes *finite* I
shows *prob-space.indep-vars* (*measure-pmf* (*prod-pmf* I M)) ($\lambda i.$ *measure-pmf* (*prod-pmf* (f i) M)) ($\lambda i\ \omega.$ *restrict* ω (f i)) J
⟨*proof*⟩

lemma *indep-vars-restrict-intro*:
fixes M :: $'a \Rightarrow 'b$ *pmf*
fixes J :: $'c$ *set*
assumes $\bigwedge \omega\ i.\ i \in J \Longrightarrow X\ i\ \omega = X\ i$ (*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 $\bigwedge \omega\ i.\ i \in J \Longrightarrow X\ i\ \omega \in$ *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: $integral^L$ (*prod-pmf* I M) ($\lambda x.$ ($\prod i \in I.\ f\ i\ (x\ i)$)) =
($\prod$ $i \in I.$ $integral^L$ (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: $integral^L$ (*prod-pmf I M*) ($\lambda x.$ (*f* (*x i*))) = $integral^L$ (*M i*) *f* (**is** *?B*)
⟨*proof*⟩

lemma *variance-prod-pmf-slice*:
fixes f :: $'a \Rightarrow real$
assumes $i \in I$ *finite I*
assumes *integrable* (*measure-pmf* (*M i*)) ($\lambda\omega.$ *f* ω^*2*)
shows *prob-space.variance* (*prod-pmf I M*) ($\lambda\omega.$ *f* (ω *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* ($\lambda i.$ *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 :: $'a \Rightarrow 'b$ *pmf*
fixes f :: $'b \Rightarrow 'c$
assumes *finite I*
shows *prod-pmf I M* $\gg=$ (λx. *return-pmf* ($\lambda i \in I$. f (x i))) = *prod-pmf I* (λi. (M i $\gg=$ (λx. *return-pmf* (f x))))
⟨*proof*⟩

lemma *pair-pmfI*:
A $\gg=$ (λa. B $\gg=$ (λb. *return-pmf* (f a b))) = *pair-pmf A B* $\gg=$ ($\lambda(a,b)$. *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* :: (($'i \Rightarrow 'a$) *pmf*)) (*prod-pmf I B* :: (($'i \Rightarrow 'b$) *pmf*)) =
prod-pmf I (λi. *pair-pmf* (A i) (B i)) $\gg=$
(λf. *return-pmf* (*restrict* (*fst* $\circ$ f) I, *restrict* (*snd* $\circ$ f) I))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

end

13 Universal Hash Families

theory *UniversalHashFamily*
imports *Main PolynomialCounting Product-PMF-Ext*
begin

definition *k-universal* **where**
k-universal k H f U V = (
($\forall x \in U$. $\forall h \in H$. f h $x \in V$) $\wedge$ *finite V* $\wedge$ $V \neq \{\}$ $\wedge$
($\forall x \in U$. $\forall v \in V$. $\mathcal{P}$(h *in pmf-of-set H*. f h x = v) = *1* / *real* (*card V*)) $\wedge$
($\forall x \subseteq U$. *card* $x \leq k$ $\wedge$ *finite* $x \longrightarrow$ *prob-space.indep-vars* (*pmf-of-set H*) (λ-. *pmf-of-set V*) f x))

A k-independent 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 k universe elements: $x_1, \cdots, x_k$ the functions $h(x_1), \cdots, h(x_m)$ form independent random variables.

In this section, we construct k-independent 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* **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)^(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)^(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)))^card K*

⟨*proof*⟩

lemma *hash-prob-single*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes $x \in$ *carrier F*
assumes $1 \leq n$
assumes $y \in$ *carrier F*
shows $\mathcal{P}$(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* $\wedge$ *finite* (*carrier F*)
assumes *J*$\subseteq$*carrier F*
assumes *finite J*
assumes *card J* $\leq n$
assumes $1 \leq n$
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* **where**
k-wise-indep-vars k M' X' I = ($\forall J \subseteq I$. *card J* $\leq k \longrightarrow$ *finite J* $\longrightarrow$ *indep-vars M' X' J*)

lemma *hash-k-wise-indep*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes $1 \leq n$
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* $\wedge$ *finite* (*carrier F*)
assumes $\omega \in$ *bounded-degree-polynomials F n*
assumes *degree* ω = *1*
shows *inj-on* (λx. *hash F x* ω) (*carrier F*)
⟨*proof*⟩

lemma (**in** *prob-space*) *k-wise-subset*:
assumes *k-wise-indep-vars k M' X' I*
assumes $J \subseteq I$
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 *UniversalHashFamilyOfPrime*
imports *Field UniversalHashFamily 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*)) $n \neq \{\}$
⟨*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*) (*UniversalHashFamily.hash* (*ZFact p*) (*zfact-embed p x*) *f*)

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

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

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

lemma *hash-prob*:
assumes *prime p*
assumes $K \subseteq$ {*0..<p*}
assumes *y* ‘ $K \subseteq$ {*0..<p*}
assumes *card* $K \leq n$
shows $\mathcal{P}$(ω *in measure-pmf* (*pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int p*)) *n*)).

$(\forall x \in K.\ hash\ p\ x\ \omega = (y\ x))) = 1\ /\ real\ p\hat{}card\ K$
⟨*proof*⟩

lemma *hash-prob-2*:
assumes *prime p*
assumes *inj-on x K*
assumes $x\ `\ K \subseteq \{0..<p\}$
assumes $y\ `\ K \subseteq \{0..<p\}$
assumes $card\ K \leq n$
shows $\mathcal{P}(\omega\ in\ measure\text{-}pmf\ (pmf\text{-}of\text{-}set\ (bounded\text{-}degree\text{-}polynomials\ (ZFact\ (int\ p))\ n)).$
$(\forall k \in K.\ hash\ p\ (x\ k)\ \omega = (y\ k))) = 1\ /\ real\ p\hat{}card\ K$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *hash-prob-range*:
assumes *prime p*
assumes $x < p$
assumes $n > 0$
shows $\mathcal{P}(\omega\ in\ measure\text{-}pmf\ (pmf\text{-}of\text{-}set\ (bounded\text{-}degree\text{-}polynomials\ (ZFact\ (int\ p))\ n)).$
$hash\ p\ x\ \omega \in A) = card\ (A \cap \{0..<p\})\ /\ p$
⟨*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*)))
$n\ (\lambda\text{-}.\ pmf\text{-}of\text{-}set\ \{0..<p\})\ (hash\ p)\ \{0..<p\}$
⟨*proof*⟩

14.1 Encoding

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

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

lemma *bounded-degree-polynomial-bit-count*:
assumes $p > 0$
assumes $x \in bounded\text{-}degree\text{-}polynomials\ (ZFact\ p)\ n$
shows $bit\text{-}count\ (list_S\ (zfact_S\ p)\ x) \leq ereal\ (real\ n * (2 * log\ 2\ p + 2) + 1)$
⟨*proof*⟩

end

15 Landau Symbols (Extensions)

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 $f2 \in O[F'](g)$
  assumes $f1 \in O[F'](g)$
  shows $(\lambda x.\ f1\ x + f2\ x) \in O[F'](g)$
  ⟨*proof*⟩

lemma *landau-sum*:
  assumes $eventually\ (\lambda x.\ g1\ x \geq (0::real))\ F'$
  assumes $eventually\ (\lambda x.\ g2\ x \geq 0)\ F'$
  assumes $f1 \in O[F'](g1)$
  assumes $f2 \in O[F'](g2)$
  shows $(\lambda x.\ f1\ x + f2\ x) \in O[F'](\lambda x.\ g1\ x + g2\ x)$
⟨*proof*⟩

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 $f \in O[F'](\lambda x.\ g1\ x + g2\ x)$
⟨*proof*⟩

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 $f \in O[F'](\lambda x.\ g1\ x + g2\ x)$
⟨*proof*⟩

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

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 $(\lambda x.\ ln\ (f\ x)) \in O[F'](\lambda x.\ ln\ (g\ x))$
⟨*proof*⟩

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

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

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

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* (λ*a*. {..< *a*}) ∪ *range* (λ*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 OrderStatistics UniversalHashFamilyOfPrime Encoding*
  Frequency-Moments Landau-Ext
begin

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*, λ*i* ∈ {*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* (λ*i* ∈ {*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*)))
) *s*)

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

lemma *f0-alg-sketch*:
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\delta \in \{0{<}..{<}1\}$
assumes $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
defines *sketch* ≡ *fold* (λ*a state*. *state* ⤜ *f0-update a*) *as* (*f0-init* δ ε *n*)
defines $t \equiv nat\ \lceil 80\ /\ (real\text{-}of\text{-}rat\ \delta)^2 \rceil$
defines $s \equiv nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)) \rceil$
defines *p* ≡ *find-prime-above* (*max n 19*)
defines $r \equiv nat\ (4 * \lceil log\ 2\ (1\ /\ real\text{-}of\text{-}rat\ \delta) \rceil + 24)$
shows *sketch* = *map-pmf* (λ*x*. (*s*,*t*,*p*,*r*, *x*, λ*i* ∈ {*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 (**in** *prob-space*) *prob-sub-additive*:
assumes *Collect P* ∈ *sets M*
assumes *Collect Q* ∈ *sets M*
shows $\mathcal{P}(\omega\ in\ M.\ P\ \omega \vee Q\ \omega) \le \mathcal{P}(\omega\ in\ M.\ P\ \omega) + \mathcal{P}(\omega\ in\ M.\ Q\ \omega)$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-sub-additiveI*:
assumes *Collect P* ∈ *sets M*
assumes *Collect Q* ∈ *sets M*
assumes $\mathcal{P}(\omega\ in\ M.\ P\ \omega) \le r1$
assumes $\mathcal{P}(\omega\ in\ M.\ Q\ \omega) \le r2$
shows $\mathcal{P}(\omega\ in\ M.\ P\ \omega \vee Q\ \omega) \le r1 + r2$
⟨*proof*⟩

lemma (**in** *prob-space*) *prob-mono*:
assumes *Collect Q* ∈ *sets M*
assumes $\bigwedge\omega.\ \omega \in space\ M \Longrightarrow P\ \omega \Longrightarrow Q\ \omega$
shows $\mathcal{P}(\omega\ in\ M.\ P\ \omega) \le \mathcal{P}(\omega\ in\ M.\ Q\ \omega)$
⟨*proof*⟩

lemma *in-events-pmf*: $A \in$ *measure-pmf.events* Ω
⟨*proof*⟩

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

lemma *pmf-mono*:
assumes $\bigwedge x.\ x \in P \Longrightarrow x \in Q$
shows *measure* (*measure-pmf* Ω) $P \leq$ *measure* (*measure-pmf* Ω) Q
⟨*proof*⟩

lemma *abs-ge-iff*: $((x::real) \leq abs\ y) = (x \leq y \lor x \leq -y)$
⟨*proof*⟩

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

lemma *count-nat-abs-diff-2*:
fixes $x :: nat$
fixes $q :: real$
assumes $q \geq 0$
defines $A \equiv \{(k::nat).\ abs\ (real\ x - real\ k) \leq q \land k \neq x\}$
shows *real* (*card* A) $\leq 2 * q$ **and** *finite* A
⟨*proof*⟩

lemma *f0-collision-prob*:
fixes $p :: nat$
assumes *Factorial-Ring.prime* p
defines $\Omega \equiv$ *pmf-of-set* (*bounded-degree-polynomials* (*ZFact* (*int* p)) 2)
assumes $M \subseteq \{0..<p\}$
assumes $c \geq 1$
assumes $r \geq 1$
shows $\mathcal{P}(\omega$ *in measure-pmf* Ω.
$\exists\, x \in M.\ \exists\, y \in M.$
$x \neq y\ \land$
truncate-down r (*hash* $p\ x\ \omega$) $\leq c\ \land$
truncate-down r (*hash* $p\ x\ \omega$) = *truncate-down* r (*hash* $p\ y\ \omega$)) $\leq$
$6 * (real\ (card\ M))^2 * c^2 * 2\ powr\ -r\ /\ (real\ p)\ ^2 + 1/real\ p$ (**is** $\mathcal{P}(\omega$ *in* -. $?l\ \omega) \leq\ ?r1 +\ ?r2$)
⟨*proof*⟩

lemma *inters-compr*: $A \cap \{x.\ P\ x\} = \{x \in A.\ P\ x\}$
⟨*proof*⟩

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

⟨proof⟩

theorem *f0-alg-correct*:
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\delta \in \{0{<}..{<}1\}$
assumes $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
defines $M \equiv fold\ (\lambda a\ state.\ state \ggg f0\text{-}update\ a)\ as\ (f0\text{-}init\ \delta\ \varepsilon\ n) \ggg f0\text{-}result$
shows $\mathcal{P}(\omega\ in\ measure\text{-}pmf\ M.\ |\omega - F\ 0\ as| \le \delta * F\ 0\ as) \ge 1 - of\text{-}rat\ \varepsilon$
⟨proof⟩

fun *f0-space-usage* :: $(nat \times rat \times rat) \Rightarrow real$ **where**
$f0\text{-}space\text{-}usage\ (n, \varepsilon, \delta) = ($
$let\ s = nat\ \lceil -18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)\rceil\ in$
$let\ r = nat\ (4 * \lceil log\ 2\ (1\ /\ real\text{-}of\text{-}rat\ \delta)\rceil + 24)\ in$
$let\ t = nat\ \lceil 80\ /\ (real\text{-}of\text{-}rat\ \delta)^2\ \rceil\ in$
$8 +$
$2 * log\ 2\ (real\ s + 1) +$
$2 * log\ 2\ (real\ t + 1) +$
$2 * log\ 2\ (real\ n + 10) +$
$2 * log\ 2\ (real\ r + 1) +$
$real\ s * (12 + 4 * log\ 2\ (10 + real\ n) +$
$real\ t * (11 + 4 * r + 2 * log\ 2\ (log\ 2\ (real\ n + 9)))))$

definition *encode-state* **where**
$encode\text{-}state =$
$N_S \times_D (\lambda s.$
$N_S \times_S ($
$N_S \times_D (\lambda p.$
$N_S \times_S ($
$([0..{<}s] \rightarrow_S (list_S\ (zfact_S\ p))) \times_S$
$([0..{<}s] \rightarrow_S (set_S\ F_S))))))$

lemma $inj\text{-}on\ encode\text{-}state\ (dom\ encode\text{-}state)$
⟨proof⟩

lemma *f-subset*:
assumes $g\ `\ A \subseteq h\ `\ B$
shows $(\lambda x.\ f\ (g\ x))\ `\ A \subseteq (\lambda x.\ f\ (h\ x))\ `\ B$
⟨proof⟩

theorem *f0-exact-space-usage*:
assumes $\varepsilon \in \{0{<}..{<}1\}$
assumes $\delta \in \{0{<}..{<}1\}$
assumes $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
defines $M \equiv fold\ (\lambda a\ state.\ state \ggg f0\text{-}update\ a)\ as\ (f0\text{-}init\ \delta\ \varepsilon\ n)$
shows $AE\ \omega\ in\ M.\ bit\text{-}count\ (encode\text{-}state\ \omega) \le f0\text{-}space\text{-}usage\ (n, \varepsilon, \delta)$
⟨proof⟩

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

In this section, we define a function that enumerates all the partitions of $\{0..<n\}$. We represent the partitions 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* ⇒ (*nat* × *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* ⇒ *'a list* ⇒ *bool* **where**
has-eq-relation r xs = (*length xs* = *length r* ∧ (∀ *i* < *length xs*. ∀ *j* < *length xs*. (*xs* ! *i* = *xs* ! *j*) = (*r* ! *i* = *r* ! *j*)))

lemma *filter-one-elim*:
length (*filter p xs*) = *1* ⟹ (∃ *u v w*. *xs* = *u*@*v*#*w* ∧ *p v* ∧ *length* (*filter p u*) = *0* ∧ *length* (*filter p w*) = *0*)
(**is** *?A xs* ⟹ *?B xs*)
⟨*proof*⟩

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

lemma *enum-partitions-aux-range*:
x ∈ *set* (*enum-partitions-aux n*) ⟹ *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 UniversalHashFamilyOfPrime Frequency-Moments Landau-Ext*
begin

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* / δ^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*, λ*i* ∈ {*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* (λi_2 ∈ {*0*..<s_2}.
($\sum$ 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)) s_2
)

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*))
(λω. *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* $\leq$ *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* $\equiv$ *fold* (λ*a state. state* $\gg=$ *f2-update a*) *as* (*f2-init* $\delta\ \varepsilon$ *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* = $\Omega \gg=$ (λ*h. 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 $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
  defines $M \equiv$ *fold* (λ*a state. state* $\gg=$ *f2-update a*) *as* (*f2-init* $\delta\ \varepsilon$ *n*) $\gg=$ *f2-result*
  shows $\mathcal{P}(\omega\ in\ measure\text{-}pmf\ M.\ |\omega - F\ 2\ as| \leq \delta * F\ 2\ as) \geq 1 - of\text{-}rat\ \varepsilon$
⟨*proof*⟩

fun *f2-space-usage* :: (*nat* $\times$ *nat* $\times$ *rat* $\times$ *rat*) $\Rightarrow$ *real* **where**
  f2-space-usage $(n, m, \varepsilon, \delta) = ($
    let $s_1 = nat\ \lceil 6\ /\ \delta^2\ \rceil$ *in*
    let $s_2 = nat\ \lceil -(18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)) \rceil$ *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* **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 $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
defines $M \equiv fold\ (\lambda a\ state.\ state \ggg f2\text{-}update\ a)\ as\ (f2\text{-}init\ \delta\ \varepsilon\ n)$
shows $AE\ \omega\ in\ M.\ bit\text{-}count\ (encode\text{-}state\ \omega) \leq f2\text{-}space\text{-}usage\ (n,\ length\ as,\ \varepsilon,$
$\delta)$
⟨*proof*⟩

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

$(ln\ (1\ /\ of\text{-}rat\ \varepsilon))\ /\ (of\text{-}rat\ \delta)^2 * (ln\ (real\ n) + ln\ (real\ m)))$
(**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

type-synonym $fk\text{-}state = nat \times nat \times nat \times nat \times (nat \times nat \Rightarrow (nat \times nat))$

fun $fk\text{-}init :: nat \Rightarrow rat \Rightarrow rat \Rightarrow nat \Rightarrow fk\text{-}state\ pmf$ **where**
$fk\text{-}init\ k\ \delta\ \varepsilon\ n =$
do {
$let\ s_1 = nat\ \lceil 3*real\ k*(real\ n)\ powr\ (1-1/\ real\ k)/\ (real\text{-}of\text{-}rat\ \delta)^2\rceil;$
$let\ s_2 = nat\ \lceil -18 * ln\ (real\text{-}of\text{-}rat\ \varepsilon)\rceil;$
$return\text{-}pmf\ (s_1,\ s_2,\ k,\ 0,\ (\lambda\text{-}.\ undefined))$
}

fun $fk\text{-}update :: nat \Rightarrow fk\text{-}state \Rightarrow fk\text{-}state\ pmf$ **where**
$fk\text{-}update\ a\ (s_1,\ s_2,\ k,\ m,\ r) =$

```
    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 (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)) s₂
    )
```

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* $\{k.\ k < \textit{length as}\} \gg= (\lambda k.\ \textit{return-pmf}\ (\textit{as} ! k,\ \textit{count-list}\ (\textit{drop}\ (k+1)\ \textit{as})\ (\textit{as} ! k)))$
$= \textit{pmf-of-set}\ \{(u,v).\ v < \textit{count-list as } u\}$
⟨*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* $(0,\ \textit{return-pmf undefined}) =$
$(\textit{length as},\ \textit{pmf-of-set}\ \{k.\ k < \textit{length as}\} \gg= (\lambda k.\ \textit{return-pmf}\ (\textit{as} ! k,\ \textit{count-list}\ (\textit{drop}\ (k+1)\ \textit{as})\ (\textit{as} ! k))))$
⟨*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* $(0,\ \textit{return-pmf}\ (\lambda\text{-}.\ \textit{undefined}))$
$= (\textit{length as},\ \textit{prod-pmf}\ (\{0..<s_1\} \times \{0..<s_2\})\ (\lambda\text{-}.\ (\textit{snd}\ (\textit{fold}\ (\lambda x\ (c,\textit{state}).\ (c+1,\ \textit{state} \gg= \textit{fk-update}''\ x\ c))\ \textit{as}\ (0,\ \textit{return-pmf undefined})))))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *fk-alg-aux-1*:
fixes k :: *nat*
fixes ε :: *rat*
assumes $\delta > 0$
assumes $\bigwedge a.\ a \in \textit{set as} \Longrightarrow a < n$
assumes *as* $\neq$ []
defines $\textit{sketch} \equiv \textit{fold}\ (\lambda a\ \textit{state}.\ \textit{state} \gg= \textit{fk-update}\ a)\ \textit{as}\ (\textit{fk-init}\ k\ \delta\ \varepsilon\ n)$
defines $s_1 \equiv \textit{nat}\ \lceil 3{*}\textit{real}\ k{*}(\textit{real}\ n)\ \textit{powr}\ (1{-}1/\ \textit{real}\ k)/\ (\textit{real-of-rat}\ \delta)^2\rceil$
defines $s_2 \equiv \textit{nat}\ \lceil -(18 * \ln\ (\textit{real-of-rat}\ \varepsilon))\rceil$
shows *sketch* =

map-pmf (λx. (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* (λ-. *undefined*))))
⟨*proof*⟩

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

lemma *power-diff-est*:
assumes $k > 0$
assumes (a :: *real*) ≥ b
assumes $b \geq 0$
shows a^k −b^k ≤ (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* (λx. f x ∗ g x) A| ≤ (*sum* (λx. |f x| *powr* p) A) *powr* ($1/p$) ∗ (*sum* (λx. |g x| *powr* q) A) *powr* ($1/q$)
⟨*proof*⟩

lemma *fk-estimate*:
assumes *as* ≠ []
assumes $\bigwedge a$. $a \in$ *set as* $\Longrightarrow a < n$
assumes $k \geq 1$
shows *real* (*length as*) ∗ *real-of-rat* (F (2∗k−1) *as*) ≤ *real* n *powr* (1 − 1 / *real* k) ∗ (*real-of-rat* (F k *as*))^2
(**is** *?lhs* ≤ *?rhs*)
⟨*proof*⟩

lemma *fk-alg-core-exp*:
assumes *as* ≠ []
assumes $k \geq 1$
shows *has-bochner-integral* (*measure-pmf* (*pmf-of-set* {(u, v). v < *count-list as* u}))
(λa. *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* ≠ []
assumes $k \geq 1$
assumes $\bigwedge a$. $a \in$ *set as* $\Longrightarrow a < n$

shows *prob-space.variance* (*measure-pmf* (*pmf-of-set* $\{(u, v).\ v < \textit{count-list as}$ $u\}$))
($\lambda a.$ *real* (*length as*) $*$ *real* (*Suc* (*snd a*) ^ *k* $-$ *snd a* ^ *k*))
$\le$ (*real-of-rat* (*F k as*))^2 $*$ *real k* $*$ *real n powr* ($1 - 1$ / *real k*)
⟨*proof*⟩

theorem *fk-alg-sketch*:
fixes ε :: *rat*
assumes $k \ge 1$
assumes $\delta > 0$
assumes $\bigwedge x.\ x \in \textit{set } xs \Longrightarrow x < n$
assumes $xs \neq []$
defines *sketch* $\equiv$ *fold* (λx *state*. *state* $\ggg$ *fk-update x*) *xs* (*fk-init k* δ ε *n*)
defines $s_1 \equiv$ *nat* $\lceil 3*\textit{real } k*(\textit{real } n)\ \textit{powr}\ (1-1/\ \textit{real } k)/\ (\textit{real-of-rat } \delta)^2\rceil$
defines $s_2 \equiv$ *nat* $\lceil -(18 * \ln (\textit{real-of-rat } \varepsilon))\rceil$
shows *sketch* = *map-pmf* ($\lambda x.$ (s_1,s_2,k,*length xs*, x))
(*prod-pmf* ($\{0..<s_1\} \times \{0..<s_2\}$) ($\lambda$-. *pmf-of-set* $\{(u,v).\ v < \textit{count-list xs } u\}$))
⟨*proof*⟩

lemma *fk-alg-correct*:
assumes $k \ge 1$
assumes $\varepsilon \in \{0<..<1\}$
assumes $\delta > 0$
assumes $\bigwedge a.\ a \in \textit{set as} \Longrightarrow a < n$
defines $M \equiv$ *fold* (λa *state*. *state* $\ggg$ *fk-update a*) *as* (*fk-init k* δ ε *n*) $\ggg$ *fk-result*
shows $\mathcal{P}(\omega$ *in measure-pmf M*. $|\omega - F\ k\ as| \le \delta * F\ k\ as) \ge 1 -$ *of-rat* ε
⟨*proof*⟩

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

definition *encode-state* **where**
encode-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 $\bigwedge a.\ a \in set\ as \Longrightarrow a < n$
  assumes $as \neq []$
  defines $M \equiv fold\ (\lambda a\ state.\ state \ggg fk\text{-}update\ a)\ as\ (fk\text{-}init\ k\ \delta\ \varepsilon\ n)$
  shows $AE\ \omega\ in\ M.\ bit\text{-}count\ (encode\text{-}state\ \omega) \leq fk\text{-}space\text{-}usage\ (k,\ n,\ length\ as,\ \varepsilon,\ \delta)$ (**is** $AE\ \omega\ in\ M.\ (\text{-} \ \leq\ ?rhs)$)
$\langle proof \rangle$

lemma *fk-asympotic-space-complexity*:
  $fk\text{-}space\text{-}usage \in$
  $O[at\text{-}top \times_F at\text{-}top \times_F at\text{-}top \times_F at\text{-}right\ (0::rat) \times_F at\text{-}right\ (0::rat)](\lambda\ (k,\ n,\ m,\ \varepsilon,\ \delta).$
  $real\ k * (real\ n)\ powr\ (1-1/\ real\ k)\ /\ (of\text{-}rat\ \delta)^2 * (ln\ (1\ /\ of\text{-}rat\ \varepsilon)) * (ln\ (real\ n) + ln\ (real\ m)))$
  (**is** $\text{-} \in O[?F](?rhs)$)
$\langle proof \rangle$

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 argument about medians we only want 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:

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

Now we can describe the estimate the algorithm obtains:

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

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

We want to show that

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

We show the result by investigating the two cases $F_0 \geq t$ and $F_0 < t$ seperately.

A.1 Case $F_0 \geq t$

Let us introduce:

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

These definitions correspond to the H, R but with a few minor modifications. We compute H^* as a multiset, this means we are keeping track of the multiplicities of its elements. Note that by definition: $|H^*| = |A|$. Similarly the operation $rank_t^{\#}$ obtains the rank-t element of the multiset (taking multiplicities into account). We also avoid the 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 $|R^* - F_0|$, $|R^* - R|$ as opposed to $|R - F_0|$ directly. In particular, what we plan to show is that:

$$\begin{aligned} \delta' &:= \frac{3}{4}\delta && (1) \\ P\left(|R^* - F_0| > \delta' F_0\right) &\leq \frac{2}{9}, \text{ and} && (2) \\ P\left(|R^* - F_0| \leq \delta' F_0 \wedge |R - R^*| > \frac{\delta}{4} F_0\right) &\leq \frac{1}{9} && (3) \end{aligned}$$

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:

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

For the verification of Equation 2 let us introduce:

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

then we observe that $\operatorname{rank}_t^{\#}(H^*) < u$ if $Q(u) \geq t$ and $\operatorname{rank}_t^{\#}(H^*) \geq v$ 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 we are taking multiplicities into account, when computing the rank t element.

Alternatively, we could also 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] Usingineariy of expectation and Bienaymé's identity, we can conclude that $\operatorname{Var} Q(u) \leq \operatorname{E} Q(u) = |A|up^{-1} = F_0 up^{-1}$ for $u \in \{0,\ldots,p\}$.

For $v = \left\lfloor \frac{tp}{(1-\delta')F_0} \right\rfloor$ we have

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

and thus we can conclude using Tchebyshev's inequality:

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

Similarly for $u = \left\lceil \frac{tp}{(1+\delta')F_0} \right\rceil$ we have

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

and thus we can conclude using Tchebyshev's inequality:

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

---

[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.

To verfiy Equation 3 we note that

$$\mathrm{rank}_t(H) = \lfloor \mathrm{rank}_t^{\#}(H^*) \rfloor_r \tag{7}$$

if there are no collisions, induced by the application of $\lfloor h(\cdot) \rfloor_r$ on the elements of A. If we are even more careful, we note that the equation would remain true, as long as there are no collision within the smallest t elements of H^*. Because we need to show Equation 3 only in the case where $R^* \geq (1-\delta')F_0$, i.e., when $\mathrm{rank}_t^{\#}(H^*) \leq v$, it is enough to bound the probability of a collision in the range $[0; v]$. Moreover Equation 7 implies $|\mathrm{rank}_t(H) - \mathrm{rank}_t^{\#}(H^*)| \leq \max(\mathrm{rank}_t^{\#}(H^*), \mathrm{rank}_t(H))2^{-r}$ from which it is possible to derive $|R^* - R| \leq \frac{\delta}{4} F_0$. Another observation we want to make 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 we have choosen $p \geq 18$, we can bound the probability that h is not injective 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 be able to show Equation 3.

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

Which shows that Equation 3 is true and using Equation 5 and 6 we can verify Equation 2, which means with the reasoning in Equation 4 we can confirm:

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

In the following subsection, we will confirm 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 $R = |H|$. We want 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, we rely on the fact that h is not injective with probability at $\frac{1}{18}$.

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

Which concludes the proof. $\square$