

Formalization of Randomized Approximation Algorithms for Frequency Moments

Emin Karayel

January 12, 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 space usage. This includes the problem of estimating the cardinality of the stream elements—the zeroth frequency moment. But, also higher order frequency moments that provide information about the skew of the data stream, which is for example critical information for parallel processing. The frequency moment of a data stream $a_1, \dots, a_m \in U$ can be defined as $F_k := \sum_{u \in U} C(u, a)^k$ where $C(u, a)$ is the count of occurrences 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	2
2	Field	11
3	Float	14
4	Lists	22
5	Frequency Moments	22

6 Primes	23
7 Multisets	25
8 Probability Spaces	28
9 Median	36
10 Order Statistics	48
11 Counting Polynomials	57
11.1 Interpolation Polynomials	60
12 Indexed Products of Probability Mass Functions	68
13 Universal Hash Families	80
14 Universal Hash Family for $\{0.. < p\}$	85
14.1 Encoding	90
15 Landau Symbols	92
16 Frequency Moment 0	97
17 Partitions	135
18 Frequency Moment 2	141
19 Frequency Moment k	161
A Informal proof of correctness for the F_0 algorithm	187
A.1 Case $F_0 \geq t$	188
A.2 Case $F_0 < t$	191

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 \rightarrow *bool list*

definition *is-encoding* :: 'a *encoding* \Rightarrow *bool*
where *is-encoding* *f* = (\forall *x y*. *is-prefix* (*f x*) (*f y*) \longrightarrow *x = y*)

lemma *encoding-imp-inj*:
assumes *is-encoding f*
shows *inj-on f* (*dom f*)
apply (*rule inj-onI*)
using *assms* **by** (*simp add:is-encoding-def, force*)

definition *decode* **where**
decode f t = (
 if ($\exists !z$. *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 -
have *a*: $\bigwedge y$. *is-prefix* (*f y*) (Some (*r@r1*)) \Longrightarrow *y = x*
proof -
fix *y*
assume *is-prefix* (*f y*) (Some (*r@r1*))
then obtain *u* **where** *u-1*: *f y* = Some *u* *prefix u* (*r@r1*)
by (*metis is-prefix.elims*(1) *option.sel*)
hence *prefix u r* \vee *prefix r u*
using *prefix-def prefix-same-cases* **by** *blast*
hence *is-prefix* (*f y*) (*f x*) \vee *is-prefix* (*f x*) (*f y*)
using *u-1 assms*(2) **by** *simp*
thus *y = x*
using *assms*(1) **apply** (*simp add:is-encoding-def*) **by** *blast*
qed
have *b*:*is-prefix* (*f x*) (Some (*r@r1*))
using *assms*(2) **by** *simp*
have *c*: $\exists !z$. *is-prefix* (*f z*) (Some (*r@r1*))
using *a b* **by** *auto*
have *d*:(*THE z*. *is-prefix* (*f z*) (Some (*r@r1*))) = *x*
using *a b* **by** *blast*
show *decode f* (*r@r1*) = (*x*,*r1*)
using *c d assms*(2) **by** (*simp add: decode-def*)
qed

```

lemma decode-elim-2:
  assumes is-encoding f
  assumes  $x \in \text{dom } f$ 
  shows  $\text{decode } f \text{ (the } (f \ x) @ r1) = (x, r1)$ 
  using assms decode-elim by fastforce

lemma snd-decode-suffix:
  suffix (snd (decode f t)) t
proof (cases  $\exists! z. \text{is-prefix } (f \ z) \text{ (Some } t)$ )
  case True
  then obtain z where is-prefix  $(f \ z) \text{ (Some } t)$  by blast
  hence  $(\text{THE } z. \text{is-prefix } (f \ z) \text{ (Some } t)) = z$  using True by blast
  thus ?thesis using True by (simp add: decode-def suffix-drop)
next
  case False
  then show ?thesis by (simp add: decode-def)
qed

lemma snd-decode-len:
  assumes  $\text{decode } f \ t = (u, v)$ 
  shows  $\text{length } v \leq \text{length } t$ 
  using snd-decode-suffix assms suffix-length-le
  by (metis snd-conv)

lemma encoding-by-witness:
  assumes  $\bigwedge x \ y. x \in \text{dom } f \implies g \text{ (the } (f \ x) @ y) = (x, y)$ 
  shows is-encoding f
proof –
  have  $\bigwedge x \ y. \text{is-prefix } (f \ x) \text{ (} f \ y) \implies x = y$ 
proof –
  fix x y
  assume a:is-prefix  $(f \ x) \text{ (} f \ y)$ 
  then obtain d where d-def:the  $(f \ x) @ d = \text{the } (f \ y)$ 
  apply (case-tac  $[\!] f \ x, \text{case-tac } [\!] f \ y, \text{simp, simp, simp, simp}$ )
  by (metis prefixE)
  have  $x \in \text{dom } f$  using a apply (simp add: dom-def del: not-None-eq)
  by (metis is-prefix.simps(2) a)
  hence  $g \text{ (the } (f \ y)) = (x, d)$  using assms by (simp add: d-def[symmetric])
  moreover have  $y \in \text{dom } f$  using a apply (simp add: dom-def del: not-None-eq)
  by (metis is-prefix.simps(3) a)
  hence  $g \text{ (the } (f \ y)) = (y, [])$  using assms [where  $y = []$ ] by simp
  ultimately show  $x = y$  by simp
qed
thus ?thesis by (simp add: is-encoding-def)
qed

fun bit-count where
  bit-count None =  $\infty$  |

```

$bit_count\ (Some\ x) = ereal\ (length\ x)$

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

lemma *bit-count-append*: $bit_count\ (x1@_Sx2) = bit_count\ x1 + bit_count\ x2$
by (*cases* *x1*, *simp*, *cases* *x2*, *simp*, *simp*)

Encodings for lists

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

function *decode-list* :: (*'a* \Rightarrow *bool list option*) \Rightarrow *bool list*
 \Rightarrow *'a list* \times *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*

by *pat-completeness auto*

termination

apply (*relation measure* ($\lambda(-,x). length\ x$)
by (*simp*+, *metis le-imp-less-Suc snd-decode-len*)

lemma *list-encoding-dom*:

assumes *set* *l* $\subseteq dom\ f$
shows *l* $\in dom\ (list_S\ f)$
using *assms* **apply** (*induction* *l*, *simp add:dom-def*, *simp*) **by** *fastforce*

lemma *list-bit-count*:

$bit_count\ (list_S\ f\ xs) = (\sum x \leftarrow xs. bit_count\ (f\ x) + 1) + 1$
apply (*induction* *xs*, *simp*, *simp add:bit-count-append*)
by (*metis add.commute add.left-commute one-ereal-def*)

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 –

have *a:sum-list* (*map* ($\lambda-. (a+1)$) *xs*) = *length* *xs* * (*a*+1)
apply (*induction* *xs*, *simp*)
by (*simp*, *subst plus-ereal.simps*(1)[*symmetric*], *subst ereal-left-distrib*, *simp*+))

have *b*: $\bigwedge x. x \in set\ xs \Longrightarrow bit_count\ (f\ x) + 1 \leq a+1$
using *assms add-right-mono* **by** *blast*

```

show ?thesis
  using assms a b sum-list-mono[where  $g=\lambda\cdot. a+1$  and  $f=\lambda x. \text{bit-count } (f$ 
 $x)+1$  and  $xs=xs$ ]
  by (simp add:list-bit-count ereal-add-le-add-iff2)
qed

```

```

lemma list-bit-count-estI:
  assumes  $\bigwedge x. x \in \text{set } xs \implies \text{bit-count } (f x) \leq a$ 
  assumes  $\text{ereal } (\text{real } (\text{length } xs)) * (a+1) + 1 \leq h$ 
  shows  $\text{bit-count } (\text{list}_S f xs) \leq h$ 
  using list-bit-count-est[OF assms(1)] assms(2) order-trans by fastforce

```

```

lemma list-encoding-aux:
  assumes is-encoding f
  shows  $x \in \text{dom } (\text{list}_S f) \implies \text{decode-list } f (\text{the } (\text{list}_S f x) @ y) = (x, y)$ 
proof (induction x)
  case Nil
  then show ?case by simp
next
  case (Cons a x)
  then show ?case
    apply (cases f a, simp add:dom-def)
    apply (cases list_S f x, simp add:dom-def)
    using assms by (simp add: dom-def decode-elim)
qed

```

```

lemma list-encoding:
  assumes is-encoding f
  shows is-encoding (list_S f)
  by (metis encoding-by-witness[where  $g=\text{decode-list } f$ ] list-encoding-aux assms)

```

Encoding for natural numbers

```

fun nat-encoding-aux :: nat  $\Rightarrow$  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  $\Rightarrow$  nat  $\times$  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)

```

```

    by (induction x rule:nat-encoding-aux.induct, simp, simp add:mult.commute)

lemma nat-encoding:
  shows is-encoding  $N_S$ 
  by (rule encoding-by-witness[where g=decode-nat], simp add:nat-encoding-aux)

lemma nat-bit-count:
  bit-count ( $N_S$  n)  $\leq 2 * \log 2$  (real n+1) + 1
proof (induction n rule:nat-encoding-aux.induct)
  case 1
  then show ?case by simp
next
  case (2 n)
  have  $\log 2$  2 +  $\log 2$  (1 + real (n div 2))  $\leq \log 2$  (2 + real n)
  apply (subst log-mult[symmetric], simp, simp, simp)
  by (subst log-le-cancel-iff, simp+)
  hence 1 + 2 *  $\log 2$  (1 + real (n div 2)) + 1  $\leq 2 * \log 2$  (2 + real n)
  by simp
  thus ?case using 2 by (simp add:add.commute)
qed

lemma nat-bit-count-est:
  assumes  $n \leq m$ 
  shows bit-count ( $N_S$  n)  $\leq 2 * \log 2$  (1+real m) + 1
proof -
  have  $2 * \log 2$  (1 + real n) + 1  $\leq 2 * \log 2$  (1+real m) + 1
  using assms by simp
  thus ?thesis
  by (metis nat-bit-count le-ereal-le add.commute)
qed

Encoding for integers

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

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

lemma int-encoding: is-encoding  $I_S$ 
  apply (rule encoding-by-witness[where g=decode-int])
  by (simp add:nat-encoding-aux)

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

```

```

proof -
  have  $a: \neg(0 \leq x) \implies 1 + 2 * \log 2 (- \text{real-of-int } x) \leq 1 + 2 * \log 2 (1 - \text{real-of-int } x)$ 
  by simp
  show ?thesis
  apply (cases  $x \geq 0$ )
    using nat-bit-count[where  $n=\text{nat } x$ ] apply (simp add: bit-count-append add.commute)
    using nat-bit-count[where  $n=\text{nat } (-x-1)$ ] apply (simp add: bit-count-append add.commute)
    using a order-trans by blast
qed

```

```

lemma int-bit-count-est:
  assumes  $\text{abs } n \leq m$ 
  shows  $\text{bit-count } (I_S \ n) \leq 2 * \log 2 (m+1) + 2$ 
proof -
  have  $2 * \log 2 (\text{abs } n+1) + 2 \leq 2 * \log 2 (m+1) + 2$  using assms by simp
  thus ?thesis using assms le-ereal-le int-bit-count by blast
qed

```

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:
   $x \in \text{dom } (e1 \times_S e2) = (\text{fst } x \in \text{dom } e1 \wedge \text{snd } x \in \text{dom } e2)$ 
  apply (case-tac [!] e1 (fst x))
  apply (case-tac [!] e2 (snd x))
  by (simp add: dom-def del: not-None-eq)+

```

```

lemma prod-encoding:
  assumes is-encoding e1
  assumes is-encoding e2
  shows is-encoding (encode-prod e1 e2)
proof (rule encoding-by-witness[where  $g=\text{decode-prod } e1 \ e2$ ])
  fix x y
  assume  $a: x \in \text{dom } (e1 \times_S e2)$ 

  have  $b: e1 (\text{fst } x) = \text{Some } (\text{the } (e1 (\text{fst } x)))$ 
    by (metis a prod-encoding-dom domIff option.exhaust-sel)
  have  $c: e2 (\text{snd } x) = \text{Some } (\text{the } (e2 (\text{snd } x)))$ 

```



```

    by (metis a prod-encoding-dom domIff option.exhaust-sel)

show decode-prod e1 e2 (the ((e1 ×S e2) x) @ y) = (x, y)
  apply (simp, subst b, subst c)
  apply simp
  using assms b c by (simp add: decode-elim)
qed

lemma prod-bit-count:
  bit-count ((e1 ×S e2) (x1, x2)) = bit-count (e1 x1) + bit-count (e2 x2)
  by (simp add: bit-count-append)

lemma prod-bit-count-2:
  bit-count ((e1 ×S e2) x) = bit-count (e1 (fst x)) + bit-count (e2 (snd x))
  by (simp add: bit-count-append)

Encoding for dependent sums

fun encode-dependent-sum :: 'a encoding ⇒ ('a ⇒ 'b encoding) ⇒ ('a × 'b) encoding
  (infixr ×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 ∧x. is-encoding (e2 x)
  shows is-encoding (encode-dependent-sum e1 e2)
proof -
  define d where d = (λx0.
    let (r1, x1) = decode e1 x0 in
    let (r2, x2) = decode (e2 r1) x1 in ((r1, r2), x2))

  have a: ∧x. x ∈ dom (e1 ×D e2) ⇒ fst x ∈ dom e1
    apply (simp add: dom-def del: not-None-eq)
    using append-encoding.simps by metis
  have b: ∧x. x ∈ dom (e1 ×D e2) ⇒ snd x ∈ dom (e2 (fst x))
    apply (simp add: dom-def del: not-None-eq)
    using append-encoding.simps by metis
  have c: ∧x. x ∈ dom (e1 ×D e2) ⇒ e1 (fst x) = Some (the (e1 (fst x)))
    using a by (simp add: domIff)
  have d: ∧x. x ∈ dom (e1 ×D e2) ⇒ e2 (fst x) (snd x) = Some (the (e2 (fst x) (snd x)))
    using b by (simp add: domIff)
  show ?thesis
    apply (rule encoding-by-witness[where g=d])
    apply (simp add: d-def, subst c, simp, subst d, simp)
    using assms a b by (simp add: d-def decode-elim-2)
qed

lemma dependent-bit-count:

```

$bit_count ((e_1 \times_D e_2) (x_1, x_2)) = bit_count (e_1 x_1) + bit_count (e_2 x_1 x_2)$
by (*simp add: bit-count-append*)

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 ($\lambda x. if P x then f (g x) else None$)*
using *assms* **by** (*simp add: inj-onD is-encoding-def*)

Encoding for extensional maps defined on an enumerable set.

definition *encode-extensional* :: '*a* list \Rightarrow '*b* encoding \Rightarrow ('*a* \Rightarrow '*b*) encoding (**infixr** \rightarrow_S 65) **where**
encode-extensional xs e f = (
if *f* \in *extensional (set xs)* *then*
list_S e (map f xs)
else
None)

lemma *encode-extensional*:
assumes *is-encoding e*
shows *is-encoding ($\lambda x. (xs \rightarrow_S e) x$)*
apply (*simp add: encode-extensional-def*)
apply (*rule encoding-compose[where f=list_S e]*)
apply (*metis list-encoding assms*)
apply (*rule inj-onI, simp*)
using *extensionalityI* **by** *fastforce*

lemma *extensional-bit-count*:
assumes *f* \in *extensional (set xs)*
shows *bit-count ((xs \rightarrow_S e) f) = ($\sum x \leftarrow xs. bit_count (e (f x)) + 1$) + 1*
using *assms*
by (*simp add: encode-extensional-def list-bit-count comp-def*)

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 ($\lambda S. set_S e S$)*
apply *simp*
apply (*rule encoding-compose[where f=list_S e]*)
apply (*metis assms list-encoding*)
apply (*rule inj-onI, simp*)
by (*metis sorted-list-of-set.set-sorted-key-list-of-set*)

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

```

shows  $\text{bit-count } (\text{set}_S e S) = (\sum x \in S. \text{bit-count } (e x) + 1) + 1$ 
using assms sorted-list-of-set
by (simp add:list-bit-count sum-list-distinct-conv-sum-set)

lemma set-bit-count-est:
  assumes finite S
  assumes  $\text{card } S \leq m$ 
  assumes  $0 \leq a$ 
  assumes  $\bigwedge x. x \in S \implies \text{bit-count } (f x) \leq a$ 
  shows  $\text{bit-count } (\text{set}_S f S) \leq \text{ereal } (\text{real } m) * (a+1) + 1$ 
proof –
  have  $\text{bit-count } (\text{set}_S f S) \leq \text{ereal } (\text{length } (\text{sorted-list-of-set } S)) * (a+1) + 1$ 
    using assms(4) assms(1) list-bit-count-est[where xs=sorted-list-of-set S] by
simp
  also have  $\dots \leq \text{ereal } (\text{real } m) * (a+1) + 1$ 
    apply (rule add-mono)
    apply (rule ereal-mult-right-mono)
    using assms by simp+
  finally show ?thesis by simp
qed

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 ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{int set}$  where
  zfact-embed  $p\ k = \text{Idl}_{\mathbb{Z}} \{ \text{int } p \} +>_{\mathbb{Z}} (\text{int } k)$ 

```

```

lemma zfact-embed-ran:
  assumes  $p > 0$ 
  shows  $\text{zfact-embed } p \text{ ` } \{0..<p\} = \text{carrier } (ZFact\ p)$ 
proof –
  have  $\text{zfact-embed } p \text{ ` } \{0..<p\} \subseteq \text{carrier } (ZFact\ p)$ 

```

```

proof (rule subsetI)
  fix x
  assume  $x \in \text{zfact-embed } p \text{ ' } \{0..<p\}$ 
  then obtain m where m-def:  $\text{zfact-embed } p \ m = x$  by blast
  have  $\text{zfact-embed } p \ m \in \text{carrier } (\text{ZFact } p)$ 
    by (simp add: ZFact-def ZFact-defs(2) int.a-rcosetsI zfact-embed-def)
  thus  $x \in \text{carrier } (\text{ZFact } p)$  using m-def by auto
qed
moreover have  $\text{carrier } (\text{ZFact } p) \subseteq \text{zfact-embed } p \text{ ' } \{0..<p\}$ 
proof (rule subsetI)
  define I where  $I = \text{Idl}_{\mathbb{Z}} \{ \text{int } p \}$ 
  fix x
  have coset-elim:  $\bigwedge x \ R \ I. \ x \in \text{a-rcosets}_R \ I \implies (\exists y. \ x = I +>_R y)$ 
    using assms apply (simp add: FactRing-simps) by blast
  assume a:  $x \in \text{carrier } (\text{ZFact } (\text{int } p))$ 
  obtain y' where y-0:  $x = I +>_{\mathbb{Z}} y'$ 
    apply (simp add: I-def carrier-def ZFact-def FactRing-simps)
    by (metis coset-elim FactRing-def ZFact-def a partial-object.select-convs(1))
  define y where  $y = y' \bmod p - y'$ 
  hence  $y \bmod p = 0$  by (simp add: mod-diff-left-eq)
  hence y-1:  $y \in I$  using I-def
    by (metis Idl-subset-eq-dvd int-Idl-subset-ideal mod-0-imp-dvd)
  have y-3:  $y + y' < p \wedge y + y' \geq 0$ 
    using y-def assms(1) by auto
  hence y-2:  $y \oplus_{\mathbb{Z}} y' < p \wedge y \oplus_{\mathbb{Z}} y' \geq 0$  using int-add-eq by presburger
  then have a3:  $I +>_{\mathbb{Z}} y' = I +>_{\mathbb{Z}} (y \oplus_{\mathbb{Z}} y')$  using I-def
    by (metis (no-types, lifting) y-1 UNIV-I abelian-group.a-coset-add-assoc
      int.Idl-subset-ideal' int.a-rcos-zero int.abelian-group-axioms
      int.cgenideal-eq-genideal int.cgenideal-ideal int.genideal-one int-carrier-eq)
  obtain w::nat where y-4:  $\text{int } w = y \oplus_{\mathbb{Z}} y'$ 
    using y-2 nonneg-int-cases by metis
  have  $x = I +>_{\mathbb{Z}} (\text{int } w)$  and  $w < p$  using y-2 a3 y-0 y-4 by presburger+
  thus  $x \in \text{zfact-embed } p \text{ ' } \{0..<p\}$  by (simp add: zfact-embed-def I-def)
qed
ultimately show ?thesis using order-antisym by auto
qed

```

```

lemma zfact-embed-inj:
  assumes  $p > 0$ 
  shows inj-on ( $\text{zfact-embed } p$ )  $\{0..<p\}$ 
proof
  fix x
  fix y
  assume a1:  $x \in \{0..<p\}$ 
  assume a2:  $y \in \{0..<p\}$ 
  assume  $\text{zfact-embed } p \ x = \text{zfact-embed } p \ y$ 
  hence  $\text{Idl}_{\mathbb{Z}} \{ \text{int } p \} +>_{\mathbb{Z}} \text{int } x = \text{Idl}_{\mathbb{Z}} \{ \text{int } p \} +>_{\mathbb{Z}} \text{int } y$ 
    by (simp add: zfact-embed-def)
  hence  $\text{int } x \ominus_{\mathbb{Z}} \text{int } y \in \text{Idl}_{\mathbb{Z}} \{ \text{int } p \}$ 

```

```

    using ring.quotient-eq-iff-same-a-r-cos
    by (metis UNIV-I int.cgenideal-eq-genideal int.cgenideal-ideal int.ring-axioms
int-carrier-eq)
    hence  $p \text{ dvd } (\text{int } x - \text{int } y)$  apply (simp add:int-Idl)
    using int-a-minus-eq by force
    thus  $x = y$  using a1 a2
    apply (simp)
    by (metis (full-types) cancel-comm-monoid-add-class.diff-cancel diff-less-mono2
dvd-0-right dvd-diff-commute less-imp-diff-less less-imp-of-nat-less linorder-neqE-nat
of-nat-0-less-iff zdvd-int-split zdvd-not-zless)
qed

```

```

lemma zfact-embed-bij:
  assumes  $p > 0$ 
  shows  $\text{bij\_betw } (\text{zfact\_embed } p) \{0..<p\} (\text{carrier } (\text{ZFact } p))$ 
  apply (rule bij-betw-imageI)
  using zfact-embed-inj zfact-embed-ran assms by auto

```

```

lemma zfact-card:
  assumes  $(p :: \text{nat}) > 0$ 
  shows  $\text{card } (\text{carrier } (\text{ZFact } (\text{int } p))) = p$ 
  apply (subst zfact-embed-ran[OF assms, symmetric])
  by (metis card-atLeastLessThan card-image diff-zero zfact-embed-inj[OF assms])

```

```

lemma zfact-finite:
  assumes  $(p :: \text{nat}) > 0$ 
  shows  $\text{finite } (\text{carrier } (\text{ZFact } (\text{int } p)))$ 
  using zfact-card
  by (metis assms card-ge-0-finite)

```

```

lemma finite-domains-are-fields:
  assumes domain R
  assumes finite (carrier R)
  shows field R
proof –
  interpret domain R using assms by auto
  have  $\text{Units } R = \text{carrier } R - \{0_R\}$ 
  proof
    have  $\text{Units } R \subseteq \text{carrier } R$  by (simp add:Units-def)
    moreover have  $0_R \notin \text{Units } R$ 
      by (meson assms(1) domain.zero-is-prime(1) primeE)
    ultimately show  $\text{Units } R \subseteq \text{carrier } R - \{0_R\}$  by blast
  next
    show  $\text{carrier } R - \{0_R\} \subseteq \text{Units } R$ 
  proof
    fix  $x$ 
    assume  $a::x \in \text{carrier } R - \{0_R\}$ 
    define  $f$  where  $f = (\lambda y. y \otimes_R x)$ 
    have inj-on  $f$  (carrier R) apply (simp add:inj-on-def f-def)

```

```

    by (metis DiffD1 DiffD2 a assms(1) domain.m-rcancel insertI1)
  hence card (carrier R) = card (f ` carrier R)
    by (metis card-image)
  moreover have f ` carrier R  $\subseteq$  carrier R
    apply (rule image-subsetI) apply (simp add:f-def) using a
    by (simp add: ring.ring-simprules(5))
  ultimately have f ` carrier R = carrier R using card-subset-eq assms(2) by
metis
  moreover have  $1_R \in \text{carrier } R$  by simp
  ultimately have  $\exists y \in \text{carrier } R. f y = 1_R$ 
    by (metis image-iff)
  then obtain y where y-carrier:  $y \in \text{carrier } R$  and y-left-inv:  $y \otimes_R x = 1_R$ 
    using f-def by blast
  hence y-right-inv:  $x \otimes_R y = 1_R$  using assms(1) a
    by (metis DiffD1 a cring.cring-simprules(14) domain.axioms(1))
  show  $x \in \text{Units } R$  using y-carrier y-left-inv y-right-inv
    by (metis DiffD1 a assms(1) cring.divides-one domain.axioms(1) factor-def)
qed
qed
then show field R by (simp add: assms(1) field.intro field-axioms.intro)
qed

lemma zfact-prime-is-field:
  assumes prime (p :: nat)
  shows field (ZFact (int p))
proof -
  define q where q = int p
  have finite (carrier (ZFact q)) using zfact-finite assms q-def prime-gt-0-nat by
blast
  moreover have domain (ZFact q) using ZFact-prime-is-domain assms q-def by
auto
  ultimately show ?thesis using finite-domains-are-fields q-def by blast
qed

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:
   $x \leq \text{round-down } \text{prec } x + 2^{\text{powr } (-\text{prec})}$ 
  using round-down-correct by (simp, meson diff-diff-eq diff-eq-diff-less-eq)

```

```

lemma truncate-down-ge:
   $x \leq \text{truncate-down } \text{prec } x + \text{abs } x * 2^{\text{powr } (-\text{prec})}$ 
proof (cases abs x > 0)
  case True
    have  $x \leq \text{round-down } (\text{int } \text{prec} - \lfloor \log 2 |x| \rfloor) x + 2^{\text{powr } (-\text{real-of-int}(\text{int } \text{prec} - \lfloor \log 2 |x| \rfloor))}$ 
    by (rule round-down-ge)
    also have  $\dots \leq \text{truncate-down } \text{prec } x + \text{abs } x * 2^{\text{powr } (-\text{prec})}$ 
    apply (rule add-mono)
    apply (simp add:truncate-down-def)
    apply (subst of-int-diff, simp)
    apply (subst powr-diff)
    apply (subst pos-divide-le-eq, simp)
    apply (subst mult.assoc)
    apply (subst powr-add[symmetric], simp)
    apply (subst le-log-iff[symmetric], simp, metis True)
    by auto
    finally show ?thesis by simp
  next
  case False
    then show ?thesis by simp
qed

```

```

lemma truncate-down-pos:
  assumes  $x \geq 0$ 
  shows  $x * (1 - 2^{\text{powr } (-\text{prec})}) \leq \text{truncate-down } \text{prec } x$ 
  apply (simp add:right-diff-distrib diff-le-eq)
  by (metis truncate-down-ge assms abs-of-nonneg)

```

```

lemma truncate-down-eq:
  assumes  $\text{truncate-down } r x = \text{truncate-down } r y$ 
  shows  $\text{abs } (x - y) \leq \max (\text{abs } x) (\text{abs } y) * 2^{\text{powr } (-\text{real } r)}$ 
proof -
  have  $x - y \leq \text{truncate-down } r x + \text{abs } x * 2^{\text{powr } (-\text{real } r)} - y$ 
  by (rule diff-right-mono, rule truncate-down-ge)
  also have  $\dots \leq y + \text{abs } x * 2^{\text{powr } (-\text{real } r)} - y$ 
  apply (rule diff-right-mono, rule add-mono)
  apply (subst assms(1), rule truncate-down-le, simp)
  by simp
  also have  $\dots \leq \text{abs } x * 2^{\text{powr } (-\text{real } r)}$  by simp
  also have  $\dots \leq \max (\text{abs } x) (\text{abs } y) * 2^{\text{powr } (-\text{real } r)}$  by simp
  finally have  $a:x - y \leq \max (\text{abs } x) (\text{abs } y) * 2^{\text{powr } (-\text{real } r)}$  by simp

  have  $y - x \leq \text{truncate-down } r y + \text{abs } y * 2^{\text{powr } (-\text{real } r)} - x$ 
  by (rule diff-right-mono, rule truncate-down-ge)
  also have  $\dots \leq x + \text{abs } y * 2^{\text{powr } (-\text{real } r)} - x$ 
  apply (rule diff-right-mono, rule add-mono)
  apply (subst assms(1)[symmetric], rule truncate-down-le, simp)
  by simp

```

also have $\dots \leq \text{abs } y * 2^{\text{powr } (-\text{real } r)}$ **by** *simp*
also have $\dots \leq \text{max } (\text{abs } x) (\text{abs } y) * 2^{\text{powr } (-\text{real } r)}$ **by** *simp*
finally have $b : y - x \leq \text{max } (\text{abs } x) (\text{abs } y) * 2^{\text{powr } (-\text{real } r)}$ **by** *simp*

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

definition *rat-of-float* :: *float* \Rightarrow *rat* **where**
rat-of-float *f* = *of-int* (*mantissa* *f*) *
 (if *exponent* *f* ≥ 0 then $2^{\text{nat } (\text{exponent } f)}$ else $1 / 2^{\text{nat } (-\text{exponent } f)}$))

lemma *real-of-rat-of-float*: *real-of-rat* (*rat-of-float* *x*) = *real-of-float* *x*
apply (*cases* *x*)
apply (*simp add:rat-of-float-def*)
apply (*rule conjI*)
apply (*metis* (*mono-tags*, *opaque-lifting*) *Float.rep-eq compute-real-of-float mantissa-exponent of-int-mult of-int-numeral of-int-power of-rat-of-int-eq*)
by (*metis* *Float.rep-eq Float-mantissa-exponent compute-real-of-float of-int-numeral of-int-power of-rat-divide of-rat-of-int-eq*)

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 –
have *a* : *inj* ($\lambda x. (\text{mantissa } x, \text{exponent } x)$)
proof (*rule injI*)
fix *x y*
assume (*mantissa* *x*, *exponent* *x*) = (*mantissa* *y*, *exponent* *y*)
hence *real-of-float* *x* = *real-of-float* *y*
by (*simp add:mantissa-exponent*)
thus *x* = *y*
by (*metis real-of-float-inverse*)
qed

have *is-encoding* ($\lambda f. \text{if True then } ((I_S \times_S I_S) (\text{mantissa } f, \text{exponent } f)) \text{ else None}$)
apply (*rule encoding-compose*[**where** *f*=(*I_S* \times_S *I_S*)])
apply (*metis prod-encoding int-encoding, simp*)
by (*metis a*)
moreover have *F_S* = ($\lambda f. \text{if } f \in \text{UNIV then } ((I_S \times_S I_S) (\text{mantissa } f, \text{exponent } f)) \text{ else None}$)
by (*rule ext, simp add:F_S-def*)
ultimately show *is-encoding* *F_S*
by *simp*
qed


```

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  $\leq$  -)
proof -
  define  $q$  where  $q = \lfloor x * 2^{powr (real\ r - real-of-int (\lfloor \log 2 |x| \rfloor))} \rfloor$ 

  have  $x > 0 \implies abs\ q \leq 2^{(r+1)}$ 
  proof -
    assume  $a: x > 0$ 

    have  $abs\ q = q$ 
    apply (rule abs-of-nonneg)
    apply (simp add: q-def)
    using  $a$  by simp
    also have  $\dots \leq x * 2^{powr (real\ r - real-of-int \lfloor \log 2 |x| \rfloor)}$ 
    apply (subst q-def)
    using of-int-floor-le by blast
    also have  $\dots = x * 2^{powr\ real-of-int (int\ r - \lfloor \log 2 |x| \rfloor)}$ 
    by auto
    also have  $\dots = 2^{powr (\log 2\ x + real-of-int (int\ r - \lfloor \log 2 |x| \rfloor))}$ 
    apply (simp add: powr-add)
    by (subst powr-log-cancel, simp, simp, simp add: a, simp)
    also have  $\dots \leq 2^{powr (real\ r + 1)}$ 
    apply (rule powr-mono)
    apply simp
    using  $a$  apply linarith
    by simp
    also have  $\dots = 2^{(r+1)}$ 
    by (subst powr-realpow[symmetric], simp, simp add: add.commute)
    finally show  $abs\ q \leq 2^{(r+1)}$ 
    by (metis of-int-le-iff of-int-numeral of-int-power)
  qed

  moreover have  $x < 0 \implies abs\ q \leq (2^{(r+1)})$ 
  proof -
    assume  $a: x < 0$ 
    have  $-(2^{(r+1)} + 1) = -(2^{powr (real\ r + 1)} + 1)$ 
    apply (subst powr-realpow[symmetric], simp)
    by (simp add: add.commute)
    also have  $\dots < -(2^{powr (\log 2 (-x) + (r - \lfloor \log 2 |x| \rfloor)) + 1})$ 
    apply (subst neg-less-iff-less)
    apply (rule add-strict-right-mono)
    apply (rule powr-less-mono)
    apply (simp)
    using  $a$  apply linarith
    by simp+
    also have  $\dots = x * 2^{powr (r - \lfloor \log 2 |x| \rfloor) - 1}$ 
    apply (simp add: powr-add)
    apply (subst powr-log-cancel, simp, simp, simp add: a)
    by simp

```

```

also have ... ≤ q
  by (simp add: q-def)
also have ... = - abs q
  apply (subst abs-of-neg)
  using a
  apply (simp add: mult-pos-neg2 q-def)
  by simp
finally have  $-(2^{(r+1)+1}) < - \text{abs } q$  using of-int-less-iff by fastforce
hence  $-(2^{(r+1)}) \leq - \text{abs } q$  by linarith
thus  $\text{abs } q \leq 2^{r+1}$  by linarith
qed

moreover have  $x = 0 \implies \text{abs } q \leq 2^{r+1}$ 
  by (simp add: q-def)
ultimately have  $\text{abs } q \leq 2^{r+1}$ 
  by fastforce
thus ?thesis using q-def by blast
qed

lemma suc-n-le-2-pow-n:
  fixes n :: nat
  shows  $n + 1 \leq 2^n$ 
  by (induction n, simp, simp)

lemma float-bit-count:
  fixes m :: int
  fixes e :: int
  defines  $f \equiv \text{float-of } (m * 2^{\text{powr } e})$ 
  shows  $\text{bit-count } (F_S f) \leq 4 + 2 * (\log 2 (|m| + 2) + \log 2 (|e| + 1))$ 
proof (cases m ≠ 0)
  case True
  have  $f = \text{Float } m \ e$ 
  by (simp add: f-def Float.abs-eq)
  moreover have  $f \neq 0$ :  $f \neq 0$  using True apply (simp add: f-def)
  by (metis Float.compute-is-float-zero Float.rep-eq is-float-zero.rep-eq real-of-float-inverse
    zero-float.rep-eq)
  ultimately obtain i :: nat where m-def:  $m = \text{mantissa } f * 2^i$  and e-def:  $e = \text{exponent } f - i$ 
  using denormalize-shift by blast

  have b:  $\text{abs } (\text{real-of-int } (\text{mantissa } f)) \geq 1$ 
  by (meson dual-order.refl f-ne-0 mantissa-noteq-0 of-int-leD)

  have c:  $2 * i \leq 2^i$ 
  apply (cases i > 0)
  using suc-n-le-2-pow-n[where n=i-1] apply simp
  apply (metis One-nat-def nat-mult-le-cancel-disj power-commutes power-minus-mult)
  by simp

```

```

have a:|real-of-int (mantissa f)| * (real i + 1) + real i ≤ |real-of-int (mantissa
f)| * 2 ^ i + 1
proof (cases i ≥ 1)
  case True
    have |real-of-int (mantissa f)| * (real i + 1) + real i = |real-of-int (mantissa
f)| * (real i + 1) + (real i - 1) + 1
    by simp
    also have ... ≤ |real-of-int (mantissa f)| * ((real i + 1) + (real i - 1)) + 1
    apply (subst (2) distrib-left)
    apply (rule add-mono)
    apply (rule add-mono, simp)
    apply (rule order-trans[where y=1* (real i - 1)], simp)
    apply (rule mult-right-mono, metis b)
    using True apply simp
    by simp
    also have ... = |real-of-int (mantissa f)| * (2 * real i) + 1
    by simp
    also have ... ≤ |real-of-int (mantissa f)| * 2 ^ i + 1
    apply (rule add-mono)
    apply (rule mult-left-mono)
    using c of-nat-mono apply fastforce
    by simp+
    finally show ?thesis by simp
next
  case False
    hence i = 0 by simp
    then show ?thesis by simp
qed

have bit-count (FS f) = bit-count (IS (mantissa f)) + bit-count (IS (exponent
f))
  by (simp add:f-def FS-def)
also have ... ≤
  ereal (2 * (log 2 (real-of-int (abs (mantissa f) + 1))) + 2) +
  ereal (2 * (log 2 (real-of-int (abs (exponent f) + 1))) + 2)
  by (rule add-mono, rule int-bit-count, rule int-bit-count)
also have ... = ereal (4 + 2 * (log 2 (real-of-int (abs (mantissa f)) + 1) +
log 2 (real-of-int (abs (e + i)) + 1)))
  by (simp add:algebra-simps e-def)
also have ... ≤ ereal (4 + 2 * (log 2 (real-of-int (abs (mantissa f)) + 1) +
log 2 (real i+1) +
log 2 (abs e + 1)))

apply (simp)
apply (subst distrib-left[symmetric])
apply (rule mult-left-mono)
apply (subst log-mult[symmetric], simp, simp, simp, simp)
apply (subst log-le-cancel-iff, simp, simp, simp)
apply (rule order-trans[where y= abs e + real i + 1], simp)
by (simp add:algebra-simps, simp)

```

```

also have ...  $\leq$  ereal (4 + 2 * (log 2 (real-of-int (abs (mantissa f * 2 ^ i)) +
2) +
  log 2 (abs e + 1)))
apply (simp)
apply (subst distrib-left[symmetric])
apply (rule mult-left-mono)
apply (subst log-mult[symmetric], simp, simp, simp, simp)
apply (subst log-le-cancel-iff, simp, simp, simp)
apply (subst abs-mult)
using a apply (simp add: distrib-right)
by simp
also have ... = ereal (4 + 2 * (log 2 (real-of-int (abs m) + 2) + log 2 (abs e +
1)))
by (simp add:m-def)
finally show ?thesis by (simp add:f-def[symmetric] bit-count-append del:N_S.simps
I_S.simps)
next
case False
hence float-of (m * 2 powr e) = Float 0 0
apply simp
using zero-float.abs-eq by linarith
then show ?thesis by (simp add:f-def F_S-def)
qed

lemma float-bit-count-zero:
  bit-count (F_S (float-of 0)) = 4
apply (subst zero-float.abs-eq[symmetric])
by (simp add:F_S-def)

lemma log-est: log 2 (real n + 1)  $\leq$  n
proof –
  have 1 + real n  $\leq$  2 powr (real n)
    using suc-n-le-2-pow-n apply (simp add: powr-realpow)
    by (metis numeral-power-eq-of-nat-cancel-iff of-nat-Suc of-nat-mono)
  thus ?thesis
    by (simp add: Transcendental.log-le-iff)
qed

lemma truncate-float-bit-count:
  bit-count (F_S (float-of (truncate-down r x)))  $\leq$  8 + 4 * real r + 2*log 2 (2 +
abs (log 2 (abs x)))
  (is ?lhs  $\leq$  ?rhs)
proof –
  define m where m =  $\lfloor x * 2^{\text{powr}(\text{real } r - \text{real-of-int } \lfloor \log 2 |x| \rfloor)} \rfloor$ 
  define e where e =  $\lfloor \log 2 |x| \rfloor - \text{int } r$ 

  have a: real r = real-of-int (int r) by simp
  have abs m + 2  $\leq 2^{\wedge}(r + 1) + 2^{\wedge}1$ 
    apply (rule add-mono)

```

```

    using truncate-mantissa-bound apply (simp add:m-def)
  by simp
also have ...  $\leq 2^{(r+2)}$ 
  by simp
finally have  $b: \text{abs } m + 2 \leq 2^{(r+2)}$  by simp
have  $c: \log 2 (\text{real-of-int } (|m| + 2)) \leq r+2$ 
  apply (subst Transcendental.log-le-iff, simp, simp)
  apply (subst powr-realpow, simp)
  by (metis of-int-le-iff of-int-numeral of-int-power b)

have  $\text{real-of-int } (\text{abs } e + 1) \leq \text{real-of-int } \lfloor \log 2 |x| \rfloor + \text{real-of-int } r + 1$ 
  by (simp add:e-def)
also have ...  $\leq 1 + \text{abs } (\log 2 (\text{abs } x)) + \text{real-of-int } r + 1$ 
  apply (simp)
  apply (subst abs-le-iff)
  by (rule conjI, linarith, linarith)
also have ...  $\leq (\text{real-of-int } r + 1) * (2 + \text{abs } (\log 2 (\text{abs } x)))$ 
  by (simp add:distrib-left distrib-right)
finally have  $d: \text{real-of-int } (\text{abs } e + 1) \leq (\text{real-of-int } r + 1) * (2 + \text{abs } (\log 2 (\text{abs } x)))$  by simp

have  $\log 2 (\text{real-of-int } (\text{abs } e + 1)) \leq \log 2 (\text{real-of-int } r + 1) + \log 2 (2 + \text{abs } (\log 2 (\text{abs } x)))$ 
  apply (subst log-mult[symmetric], simp, simp, simp, simp)
  using d by simp
also have ...  $\leq r + \log 2 (2 + \text{abs } (\log 2 (\text{abs } x)))$ 
  apply (rule add-mono)
  using log-est apply (simp add:add commute)
  by simp
finally have  $e: \log 2 (\text{real-of-int } (\text{abs } e + 1)) \leq r + \log 2 (2 + \text{abs } (\log 2 (\text{abs } x)))$  by simp

have  $?lhs \leq \text{ereal } (4 + (2 * \log 2 (\text{real-of-int } (|m| + 2))) + 2 * \log 2 (\text{real-of-int } (|e| + 1)))$ 
  apply (simp add:truncate-down-def round-down-def m-def[symmetric])
  apply (subst a, subst of-int-diff[symmetric], subst e-def[symmetric])
  using float-bit-count by simp
also have ...  $\leq \text{ereal } (4 + (2 * \text{real } (r+2)) + 2 * (r + \log 2 (2 + \text{abs } (\log 2 (\text{abs } x)))))$ 
  apply (subst ereal-less-eq)
  apply (rule add-mono, simp)
  apply (rule add-mono, rule mult-left-mono, metis c, simp)
  by (rule mult-left-mono, metis e, simp)
also have ... =  $?rhs$  by simp
finally show  $?thesis$  by simp
qed

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:
   $(x \in \text{set } xs) = (\text{count-list } xs \ x \geq 1)$ 
  by (induction xs, simp, simp)
```

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

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

```
lemma card-gr-1-iff:
  assumes finite S
  assumes  $x \in S$ 
  assumes  $y \in S$ 
  assumes  $x \neq y$ 
  shows  $\text{card } S > 1$ 
  using assms card-le-Suc0-iff-eq leI by auto
```

```
lemma count-list-ge-2-iff:
  assumes  $y < z$ 
  assumes  $z < \text{length } xs$ 
  assumes  $xs ! y = xs ! z$ 
  shows  $\text{count-list } xs (xs ! y) > 1$ 
  apply (subst count-list-card)
  apply (rule card-gr-1-iff[where  $x=y$  and  $y=z$ ])
  using assms by simp+
```

```
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**

$$F\ k\ xs = (\sum\ x \in \text{set } xs. (\text{rat-of-nat } (\text{count-list } xs\ x) \sim k))$$

lemma $F\text{-gr-0}$:

assumes $as \neq []$

shows $F\ k\ as > 0$

proof –

have $\text{rat-of-nat } 1 \leq \text{rat-of-nat } (\text{card } (\text{set } as))$

apply $(\text{rule } \text{of-nat-mono})$

using $\text{assms card-0-eq}[\text{where } A = \text{set } as]$

by $(\text{metis } \text{List.finite-set One-nat-def Suc-leI neq0-conv set-empty})$

also have $\dots \leq F\ k\ as$

apply $(\text{simp add: } F\text{-def})$

apply $(\text{rule } \text{sum-mono}[\text{where } K = \text{set } as \text{ and } f = \lambda \cdot. (1 :: \text{rat}), \text{simplified}])$

by $(\text{metis } \text{count-list-gr-1 of-nat-1 of-nat-power-le-of-nat-cancel-iff one-le-power})$

finally show $F\ k\ as > 0$ **by** simp

qed

end

6 Primes

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

theory Primes-Ext

imports $\text{Main HOL-Computational-Algebra.Primes Bertrands-Postulate.Bertrand}$

begin

lemma inf-primes : $\text{wf } ((\lambda n. (\text{Suc } n, n)) \cdot \{n. \neg (\text{prime } n)\})$ **(is** $\text{wf } ?S$)

proof $(\text{rule } \text{wfI-min})$

fix $x :: \text{nat}$

fix $Q :: \text{nat set}$

assume $a : x \in Q$

have $\exists z \in Q. \text{prime } z \vee \text{Suc } z \notin Q$

proof $(\text{cases } \exists z \in Q. \text{Suc } z \notin Q)$

case True

then show $?thesis$ **by** auto

next

case False

hence $b : \bigwedge z. z \in Q \implies \text{Suc } z \in Q$ **by** blast

have $c : \bigwedge k. k + x \in Q$

proof –

fix k

show $k + x \in Q$

by $(\text{induction } k, \text{simp add: } a, \text{simp add: } b)$

```

qed
show ?thesis
  apply (cases  $\exists z \in Q. \text{prime } z$ )
  apply blast
  by (metis add.commute less-natE bigger-prime c)
qed
thus  $\exists z \in Q. \forall y. (y, z) \in ?S \longrightarrow y \notin Q$  by blast
qed

function find-prime-above :: nat  $\Rightarrow$  nat where
  find-prime-above n = (if prime n then n else find-prime-above (Suc n))
  by auto
termination
  apply (relation ( $\lambda n. (\text{Suc } n, n)$ ) '  $\{n. \neg (\text{prime } n)\}$ )
  using inf-primes apply blast
  by simp

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

lemma find-prime-above-is-prime:
  prime (find-prime-above n)
  apply (induction n rule:find-prime-above.induct)
  by (simp add: find-prime-above.simps)+

lemma find-prime-above-min:
  find-prime-above n  $\geq$  2
  by (metis find-prime-above-is-prime prime-ge-2-nat)

lemma find-prime-above-lower-bound:
  find-prime-above n  $\geq$  n
  apply (induction n rule:find-prime-above.induct)
  by (metis find-prime-above.simps linorder-le-cases not-less-eq-eq)

lemma find-prime-above-upper-boundI:
  assumes prime m
  shows  $n \leq m \Longrightarrow \text{find-prime-above } n \leq m$ 
proof (induction n rule:find-prime-above.induct)
  case (1 n)
  have  $a: \neg \text{prime } n \Longrightarrow \text{Suc } n \leq m$ 
  by (metis assms 1.premis not-less-eq-eq le-antisym)
  show ?case using 1
  apply (cases prime n)
  apply (subst find-prime-above.simps)
  using assms(1) apply simp
  by (metis a find-prime-above.simps)
qed

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

```



```

proof (cases  $n \leq 1$ )
  case True
    have find-prime-above  $n \leq 2$ 
      apply (rule find-prime-above-upper-boundI, simp) using True by simp
    then show ?thesis using trans-le-add2 by blast
next
  case False
    hence  $a:n > 1$  by auto
    then obtain  $p$  where  $p\text{-bound}$ :  $p \in \{n < .. < 2*n\}$  and  $p\text{-prime}$ : prime  $p$ 
      using bertrand by metis
    have find-prime-above  $n \leq p$ 
      apply (rule find-prime-above-upper-boundI)
      apply (metis  $p\text{-prime}$ )
      using  $p\text{-bound}$  by simp
    thus ?thesis using  $p\text{-bound}$ 
      by (metis greaterThanLessThan-iff nat-le-iff-add nat-less-le trans-le-add1)
qed

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  $\bigwedge n \ M \ x. \ P \ M \implies \neg(x \in \# \ M) \implies n > 0 \implies P \ (M + \text{replicate-mset } n \ x)$ 
  shows  $P \ M$ 

```

proof (*induction size M arbitrary: M rule:nat-less-induct*)

```

  case 1
  show ?case
  proof (cases  $M = \{\#\}$ )
    case True
      then show ?thesis using assms by simp
  next
    case False
      then obtain  $x$  where  $x\text{-def}$ :  $x \in \# \ M$  using multiset-nonemptyE by auto
      define  $M1$  where  $M1 = M - \text{replicate-mset } (\text{count } M \ x) \ x$ 
      then have  $M\text{-def}$ :  $M = M1 + \text{replicate-mset } (\text{count } M \ x) \ x$ 
        by (metis count-le-replicate-mset-subset-eq dual-order.refl subset-mset.diff-add)
      have  $\text{size } M1 < \text{size } M$ 

```

```

    by (metis M-def x-def count-greater-zero-iff less-add-same-cancel1 size-replicate-mset
size-union)
  hence P M1 using 1 by blast
  then show P M
    apply (subst M-def, rule assms(2), simp)
    by (simp add: M1-def x-def count-eq-zero-iff[symmetric])+
  qed
qed

```

```

lemma prod-mset-conv:
  fixes f :: 'a ⇒ 'b::{comm-monoid-mult}
  shows prod-mset (image-mset f A) = prod (λx. f x ^ (count A x)) (set-mset A)
proof (induction A rule: disj-induct-mset)
  case 1
  then show ?case by simp
next
  case (2 n M x)
  moreover have count M x = 0 using 2 by (simp add: count-eq-zero-iff)
  moreover have ∧y. y ∈ set-mset M ⇒ y ≠ x using 2 by blast
  ultimately show ?case by (simp add: algebra-simps)
qed

```

```

lemma sum-collapse:
  fixes f :: 'a ⇒ 'b::{comm-monoid-add}
  assumes finite A
  assumes z ∈ A
  assumes ∧y. y ∈ A ⇒ y ≠ z ⇒ f y = 0
  shows sum f A = f z
  using sum.union-disjoint[where A=A- {z} and B={z} and g=f]
  by (simp add: assms sum.insert-if)

```

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 ⇒ 'b::{ring,semiring-1}
  shows sum-list (map f xs) = (∑ x ∈ set xs. of-nat (count-list xs x) * f x)
proof -
  define M where M = mset xs
  have sum-mset (image-mset f M) = (∑ x ∈ set-mset M. of-nat (count M x) * f
x)
  proof (induction M rule: disj-induct-mset)
    case 1
    then show ?case by simp
  next
    case (2 n M x)
    have a: ∧y. y ∈ set-mset M ⇒ y ≠ x using 2(2) by blast
    show ?case using 2 by (simp add: a count-eq-zero-iff[symmetric])
  qed
  moreover have ∧x. count-list xs x = count (mset xs) x

```

```

    by (induction xs, simp, simp)
  ultimately show ?thesis
    by (simp add:M-def sum-mset-sum-list[symmetric])
qed

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma sorted-list-of-multiset-image-commute:
  assumes mono f
  shows sorted-list-of-multiset (image-mset f M) = map f (sorted-list-of-multiset
M) (is ?A = ?B)
  apply (rule list-eq-iff, simp)
  apply (simp add:sorted-sorted-list-of-multiset)

```

```

apply (subst sorted-wrt-map)
by (metis (no-types, lifting) monoE sorted-sorted-list-of-multiset sorted-wrt-mono-rel
assms)

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) =  $\mathcal{P}(x \text{ in } M. x \in E)$ 
by (simp add: Collect-conj-eq inf-commute)

```

```

lemma set-comp-subsetI:  $(\bigwedge x. P x \implies f x \in B) \implies \{f x | x. P x\} \subseteq B$ 
by blast

```

```

lemma set-comp-cong:
assumes  $\bigwedge x. P x \implies f x = h (g x)$ 
shows  $\{f x | x. P x\} = h \text{ ` } \{g x | x. P x\}$ 
using assms by (simp add: setcompr-eq-image, auto)

```

```

lemma indep-sets-distr:
assumes  $f \in \text{measurable } M N$ 
assumes prob-space M
assumes prob-space.indep-sets M  $(\lambda i. (\lambda a. f \text{ ` } a \cap \text{space } M) \text{ ` } A i) I$ 
assumes  $\bigwedge i. i \in I \implies A i \subseteq \text{sets } N$ 
shows prob-space.indep-sets (distr M N f) A I
proof –
define F where  $F = (\lambda i. (\lambda a. f \text{ ` } a \cap \text{space } M) \text{ ` } A i)$ 
have indep-F: prob-space.indep-sets M F I
using F-def assms(3) by simp

```

```

have sets-A:  $\bigwedge i. i \in I \implies A i \subseteq \text{sets } N$ 
using assms(4) by blast

```

```

have indep-A:  $\bigwedge A' J. J \neq \{\} \implies J \subseteq I \implies \text{finite } J \implies$ 
 $\forall j \in J. A' j \in A j \implies \text{measure } (\text{distr } M N f) (\bigcap (A' \text{ ` } J)) = (\prod j \in J. \text{measure}$ 
 $(\text{distr } M N f) (A' j))$ 

```

```

proof –
fix A' J
assume a1:  $J \subseteq I$ 
assume a2: finite J
assume a3:  $J \neq \{\}$ 
assume a4:  $\forall j \in J. A' j \in A j$ 

```

```

define F' where F' = (λi. f - ' A' i ∩ space M)

have ⋂ (F' ' J) = f - ' (⋂ (A' ' J)) ∩ space M
  apply (rule order-antisym)
  apply (rule subsetI, simp add:F'-def a3)
  by (rule subsetI, simp add:F'-def a3)
moreover have ⋂ (A' ' J) ∈ sets N
  using a4 a1 sets-A
  by (metis a2 a3 sets.finite-INT subset-iff)
ultimately have r1: measure (distr M N f) (⋂ (A' ' J)) = measure M (⋂
(F' ' J))
  using assms(1) measure-distr by metis

have ⋂ j. j ∈ J ⇒ F' j ∈ F j
  using a4 F'-def F-def by blast
hence r2: measure M (⋂ (F' ' J)) = (⋂ j ∈ J. measure M (F' j))
  using indep-F prob-space.indep-setsD assms(2) a1 a2 a3 by metis

have ⋂ j. j ∈ J ⇒ F' j = f - ' A' j ∩ space M
  by (simp add:F'-def)
moreover have ⋂ j. j ∈ J ⇒ A' j ∈ sets N
  using a4 a1 sets-A by blast
ultimately have r3: ⋂ j. j ∈ J ⇒ measure M (F' j) = measure (distr M N
f) (A' j)
  using assms(1) measure-distr by metis

show measure (distr M N f) (⋂ (A' ' J)) = (⋂ j ∈ J. measure (distr M N f)
(A' j))
  using r1 r2 r3 by auto
qed

show ?thesis
  apply (rule prob-space.indep-setsI)
  using assms apply (simp add:prob-space.prob-space-distr)
  apply (simp add:sets-A)
  using indep-A by blast
qed

lemma indep-vars-distr:
  assumes f ∈ measurable M N
  assumes ⋂ i. i ∈ I ⇒ X' i ∈ measurable N (M' i)
  assumes prob-space.indep-vars M M' (λi. (X' i) ∘ f) I
  assumes prob-space M
  shows prob-space.indep-vars (distr M N f) M' X' I
proof -
  have b1: f ∈ space M → space N using assms(1) by (simp add:measurable-def)
  have a: ⋂ i. i ∈ I ⇒ {(X' i ∘ f) - ' A ∩ space M | A. A ∈ sets (M' i)} = (λa.
f - ' a ∩ space M) ' {X' i - ' A ∩ space N | A. A ∈ sets (M' i)}

```

```

    apply (rule set-comp-cong)
    apply (rule order-antisym, rule subsetI, simp) using b1 apply fast
    by (rule subsetI, simp)
  show ?thesis
using assms apply (simp add:prob-space.indep-vars-def2 prob-space.prob-space-distr)
  apply (rule indep-sets-distr)
  apply (simp add:a cong:prob-space.indep-sets-cong)
  apply (simp add:a cong:prob-space.indep-sets-cong)
  apply (simp add:a cong:prob-space.indep-sets-cong)
using assms(2) measurable-sets by blast
qed

```

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

lemma *make-ext*:

```

  assumes  $\bigwedge x. P\ x = P\ (\text{restrict } x\ I)$ 
  shows  $(\forall x \in \text{Pi } I\ A. P\ x) = (\forall x \in \text{PiE } I\ A. P\ x)$ 
  apply (simp add:PiE-def Pi-def)
  apply (rule order-antisym)
  apply (simp add:Pi-def)
  using assms by fastforce

```

lemma *PiE-reindex*:

```

  assumes inj-on  $f\ I$ 
  shows  $\text{PiE } I\ (A \circ f) = (\lambda a. \text{restrict } (a \circ f)\ I) \text{ ` } \text{PiE } (f \text{ ` } I)\ A$  (is ?lhs = ?f ` ?rhs)
proof -
  have ?lhs  $\subseteq$  ?f ` ?rhs
  proof (rule subsetI)
    fix x
    assume  $a: x \in \text{PiE } I\ (A \circ f)$ 
    define y where y-def:  $y = (\lambda k. \text{if } k \in f \text{ ` } I \text{ then } x\ (\text{the-inv-into } I\ f\ k) \text{ else undefined})$ 
    have  $b: y \in \text{PiE } (f \text{ ` } I)\ A$ 
    apply (rule PiE-I)
    using a apply (simp add:y-def PiE-iff)
    apply (metis imageE assms the-inv-into-f-eq)
    using a by (simp add:y-def PiE-iff extensional-def)
    have  $c: x = (\lambda a. \text{restrict } (a \circ f)\ I)\ y$ 
    apply (rule ext)
    using a apply (simp add:y-def PiE-iff)
    apply (rule conjI)
    using assms the-inv-into-f-eq
    apply (simp add: the-inv-into-f-eq)
    by (meson extensional-arb)
    show  $x \in ?f \text{ ` } ?rhs$  using b c by blast
  qed
  moreover have ?f ` ?rhs  $\subseteq$  ?lhs
  apply (rule image-subsetI)

```

by (simp add: Pi-def PiE-def)
ultimately show ?thesis by blast
qed

lemma (in prob-space) indep-sets-reindex:

assumes inj-on f I

shows indep-sets A (f ' I) = indep-sets (λi. A (f i)) I

proof -

have a: $\bigwedge J. J \subseteq I \implies (\prod j \in f ' J. g j) = (\prod j \in J. g (f j))$

by (metis assms prod.reindex-cong subset-inj-on)

have $\bigwedge J. J \subseteq I \implies (\prod_E i \in J. A (f i)) = (\lambda a. \text{restrict } (a \circ f) J) ' \text{PiE } (f ' J)$
A

apply (subst PiE-reindex[symmetric])

using assms inj-on-subset apply blast

by (simp add: comp-def)

hence b: $\bigwedge P J. J \subseteq I \implies (\bigwedge x. P x = P (\text{restrict } x J)) \implies (\forall A' \in \text{PiE } (f ' J)$
A. $P (A' \circ f)) = (\forall A' \in \prod_E i \in J. A (f i). P A')$
by (simp)

have c: $\bigwedge J. J \subseteq I \implies \text{finite } (f ' J) = \text{finite } J$

by (meson assms finite-image-iff inj-on-subset)

show ?thesis

apply (simp add: indep-sets-def all-subset-image a c)

apply (subst make-ext) apply (simp cong: restrict-cong)

apply (subst make-ext) apply (simp cong: restrict-cong)

by (simp add: b[symmetric])

qed

lemma (in prob-space) indep-vars-reindex:

assumes inj-on f I

assumes indep-vars M' X' (f ' I)

shows indep-vars (M' o f) (λk ω. X' (f k) ω) I

using assms by (simp add: indep-vars-def2 indep-sets-reindex)

lemma (in prob-space) variance-divide:

fixes f :: 'a ⇒ real

assumes integrable M f

shows variance (λω. f ω / r) = variance f / r²

apply (subst Bochner-Integration.integral-divide[OF assms(1)])

apply (subst diff-divide-distrib[symmetric])

using assms by (simp add: power2-eq-square algebra-simps)

lemma pmf-eq:

assumes $\bigwedge x. x \in \text{set-pmf } \Omega \implies (x \in P) = (x \in Q)$

shows measure (measure-pmf Ω) P = measure (measure-pmf Ω) Q

apply (rule measure-eq-AE)

```

    apply (subst AE-measure-pmf-iff)
  using assms by auto

lemma pmf-mono-1:
  assumes  $\bigwedge x. x \in P \implies x \in \text{set-pmf } \Omega \implies x \in Q$ 
  shows  $\text{measure } (\text{measure-pmf } \Omega) P \leq \text{measure } (\text{measure-pmf } \Omega) Q$ 
proof -
  have  $\text{measure } (\text{measure-pmf } \Omega) P = \text{measure } (\text{measure-pmf } \Omega) (P \cap \text{set-pmf } \Omega)$ 

    by (rule pmf-eq, simp)
  also have  $\dots \leq \text{measure } (\text{measure-pmf } \Omega) Q$ 
  apply (rule finite-measure.finite-measure-mono, simp)
    apply (rule subsetI) using assms apply blast
    by simp
  finally show ?thesis by simp
qed

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

lemma (in prob-space) real-prod-integrable:
  fixes f g :: 'a  $\Rightarrow$  real
  assumes [measurable]:  $f \in \text{borel-measurable } M$   $g \in \text{borel-measurable } M$ 
  assumes sq-int:  $\text{integrable } M (\lambda \omega. f \ \omega^2)$   $\text{integrable } M (\lambda \omega. g \ \omega^2)$ 
  shows  $\text{integrable } M (\lambda \omega. f \ \omega * g \ \omega)$ 
  unfolding integrable-iff-bounded
proof
  have  $(\int^+ \omega. \text{ennreal } (\text{norm } (f \ \omega * g \ \omega)) \ \partial M)^2 = (\int^+ \omega. \text{ennreal } |f \ \omega| * \text{ennreal } |g \ \omega| \ \partial M)^2$ 
    by (simp add: abs-mult ennreal-mult)
  also have  $\dots \leq (\int^+ \omega. \text{ennreal } |f \ \omega|^2 \ \partial M) * (\int^+ \omega. \text{ennreal } |g \ \omega|^2 \ \partial M)$ 
    apply (rule Cauchy-Schwarz-nn-integral) by auto
  also have  $\dots < \infty$ 
    using sq-int by (auto simp: integrable-iff-bounded ennreal-power ennreal-mult-less-top)
  finally have  $(\int^+ x. \text{ennreal } (\text{norm } (f \ x * g \ x)) \ \partial M)^2 < \infty$ 
    by simp
  thus  $(\int^+ x. \text{ennreal } (\text{norm } (f \ x * g \ x)) \ \partial M) < \infty$ 
    by (simp add: power-less-top-ennreal)
qed auto

lemma (in prob-space) covariance-eq:
  fixes f :: 'a  $\Rightarrow$  real
  assumes  $f \in \text{borel-measurable } M$   $g \in \text{borel-measurable } M$ 
  assumes  $\text{integrable } M (\lambda \omega. f \ \omega^2)$   $\text{integrable } M (\lambda \omega. g \ \omega^2)$ 
  shows  $\text{covariance } f g = \text{expectation } (\lambda \omega. f \ \omega * g \ \omega) - \text{expectation } f * \text{expectation } g$ 
proof -
  have  $\text{integrable } M f$  using square-integrable-imp-integrable assms by auto
  moreover have  $\text{integrable } M g$  using square-integrable-imp-integrable assms by

```



```

auto
ultimately show ?thesis
  using assms real-prod-integrable
  by (simp add:covariance-def algebra-simps prob-space)
qed

lemma (in prob-space) covar-integrable:
  fixes f g :: 'a  $\Rightarrow$  real
  assumes f  $\in$  borel-measurable M g  $\in$  borel-measurable M
  assumes integrable M ( $\lambda\omega. f \ \omega^{\wedge}2$ ) integrable M ( $\lambda\omega. g \ \omega^{\wedge}2$ )
  shows integrable M ( $\lambda\omega. (f \ \omega - \text{expectation } f) * (g \ \omega - \text{expectation } g)$ )
proof -
  have integrable M f using square-integrable-imp-integrable assms by auto
  moreover have integrable M g using square-integrable-imp-integrable assms by
auto
  ultimately show ?thesis using assms real-prod-integrable by (simp add: alge-
bra-simps)
qed

lemma (in prob-space) sum-square-int:
  fixes f :: 'b  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes finite I
  assumes  $\bigwedge i. i \in I \implies f \ i \in$  borel-measurable M
  assumes  $\bigwedge i. i \in I \implies$  integrable M ( $\lambda\omega. f \ i \ \omega^{\wedge}2$ )
  shows integrable M ( $\lambda\omega. (\sum i \in I. f \ i \ \omega)^2$ )
  apply (simp add:power2-eq-square sum-distrib-left sum-distrib-right)
  apply (rule Bochner-Integration.integrable-sum)
  apply (rule Bochner-Integration.integrable-sum)
  apply (rule real-prod-integrable)
  using assms by auto

lemma (in prob-space) var-sum-1:
  fixes f :: 'b  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes finite I
  assumes  $\bigwedge i. i \in I \implies f \ i \in$  borel-measurable M
  assumes  $\bigwedge i. i \in I \implies$  integrable M ( $\lambda\omega. f \ i \ \omega^{\wedge}2$ )
  shows
    variance ( $\lambda\omega. (\sum i \in I. f \ i \ \omega)$ ) = ( $\sum i \in I. (\sum j \in I. \text{covariance } (f \ i) (f \ j))$ )
  (is ?lhs = ?rhs)
proof -
  have a: $\bigwedge i \ j. i \in I \implies j \in I \implies$  integrable M ( $\lambda\omega. (f \ i \ \omega - \text{expectation } (f \ i)) * (f \ j \ \omega - \text{expectation } (f \ j))$ )
  using assms covar-integrable by simp
  have ?lhs = expectation ( $\lambda\omega. (\sum i \in I. f \ i \ \omega - \text{expectation } (f \ i))^2$ )
  apply (subst Bochner-Integration.integral-sum)
  apply (simp add: square-integrable-imp-integrable[OF assms(2) assms(3)])
  by (subst sum-subtractf[symmetric], simp)
  also have ... = expectation ( $\lambda\omega. (\sum i \in I. (\sum j \in I. (f \ i \ \omega - \text{expectation } (f \ i)) * (f \ j \ \omega - \text{expectation } (f \ j))))$ )

```

```

    apply (simp add: power2-eq-square sum-distrib-right sum-distrib-left)
    apply (rule Bochner-Integration.integral-cong, simp)
    apply (rule sum.cong, simp)+
    by (simp add:mult.commute)
  also have ... = ( $\sum i \in I. (\sum j \in I. \text{covariance } (f\ i) (f\ j))$ )
    using a by (simp add: Bochner-Integration.integral-sum covariance-def)
  finally show ?thesis by simp
qed

```

```

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

```

```

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 -
  have a:indep-var borel (( $\lambda t. t - \text{expectation } f$ )  $\circ$  f) borel (( $\lambda t. t - \text{expectation } g$ )  $\circ$  g)
    by (rule indep-var-compose[OF assms(3)], simp, simp)

  show ?thesis
    apply (simp add:covariance-def)
    apply (subst indep-var-lebesgue-integral)
    using a assms by (simp add:comp-def prob-space)+
qed

```

```

lemma (in prob-space) var-sum-2:
  fixes f :: 'b  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes finite I
  assumes  $\bigwedge i. i \in I \implies f\ i \in \text{borel-measurable } M$ 
  assumes  $\bigwedge i. i \in I \implies \text{integrable } M (\lambda \omega. f\ i\ \omega^{\wedge 2})$ 
  shows variance ( $\lambda \omega. (\sum i \in I. f\ i\ \omega)$ ) =
    ( $\sum i \in I. \text{variance } (f\ i)$ ) + ( $\sum i \in I. \sum j \in I - \{i\}. \text{covariance } (f\ i) (f\ j)$ )
  apply (subst var-sum-1[OF assms(1) assms(2) assms(3)], simp)
  apply (subst covar-self-eq[symmetric])
  apply (subst sum.distrib[symmetric])
  apply (rule sum.cong, simp)
  apply (subst sum.insert[symmetric], simp add:assms, simp)
  by (rule sum.cong, simp add:insert-absorb, simp)

```

```

lemma (in prob-space) var-sum-pairwise-indep:
  fixes f :: 'b  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes finite I
  assumes  $\bigwedge i. i \in I \implies f\ i \in \text{borel-measurable } M$ 

```

assumes $\bigwedge i. i \in I \implies \text{integrable } M \ (\lambda\omega. f \ i \ \omega^{\wedge 2})$
assumes $\bigwedge i \ j. i \in I \implies j \in I \implies i \neq j \implies \text{indep-var borel } (f \ i) \ \text{borel } (f \ j)$
shows $\text{variance } (\lambda\omega. (\sum i \in I. f \ i \ \omega)) = (\sum i \in I. \text{variance } (f \ i))$
proof –
have $\bigwedge i \ j. i \in I \implies j \in I - \{i\} \implies \text{covariance } (f \ i) \ (f \ j) = 0$
apply (rule covar-indep-eq-zero)
using *assms square-integrable-imp-integrable*[OF *assms*(2) *assms*(3)] **by** *auto*

hence $a: (\sum i \in I. \sum j \in I - \{i\}. \text{covariance } (f \ i) \ (f \ j)) = 0$
by *simp*

show ?thesis
by (*subst var-sum-2*[OF *assms*(1) *assms*(2) *assms*(3)], *simp*, *simp add:a*)
qed

lemma (in *prob-space*) *indep-var-from-indep-vars*:
assumes $i \neq j$
assumes $\text{indep-vars } (\lambda\cdot. M') \ f \ \{i, j\}$
shows $\text{indep-var } M' \ (f \ i) \ M' \ (f \ j)$
proof –
have $a: \text{inj } (\text{case-bool } i \ j) \text{ using } \text{assms}(1)$
by (*simp add: bool.case-eq-if inj-def*)
have $b: \text{range } (\text{case-bool } i \ j) = \{i, j\}$
by (*simp add: UNIV-bool insert-commute*)
have $c: \text{indep-vars } (\lambda\cdot. M') \ f \ (\text{range } (\text{case-bool } i \ j)) \text{ using } \text{assms}(2) \ b \text{ by } \text{simp}$

have $\text{True} = \text{indep-vars } (\lambda x. M') \ (\lambda x. f \ (\text{case-bool } i \ j \ x)) \ \text{UNIV}$
using *indep-vars-reindex*[OF *a c*]
by (*simp add: comp-def*)
also have $\dots = \text{indep-vars } (\lambda x. \text{case-bool } M' \ M' \ x) \ (\lambda x. \text{case-bool } (f \ i) \ (f \ j) \ x)$
UNIV
apply (rule *indep-vars-cong*, *simp*)
apply (*metis bool.case-distrib*)
by (*simp add: bool.case-eq-if*)
also have $\dots = ?thesis$
apply (*subst indep-var-def*) **by** *simp*
finally show ?thesis **by** *simp*
qed

lemma (in *prob-space*) *var-sum-pairwise-indep-2*:
fixes $f :: 'b \Rightarrow 'a \Rightarrow \text{real}$
assumes *finite I*
assumes $\bigwedge i. i \in I \implies f \ i \in \text{borel-measurable } M$
assumes $\bigwedge i. i \in I \implies \text{integrable } M \ (\lambda\omega. f \ i \ \omega^{\wedge 2})$
assumes $\bigwedge J. J \subseteq I \implies \text{card } J = 2 \implies \text{indep-vars } (\lambda \cdot. \text{borel}) \ f \ J$
shows $\text{variance } (\lambda\omega. (\sum i \in I. f \ i \ \omega)) = (\sum i \in I. \text{variance } (f \ i))$
apply (rule *var-sum-pairwise-indep*[OF *assms*(1) *assms*(2) *assms*(3)], *simp*,
simp)
apply (rule *indep-var-from-indep-vars*, *simp*)

```

by (rule assms(4), simp, simp)

lemma (in prob-space) var-sum-all-indep:
  fixes f :: 'b  $\Rightarrow$  'a  $\Rightarrow$  real
  assumes finite I
  assumes  $\bigwedge i. i \in I \implies f\ i \in \text{borel-measurable } M$ 
  assumes  $\bigwedge i. i \in I \implies \text{integrable } M\ (\lambda\omega. f\ i\ \omega^2)$ 
  assumes indep-vars ( $\lambda\ -. \text{borel}$ ) f I
  shows variance ( $\lambda\omega. (\sum i \in I. f\ i\ \omega)$ ) = ( $\sum i \in I. \text{variance } (f\ i)$ )
  apply (rule var-sum-pairwise-indep-2[OF assms(1) assms(2) assms(3)], simp,
simp)
  using indep-vars-subset[OF assms(4)] by simp

end

```

9 Median

```

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

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)
  by simp

lemma sort-map-strict-mono:
  fixes f :: nat  $\Rightarrow$  'b :: linorder
  shows j < n  $\implies$  i < j  $\implies$  sort-map f n i  $\leq$  sort-map f n j
proof (induction n arbitrary: i j)
  case 0
  then show ?case by simp
next
  case (Suc n)
  define g where g = ( $\lambda k. \text{fold id } [\text{sort-primitive } j\ n. j <- [0.. $k$ ]] (\text{sort-map } f\ n))$ )
  define k where k = n
  have a: ( $\forall i\ j. j < n \longrightarrow i < j \longrightarrow g\ k\ i \leq g\ k\ j$ )  $\wedge$  ( $\forall l. l < k \longrightarrow g\ k\ l \leq g\ k\ n$ )
  proof (induction k)
    case 0
    then show ?case using Suc by (simp add: g-def del: sort-map.simps)
  next

```

```

case (Suc k)
have g (Suc k) = sort-primitive k n (g k)
  by (simp add:g-def)
then show ?case using Suc
  apply (cases g k k ≤ g k n)
  apply (simp add:min-def max-def)
  using less-antisym apply blast
  apply (cases g k n ≤ g k k)
  apply (simp add:min-def max-def)
  apply (metis less-antisym max.coboundedI2 max.orderE)
  by simp
qed

hence  $\bigwedge i j. j < \text{Suc } n \implies i < j \implies g \ n \ i \leq g \ n \ j$ 
  apply (simp add:k-def) using less-antisym by blast
moreover have sort-map f (Suc n) = g n
  by (simp add:sort-map-ind g-def del:sort-map.simps)
ultimately show ?case
  apply (simp del:sort-map.simps)
  using Suc by blast
qed

lemma sort-map-mono:
  fixes f :: nat  $\Rightarrow$  'b :: linorder
  shows j < n  $\implies$  i ≤ j  $\implies$  sort-map f n i ≤ sort-map f n j
  using sort-map-strict-mono
  by (metis eq-iff le-imp-less-or-eq)

lemma sort-map-perm:
  fixes f :: nat  $\Rightarrow$  'b :: linorder
  shows image-mset (sort-map f n) (mset [0..\lambda(ts :: ((nat \Rightarrow 'b) \Rightarrow nat \Rightarrow 'b)). \exists i < n. \exists j < n. ts = \text{sort-primitive } i \ j)
  define t :: ((nat  $\Rightarrow$  'b)  $\Rightarrow$  nat  $\Rightarrow$  'b) list
    where t = [sort-primitive j i. i <- [0..\bigwedge x f. is\text{-swap } x \implies \text{image-mset } (x \ f) \ (\text{mset-set } \{0..
  proof -
    fix x
    fix f :: nat  $\Rightarrow$  'b :: linorder
    assume is-swap x
    then obtain i j where x-def: x = sort-primitive i j and i-bound: i < n and j-bound: j < n
      using is-swap-def by blast
    define inv where inv = mset-set {k. k < n  $\wedge$  k  $\neq$  i  $\wedge$  k  $\neq$  j}
    have b: {0..\wedge k  $\neq$  i  $\wedge$  k  $\neq$  j}  $\cup$  {i,j}
    apply (rule order-antisym, rule subsetI, simp, blast, rule subsetI, simp)

```

```

    using i-bound j-bound by meson
  have c:  $\bigwedge k. k \in \# \text{ inv} \implies (x f) k = f k$ 
    by (simp add: x-def inv-def)
  have image-mset (x f) inv = image-mset f inv
    apply (rule multiset-eqI)
    using c multiset.map-cong0 by force
  moreover have image-mset (x f) (mset-set {i,j}) = image-mset f (mset-set {i,j})
    apply (cases i = j)
    by (simp add: x-def max-def min-def)+
  moreover have mset-set {0.. $n$ } = inv + mset-set {i,j}
    by (simp only: inv-def b, rule mset-set-Union, simp, simp, simp)
  ultimately show image-mset (x f) (mset-set {0.. $n$ }) = image-mset f (mset-set {0.. $n$ })
    by simp
qed

```

```

  have ( $\forall x \in \text{set } t. \text{is-swap } x \implies \text{image-mset (fold id } t \text{ f) (mset [0.. $n$ ])} =$ 
    image-mset f (mset [0.. $n$ ]))
    by (induction t arbitrary: f, simp, simp add: a)
  moreover have  $\bigwedge x. x \in \text{set } t \implies \text{is-swap } x$ 
    apply (simp add: t-def is-swap-def)
    by (meson atLeastLessThan-iff imageE less-imp-le less-le-trans)
  ultimately have image-mset (fold id t f) (mset [0.. $n$ ]) = image-mset f (mset [0.. $n$ ])
    by blast
  then show ?thesis by (simp add: t-def)
qed

```

```

lemma sort-map-eq-sort:
  fixes f :: nat  $\Rightarrow$  ('b :: linorder)
  shows map (sort-map f n) [0.. $n$ ] = sort (map f [0.. $n$ ]) (is ?A = ?B)
proof -
  have mset ?A = mset ?B
    using sort-map-perm[where f=f and n=n]
    by (simp del: sort-map.simps)
  moreover have sorted ?B
    by simp
  moreover have sorted ?A
    apply (subst sorted-wrt-iff-nth-less)
    apply (simp del: sort-map.simps)
    using sort-map-mono
    by (metis nat-less-le)
  ultimately show ?A = ?B
    using list-eq-iff by blast
qed

```

```

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

```

lemma *median-alt-def*:
assumes $n > 0$
shows $\text{median } f \ n = (\text{sort-map } f \ n) \ (n \text{ div } 2)$
using *assms*
by (*simp add:median-def sort-map-eq-sort[symmetric] del:sort-map.simps*)

definition *interval* :: $('a :: \text{linorder}) \text{ set} \Rightarrow \text{bool}$ **where**
 $\text{interval } I = (\forall x \ y \ z. x \in I \longrightarrow z \in I \longrightarrow x \leq y \longrightarrow y \leq z \longrightarrow y \in I)$

lemma *interval-rule*:
assumes *interval* I
assumes $a \leq x \ x \leq b$
assumes $a \in I$
assumes $b \in I$
shows $x \in I$
using *assms*(1) **apply** (*simp add:interval-def*)
using *assms* **by** *blast*

lemma *sorted-int*:
assumes *interval* I
assumes *sorted* xs
assumes $k < \text{length } xs \ i \leq j \ j \leq k$
assumes $xs ! i \in I \ xs ! k \in I$
shows $xs ! j \in I$
apply (*rule interval-rule[where a=xs ! i and b=xs ! k]*)
using *assms* **by** (*simp add: sorted-nth-mono*)+

lemma *mid-in-interval*:
assumes $2 * \text{length } (\text{filter } (\lambda x. x \in I) \ xs) > \text{length } xs$
assumes *interval* I
assumes *sorted* xs
shows $xs ! (\text{length } xs \text{ div } 2) \in I$
proof –
have $\text{length } (\text{filter } (\lambda x. x \in I) \ xs) > 0$ **using** *assms*(1) **by** *linarith*
then obtain v **where** $v-1: v < \text{length } xs$ **and** $v-2: xs ! v \in I$
by (*metis filter-False in-set-conv-nth length-greater-0-conv*)

define J **where** $J = \{k. k < \text{length } xs \wedge xs ! k \in I\}$

have *card-J-min*: $2 * \text{card } J > \text{length } xs$
using *assms*(1) **by** (*simp add:J-def length-filter-conv-card*)

consider
 $(a) \ xs ! (\text{length } xs \text{ div } 2) \in I \mid$
 $(b) \ xs ! (\text{length } xs \text{ div } 2) \notin I \wedge v > (\text{length } xs \text{ div } 2) \mid$
 $(c) \ xs ! (\text{length } xs \text{ div } 2) \notin I \wedge v < (\text{length } xs \text{ div } 2)$
by (*metis linorder-cases v-2*)

```

thus ?thesis
proof (cases)
  case a
  then show ?thesis by simp
next
  case b
  have p:  $\bigwedge k. k \leq \text{length } xs \text{ div } 2 \implies xs ! k \notin I$ 
    using b v-2 sorted-int[OF assms(2) assms(3) v-1, where j=length xs div 2]
apply simp by blast
  have card J  $\leq$  card {Suc (length xs div 2)..<length xs}
    apply (rule card-mono, simp)
    apply (rule subsetI, simp add: J-def not-less-eq-eq[symmetric])
    using p by metis
  hence card J  $\leq$  length xs - (Suc (length xs div 2))
    using card-atLeastLessThan by metis
  hence length xs  $\leq$  2*(length xs - (Suc (length xs div 2)))
    using card-J-min by linarith
  hence False
    apply (simp add: nat-distrib)
    apply (subst (asm) le-diff-conv2)
    using b v-1 apply linarith
    by simp
  then show ?thesis by simp
next
  case c
  have p:  $\bigwedge k. k \geq \text{length } xs \text{ div } 2 \implies k < \text{length } xs \implies xs ! k \notin I$ 
    using c v-1 v-2 sorted-int[OF assms(2) assms(3), where i=v and j=length
xs div 2] apply simp by blast
  have card J  $\leq$  card {0..<(length xs div 2)}
    apply (rule card-mono, simp)
    apply (rule subsetI, simp add: J-def not-less-eq-eq[symmetric])
    using p linorder-le-less-linear by blast
  hence card J  $\leq$  (length xs div 2)
    using card-atLeastLessThan by simp
  then show ?thesis using card-J-min by linarith
qed
qed

```

lemma median-est:

```

fixes  $\delta :: \text{real}$ 
assumes  $2 * \text{card } \{k. k < n \wedge \text{abs } (f k - \mu) \leq \delta\} > n$ 
shows  $\text{abs } (\text{median } f \ n - \mu) \leq \delta$ 
proof -
  have a:  $\{k. k < n \wedge \text{abs } (f k - \mu) \leq \delta\} = \{i. i < n \wedge |\text{map } f \ [0..<n] ! i - \mu| \leq \delta\}$ 
    apply (rule order-antisym)
    apply (rule subsetI, simp)
    apply (rule subsetI, simp)

```



```

by (metis add-0 diff-zero nth-map-upt)

show ?thesis
  apply (simp add:median-def)
  apply (rule mid-in-interval[where I={x. abs (x-μ) ≤ δ} and xs=sort (map
f [0..

```

```

proof -
  fix x g
  assume meas-ptw g
  hence a:  $\bigwedge k. k < n \implies g k \in \text{borel-measurable } M$  by (simp add: meas-ptw-def)
  assume is-swap x
  then obtain i j where x-def:  $x = \text{sort-primitive } i \ j$  and i-le:  $i < n$  and j-le:  $j < n$ 
n
    apply (simp add: is-swap-def) by blast
  have  $\bigwedge k. k < n \implies (\lambda \omega. x (\lambda i. g \ i \ \omega) \ k) \in \text{borel-measurable } M$ 
  proof -
    fix k
    assume k < n
    thus  $(\lambda \omega. x (\lambda i. g \ i \ \omega) \ k) \in \text{borel-measurable } M$ 
      apply (simp add: x-def)
      apply (cases k = i, simp)
      using a i-le j-le borel-measurable-min apply blast
      apply (cases k = j, simp)
      using a i-le j-le borel-measurable-max apply blast
      using a by simp
    qed
  thus meas-ptw  $(\lambda k \ \omega. x (\lambda i. g \ i \ \omega) \ k)$ 
    by (simp add: meas-ptw-def)
  qed

have  $(\forall x \in \text{set } t. \text{is-swap } x) \implies \text{meas-ptw } (\lambda k \ \omega. (\text{fold id } t (\lambda k. X \ k \ \omega)) \ k)$ 
proof (induction t rule: rev-induct)
  case Nil
  then show ?case using assms by (simp add: meas-ptw-def)
next
  case (snoc x xs)
  have a: meas-ptw  $(\lambda k \ \omega. \text{fold } (\lambda a. a) \ xs \ (\lambda k. X \ k \ \omega) \ k)$  using snoc by simp
  have b: is-swap x using snoc by simp
  show ?case apply simp
    using ind-step[OF a b] by simp
  qed
moreover have  $\bigwedge x. x \in \text{set } t \implies \text{is-swap } x$ 
  apply (simp add: t-def is-swap-def)
  by (meson atLeastLessThan-iff imageE less-imp-le less-le-trans)
moreover have  $n \text{ div } 2 < n$  using n-ge-0 by simp
ultimately show ?thesis
  apply (subst median-alt-def[OF n-ge-0])
  by (simp add: t-def[symmetric] meas-ptw-def)
qed

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 ( $\lambda$ -. borel)  $X \{0..<n\}$ 
assumes  $n \geq -\ln \varepsilon / (2 * \alpha^2)$ 
assumes  $\bigwedge i. i < n \implies \mathcal{P}(\omega \text{ in } M. X \ i \ \omega \in \{a..b\}) \geq 1/2 + \alpha$ 
shows  $\mathcal{P}(\omega \text{ in } M. \text{median } (\lambda i. X \ i \ \omega) \ n \in \{a..b\}) \geq 1 - \varepsilon$  (is  $\mathcal{P}(\omega \text{ in } M. ?lhs \ \omega) \geq ?C$ )
proof -
  define  $Y :: nat \Rightarrow 'a \Rightarrow real$  where  $Y = (\lambda i. \text{indicator } \{a..b\} \circ (X \ i))$ 

  define  $t$  where  $t = (\sum i = 0..<n. \text{expectation } (Y \ i)) - n/2$ 
  have  $0 < -\ln \varepsilon / (2 * \alpha^2)$ 
    apply (rule divide-pos-pos)
    apply (simp, subst ln-less-zero-iff)
    using assms by auto
  also have  $\dots \leq real \ n$  using assms by simp
  finally have  $real \ n > 0$  by simp
  hence  $n\text{-ge-}1:n \geq 1$  by linarith
  hence  $n\text{-ge-}0:n > 0$  by simp

  have ind-comp:  $\bigwedge i. \text{indicator } \{a..b\} \circ (X \ i) = \text{indicator } \{\omega. X \ i \ \omega \in \{a..b\}\}$ 
    apply (rule ext)
    by (simp add:indicator-def comp-def)

  have  $\alpha * n \leq (\sum i = 0..<n. 1/2 + \alpha) - n/2$ 
    by (simp add:algebra-simps)
  also have  $\dots \leq (\sum i = 0..<n. \text{expectation } (Y \ i)) - n/2$ 
    apply (rule diff-right-mono, rule sum-mono)
    using assms(5) by (simp add:Y-def ind-comp measure-inters)
  also have  $\dots = t$  by (simp add:t-def)
  finally have  $t\text{-ge-}a: t \geq \alpha * n$  by simp

  have  $d: 0 \leq \alpha * n$ 
    apply (rule mult-nonneg-nonneg)
    using assms(1)  $n\text{-ge-}0$  by simp+
  also have  $\dots \leq t$  using  $t\text{-ge-}a$  by simp
  finally have  $t\text{-ge-}0: t \geq 0$  by simp

  have  $(\alpha * n)^2 \leq t^2$  using  $t\text{-ge-}a$   $d$  power-mono by blast
  hence  $t\text{-ge-}a\text{-sq}: \alpha^2 * real \ n * real \ n \leq t^2$ 
    by (simp add:algebra-simps power2-eq-square)

  have  $Y\text{-indep}: indep\text{-vars } (\lambda$ -. borel)  $Y \{0..<n\}$ 
    apply (subst Y-def)
    apply (rule indep-vars-compose[where  $M' = (\lambda$ -. borel)])
    apply (metis assms(3))
    by simp

  hence  $b\text{:Hoeffding-ineq } M \{0..<n\} \ Y \ (\lambda i. 0) \ (\lambda i. 1)$ 
    apply (simp add:Hoeffding-ineq-def indep-interval-bounded-random-variables-def)
    by (simp add:prob-space-axioms indep-interval-bounded-random-variables-axioms-def

```

Y-def Y-indep)

have $c: \bigwedge \omega. (\sum i = 0..<n. Y\ i\ \omega) > n/2 \implies \text{median } (\lambda i. X\ i\ \omega)\ n \in \{a..b\}$
 proof –
 fix ω
 assume $(\sum i = 0..<n. Y\ i\ \omega) > n/2$
 hence $n < 2 * \text{card } (\{0..<n\} \cap \{i. X\ i\ \omega \in \{a..b\}\})$
 by (simp add: *Y-def indicator-def*)
 also have $\dots = 2 * \text{card } \{i. i < n \wedge X\ i\ \omega \in \{a..b\}\}$
 apply (simp)
 apply (rule arg-cong[where $f=\text{card}$])
 by (rule order-antisym, rule subsetI, simp, rule subsetI, simp)
 finally have $2 * \text{card } \{i. i < n \wedge X\ i\ \omega \in \{a..b\}\} > n$ by simp
 thus $\text{median } (\lambda i. X\ i\ \omega)\ n \in \{a..b\}$
 using *median-est-2* by simp
 qed

have $1 - \varepsilon \leq 1 - \exp(- (2 * \alpha^2 * \text{real } n))$
 apply simp
 apply (subst *ln-ge-iff[symmetric]*)
 using *assms(2)* apply simp
 using *assms(4)* apply (subst (asm) *pos-divide-le-eq*)
 apply (simp add: *assms(1) power2-eq-square*)
 by (simp add: *mult-of-nat-commute*)
 also have $\dots \leq 1 - \exp(- (2 * t^2 / \text{real } n))$
 apply simp
 apply (subst *pos-le-divide-eq*) using *n-ge-0* apply simp
 using *t-ge-a-sq* by *linarith*
 also have $\dots \leq 1 - \mathcal{P}(\omega \text{ in } M. (\sum i = 0..<n. Y\ i\ \omega) \leq n/2)$
 using *Hoeffding-ineq.Hoeffding-ineq-le[OF b, where $\varepsilon=t$, simplified]* *n-ge-0*
t-ge-0
 by (simp add: *t-def*)
 also have $\dots = \mathcal{P}(\omega \text{ in } M. (\sum i = 0..<n. Y\ i\ \omega) > n/2)$
 apply (subst *prob-compl[symmetric]*)
 apply measurable
 using *Y-indep* apply (simp add: *indep-vars-def*)
 apply (rule arg-cong2[where $f=\text{measure}$], simp)
 by (rule order-antisym, rule subsetI, simp add: *not-le*, rule subsetI, simp add: *not-le*)
 also have $\dots \leq \mathcal{P}(\omega \text{ in } M. \text{median } (\lambda i. X\ i\ \omega)\ n \in \{a..b\})$
 apply (rule *finite-measure-mono*)
 apply (rule subsetI) using *c* apply simp
 apply measurable
 apply (rule *median-measurable[OF n-ge-1]*)
 using *assms(3)* by (simp add: *indep-vars-def*)
 finally show ?thesis by simp
 qed

lemma (in *prob-space*) *median-bound-2*:
 fixes $\mu :: \text{real}$

```

fixes  $\delta :: \text{real}$ 
assumes  $\varepsilon \in \{0 < .. < 1\}$ 
assumes indep-vars  $(\lambda -. \text{borel}) X \{0 .. < n\}$ 
assumes  $n \geq -18 * \ln \varepsilon$ 
assumes  $\bigwedge i. i < n \implies \mathcal{P}(\omega \text{ in } M. \text{abs } (X \ i \ \omega - \mu) > \delta) \leq 1/3$ 
shows  $\mathcal{P}(\omega \text{ in } M. \text{abs } (\text{median } (\lambda i. X \ i \ \omega) \ n - \mu) \leq \delta) \geq 1 - \varepsilon$ 
proof -
  have  $b: \bigwedge i. i < n \implies \text{space } M - \{\omega \in \text{space } M. X \ i \ \omega \in \{\mu - \delta .. \mu + \delta\}\} =$ 
 $\{\omega \in \text{space } M. \text{abs } (X \ i \ \omega - \mu) > \delta\}$ 
    apply (rule order-antisym)
    apply (rule subsetI, simp, linarith)
    by (rule subsetI, simp, linarith)

  have  $\bigwedge i. i < n \implies 1 - \mathcal{P}(\omega \text{ in } M. X \ i \ \omega \in \{\mu - \delta .. \mu + \delta\}) \leq 1/3$ 
    apply (subst prob-compl[symmetric])
    apply (measurable)
    using assms(2) apply (simp add: indep-vars-def)
    apply (subst b, simp)
    using assms(4) by simp

  hence  $a: \bigwedge i. i < n \implies \mathcal{P}(\omega \text{ in } M. X \ i \ \omega \in \{\mu - \delta .. \mu + \delta\}) \geq 2/3$  by simp

  have  $1 - \varepsilon \leq \mathcal{P}(\omega \text{ in } M. \text{median } (\lambda i. X \ i \ \omega) \ n \in \{\mu - \delta .. \mu + \delta\})$ 
    apply (rule median-bound-gen[OF - assms(1) assms(2), where  $\alpha=1/6$ ], simp)

    apply (simp add: power2-eq-square)
    using assms(3) apply simp
    using a by simp

  also have  $... = \mathcal{P}(\omega \text{ in } M. \text{abs } (\text{median } (\lambda i. X \ i \ \omega) \ n - \mu) \leq \delta)$ 
    apply (rule arg-cong2[where  $f=\text{measure}$ ], simp)
    apply (rule order-antisym)
    apply (rule subsetI, simp, linarith)
    by (rule subsetI, simp, linarith)

  finally show ?thesis by simp
qed

```

```

lemma sorted-mono-map:
  assumes sorted xs
  assumes mono f
  shows sorted (map f xs)
  using assms apply (simp add: sorted-wrt-map)
  apply (rule sorted-wrt-mono-rel[where  $P=(\leq)$ ])
  by (simp add: mono-def, simp)

```

```

lemma map-sort:
  assumes mono f
  shows  $\text{sort } (\text{map } f \ xs) = \text{map } f \ (\text{sort } xs)$ 
  apply (rule properties-for-sort)
  apply simp

```

```

by (rule sorted-mono-map, simp, simp add:assms)

lemma median-cong:
  assumes  $\bigwedge i. i < n \implies f\ i = g\ i$ 
  shows  $median\ f\ n = median\ g\ n$ 
  apply (cases  $n = 0$ , simp add:median-def)
  apply (simp add:median-def)
  apply (rule arg-cong2[where  $f=(!)$ ])
  apply (rule arg-cong[where  $f=sort$ ])
  by (rule map-cong, simp, simp add:assms, simp)

lemma median-restrict:
  assumes  $n > 0$ 
  shows  $median\ (\lambda i \in \{0..<n\}. f\ i)\ n = median\ f\ n$ 
  by (rule median-cong, simp)

lemma median-rat:
  assumes  $n > 0$ 
  shows  $real-of-rat\ (median\ f\ n) = median\ (\lambda i. real-of-rat\ (f\ i))\ n$ 
proof -
  have  $a: map\ (\lambda i. real-of-rat\ (f\ i))\ [0..<n] =$ 
     $map\ real-of-rat\ (map\ (\lambda i. f\ i)\ [0..<n])$ 
  by (simp)
  show ?thesis
    apply (simp add:a median-def del:map-map)
    apply (subst map-sort[where  $f=real-of-rat$ ], simp add:mono-def of-rat-less-eq)
    apply (subst nth-map[where  $f=real-of-rat$ ]) using assms
    apply fastforce
    by simp
qed

lemma median-const:
  assumes  $k > 0$ 
  shows  $median\ (\lambda i \in \{0..<k\}. a)\ k = a$ 
proof -
  have  $b: sorted\ (map\ (\lambda-. a)\ [0..<k])$ 
  by (subst sorted-wrt-map, simp)
  have  $a: sort\ (map\ (\lambda-. a)\ [0..<k]) = map\ (\lambda-. a)\ [0..<k]$ 
  by (subst sorted-sort-id[OF b], simp)
  have  $median\ (\lambda i \in \{0..<k\}. a)\ k = median\ (\lambda-. a)\ k$ 
  by (subst median-restrict[OF assms(1)], simp)
  also have  $\dots = a$ 
  apply (simp add:median-def a)
  apply (subst nth-map)
  using assms by simp+
  finally show ?thesis by simp
qed

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  $A \subseteq f^{-1} B$ 
  shows  $\text{card } (f^{-1} A \cap B) = \text{card } A$ 
proof -
  have  $A = f^{-1} (f^{-1} A \cap B)$  using assms(2) by auto
  thus ?thesis using assms card-image
    by (metis inf-le2 inj-on-subset)
qed

```

```

lemma card-ordered-pairs:

```

```

  fixes  $M :: ('a :: \text{linorder}) \text{ set}$ 
  assumes finite M
  shows  $2 * \text{card } \{(x,y) \in M \times M. x < y\} = \text{card } M * (\text{card } M - 1)$ 
proof -
  have  $2 * \text{card } \{(x,y) \in M \times M. x < y\} =$ 
     $\text{card } \{(x,y) \in M \times M. x < y\} + \text{card } ((\lambda x. (\text{snd } x, \text{fst } x))^{-1} \{(x,y) \in M \times M. x$ 
     $< y\})$ 
    apply (subst card-image)
    apply (rule inj-onI, simp add:case-prod-beta prod-eq-iff)
    by simp
  also have  $\dots = \text{card } \{(x,y) \in M \times M. x < y\} + \text{card } \{(x,y) \in M \times M. y < x\}$ 
    apply (rule arg-cong2[where f=(+)], simp)
    apply (rule arg-cong[where f=card])
    apply (rule order-antisym)
    apply (rule image-subsetI, simp add:case-prod-beta)
    apply (rule subsetI, simp)
    using image-iff by fastforce
  also have  $\dots = \text{card } (\{(x,y) \in M \times M. x < y\} \cup \{(x,y) \in M \times M. y < x\})$ 
    apply (rule card-Un-disjoint[symmetric])
    apply (rule finite-subset[where B=M × M], rule subsetI, simp add:case-prod-beta
    mem-Times-iff)
    using assms apply simp
    apply (rule finite-subset[where B=M × M], rule subsetI, simp add:case-prod-beta
    mem-Times-iff)
    using assms apply simp
    apply (rule order-antisym, rule subsetI, simp add:case-prod-beta, force)
    by simp
  also have  $\dots = \text{card } ((M \times M) - \{(x,y) \in M \times M. x = y\})$ 
    apply (rule arg-cong[where f=card])
    apply (rule order-antisym, rule subsetI, simp add:case-prod-beta, force)
    by (rule subsetI, simp add:case-prod-beta, force)
  also have  $\dots = \text{card } (M \times M) - \text{card } \{(x,y) \in M \times M. x = y\}$ 
    apply (rule card-Diff-subset)

```

```

  apply (rule finite-subset[where B=M × M], rule subsetI, simp add:case-prod-beta
mem-Times-iff)
  using assms apply simp
  by (rule subsetI, simp add:case-prod-beta mem-Times-iff)
also have ... = card M ^ 2 - card ((λx. (x,x)) ` M)
  apply (rule arg-cong2[where f=(-)])
  using assms apply (simp add:power2-eq-square)
  apply (rule arg-cong[where f=card])
  apply (rule order-antisym, rule subsetI, simp add:case-prod-beta, force)
  by (rule image-subsetI, simp)
also have ... = card M ^ 2 - card M
  apply (rule arg-cong2[where f=(-)], simp)
  apply (rule card-image)
  by (rule inj-onI, simp)
also have ... = card M * (card M - 1)
  apply (cases card M ≥ 0, simp add:power2-eq-square algebra-simps)
  by simp
finally show ?thesis by simp
qed

end

```

10 Order Statistics

theory *OrderStatistics*

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

This section contains definitions and results about order statistics.

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

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

lemma *rank-mono*:

```

  assumes finite S
  shows  $x \leq y \implies \text{rank-of } x\ S \leq \text{rank-of } y\ S$ 
  apply (simp add:rank-of-def)
  apply (rule card-mono)
  using assms apply simp
  by (rule subsetI, simp, force)

```

lemma *rank-mono-commute*:

```

  assumes finite S
  assumes  $S \subseteq T$ 
  assumes strict-mono-on f T
  assumes  $x \in T$ 
  shows  $\text{rank-of } x\ S = \text{rank-of } (f\ x)\ (f\ ` S)$ 
proof -

```



```

have rank-of (f x) (f ‘ S) = card (f ‘ {y ∈ S. y < x})
  apply (simp add:rank-of-def)
  apply (rule arg-cong[where f=card])
  apply (rule order-antisym)
  apply (rule subsetI, simp)
  using assms strict-mono-on-leD apply fastforce
  apply (rule image-subsetI, simp)
  using assms by (simp add: in-mono strict-mono-on-def)
also have ... = card {y ∈ S. y < x}
  apply (rule card-image)
  apply (rule inj-on-subset[where A=T])
  apply (metis assms(3) strict-mono-on-imp-inj-on)
  using assms by blast
also have ... = rank-of x S
  by (simp add:rank-of-def)
finally show ?thesis
  by simp
qed

```

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

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

lemma *rank-strict-mono*:

```

assumes finite S
shows strict-mono-on (λx. rank-of x S) S
proof -
  have ∧x y. x ∈ S ⇒ y ∈ S ⇒ x < y ⇒ rank-of x S < rank-of y S
    apply (simp add:rank-of-def)
    apply (rule psubset-card-mono)
    apply (simp add:assms)
    apply (simp add: psubset-eq)
    apply (rule conjI, rule subsetI, force)
    by blast

```

```

thus ?thesis
  by (simp add:rank-of-def strict-mono-on-def)
qed

```

lemma *rank-of-image*:

```

assumes finite S
shows (λx. rank-of x S) ‘ S = {0..<card S}
  apply (rule card-seteq, simp)
  apply (rule image-subsetI, simp add:rank-of-def)
  apply (rule psubset-card-mono, metis assms, blast)
  apply simp
  apply (subst card-image)
  apply (metis strict-mono-on-imp-inj-on rank-strict-mono assms)

```

by *simp*

lemma *card-least*:
 assumes *finite S*
 shows $\text{card } (\text{least } k \ S) = \min k \ (\text{card } S)$
proof (*cases card S < k*)
 case *True*
 have $\bigwedge t. \text{rank-of } t \ S \leq \text{card } S$
 apply (*simp add:rank-of-def*)
 by (*rule card-mono, metis assms, simp*)
 hence $\bigwedge t. \text{rank-of } t \ S < k$
 by (*metis True not-less-iff-gr-or-eq order-less-le-trans*)
 hence $\text{least } k \ S = S$
 by (*simp add:least-def*)
 then show *?thesis* using *True* by *simp*

next
 case *False*
 hence $a:\text{card } S \geq k$ using *leI* by *blast*
 have $\text{card } ((\lambda x. \text{rank-of } x \ S) - ' \{0..<k\} \cap S) = \text{card } \{0..<k\}$
 apply (*rule card-vimage-inj-on*)
 apply (*metis strict-mono-on-imp-inj-on rank-strict-mono assms*)
 apply (*subst rank-of-image, metis assms*)
 using *a* by *simp*
 hence $\text{card } (\text{least } k \ S) = k$
 by (*simp add: Collect-conj-eq Int-commute least-def vimage-def*)
 then show *?thesis* using *a* by *linarith*

qed

lemma *least-subset*: $\text{least } k \ S \subseteq S$
 by (*simp add:least-def*)

lemma *preserve-rank*:
 assumes *finite S*
 shows $\text{rank-of } x \ (\text{least } m \ S) = \min m \ (\text{rank-of } x \ S)$
proof (*cases rank-of x S ≥ m*)
 case *True*
 hence $\{y \in \text{least } m \ S. y < x\} = \text{least } m \ S$
 apply (*simp add: least-def*)
 apply (*rule Collect-cong*)
 using *rank-mono[OF assms]*
 by (*metis linorder-not-less order-less-le-trans*)
 moreover have $m \leq \text{card } S$
 apply (*rule order-trans[where y=rank-of x S], metis True*)
 apply (*simp add:rank-of-def*)
 by (*rule card-mono[OF assms], simp*)
 hence $\text{card } (\text{least } m \ S) = m$
 apply (*subst card-least[OF assms]*)
 by *simp*
 ultimately show *?thesis* using *True* by (*simp add:rank-of-def*)

```

next
  case False
  have rank-of x (least m S) = rank-of x S
    apply (simp add:rank-of-def)
    apply (rule arg-cong[where f=card])
    apply (rule Collect-cong)
    apply (simp add: least-def)
    by (metis False rank-mono[OF assms] less-le-not-le min-def min-less-iff-conj
nle-le)
  thus ?thesis using False by simp
qed

```

```

lemma rank-insert:
  assumes finite T
  shows rank-of y (insert v T) = of-bool (v < y ∧ v ∉ T) + rank-of y T
proof -
  have a:v ∉ T ⟹ v < y ⟹ rank-of y (insert v T) = Suc (rank-of y T)
  proof -
    assume a-1: v ∉ T
    assume a-2: v < y
    have rank-of y (insert v T) = card (insert v {z ∈ T. z < y})
      apply (simp add: rank-of-def)
      apply (subst insert-compr)
      by (metis a-2 mem-Collect-eq)
    also have ... = Suc (card {z ∈ T. z < y})
      apply (subst card-insert-disjoint)
      using assms a-1 by simp+
    also have ... = Suc (rank-of y T)
      by (simp add:rank-of-def)
    finally show rank-of y (insert v T) = Suc (rank-of y T)
      by blast
  qed
  have b:v ∉ T ⟹ ¬(v < y) ⟹ rank-of y (insert v T) = rank-of y T
    by (simp add:rank-of-def, metis)
  have c:v ∈ T ⟹ rank-of y (insert v T) = rank-of y T
    by (simp add:insert-absorb)

```

```

  show ?thesis
    apply (cases v ∈ T, simp add: c)
    apply (cases v < y, simp add: a)
    by (simp add: b)
qed

```

```

lemma least-mono-commute:
  assumes finite S
  assumes strict-mono-on f S
  shows f ' least k S = least k (f ' S)
proof -
  have a:inj-on f S

```

```

    using strict-mono-on-imp-inj-on[OF assms(2)] by simp
  have b: card (least k (f ' S)) ≤ card (f ' least k S)
    apply (subst card-least, simp add:assms)
    apply (subst card-image, metis a)
    apply (subst card-image, rule inj-on-subset[OF a], simp add:least-def)
    by (subst card-least, simp add:assms, simp)

  show ?thesis
    apply (rule card-seteq, simp add:least-def assms)
    apply (rule image-subsetI, simp add:least-def)
    apply (subst rank-mono-commute[symmetric, where T=S], metis assms(1),
    simp, metis assms(2), simp, simp)
    by (metis b)
qed

lemma least-insert:
  assumes finite S
  shows least k (insert x (least k S)) = least k (insert x S) (is ?lhs = ?rhs)
proof -
  have c: x ∈ least k S ⟹ x ∈ S by (simp add:least-def)
  have b:min k (card (insert x S)) ≤ card (insert x (least k S))
    apply (cases x ∈ least k S)
    using c apply (simp add: insert-absorb)
    apply (subst card-least, simp add:assms least-def, simp)
    apply (subst card-insert-disjoint, simp add:assms least-def, simp)
    apply (cases x ∈ S)
    apply (simp add:insert-absorb)
    apply (subst card-least, simp add:assms least-def)
    using nat-less-le apply blast
    apply (subst card-insert-disjoint, simp add:assms least-def, simp)
    apply (subst card-least, simp add:assms least-def)
    by simp
  have a:card ?rhs ≤ card ?lhs
    apply (subst card-least, simp add:assms least-def)
    apply (subst card-least, simp add:assms least-def)
    by (meson b min.boundedI min.cobounded1)

  have d:⋀y. y ∈ least k (insert x (least k S)) ⟹ y ∈ least k (insert x S)
    apply (subst least-def, subst (asm) least-def)
    apply (subst rank-insert[OF assms])
    apply (subst (asm) rank-insert, simp add:assms least-def)
    apply (subst (asm) preserve-rank, simp add:assms)
    apply (cases x ∈ least k S)
  apply (simp, metis insert-subset least-subset min.strict-order-iff min-def mk-disjoint-insert)
  apply (simp)
    using least-def apply fastforce
  by (metis insert-subset least-subset min-def mk-disjoint-insert nat-neq-iff)

show ?thesis

```

```

    apply (rule card-seteq, simp add:least-def assms)
    apply (rule subsetI, metis d)
    using a by simp
qed

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

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

lemma nth-mset-bound-left:
  assumes  $k < \text{size } M$ 
  assumes count-less  $x\ M \leq k$ 
  shows  $x \leq \text{nth-mset } k\ M$ 
proof (rule ccontr)
  define xs where xs = sorted-list-of-multiset M
  have s-xs: sorted xs by (simp add:xs-def sorted-sorted-list-of-multiset)
  have l-xs:  $k < \text{length } xs$  apply (simp add:xs-def)
    by (metis size-mset mset-sorted-list-of-multiset assms(1))
  have M-xs:  $M = \text{mset } xs$  by (simp add:xs-def)
  hence a: $\bigwedge i. i \leq k \implies xs\ !\ i \leq xs\ !\ k$ 
    using s-xs l-xs sorted-iff-nth-mono by blast

  assume  $\neg(x \leq \text{nth-mset } k\ M)$ 
  hence  $x > \text{nth-mset } k\ M$  by simp
  hence b: $x > xs\ !\ k$  by (simp add:nth-mset-def xs-def[symmetric])

  have  $k < \text{card } \{0..k\}$  by simp
  also have  $\dots \leq \text{card } \{i. i < \text{length } xs \wedge xs\ !\ i < x\}$ 
    apply (rule card-mono, simp)
    apply (rule subsetI, simp)
    using a b l-xs order-le-less-trans by auto
  also have  $\dots = \text{count-less } x\ M$ 
    apply (simp add:count-less-def M-xs)
    apply (subst mset-filter[symmetric], subst size-mset)
    by (subst length-filter-conv-card, simp)
  also have  $\dots \leq k$ 
    using assms by simp
  finally show False by simp
qed

lemma nth-mset-bound-left-excl:
  assumes  $k < \text{size } M$ 
  assumes count-le  $x\ M \leq k$ 
  shows  $x < \text{nth-mset } k\ M$ 
proof (rule ccontr)
  define xs where xs = sorted-list-of-multiset M
  have s-xs: sorted xs by (simp add:xs-def sorted-sorted-list-of-multiset)

```

```

have l-xs:  $k < \text{length } xs$  apply (simp add:xs-def)
  by (metis size-mset mset-sorted-list-of-multiset assms(1))
have M-xs:  $M = \text{mset } xs$  by (simp add:xs-def)
hence a: $\bigwedge i. i \leq k \implies xs ! i \leq xs ! k$ 
  using s-xs l-xs sorted-iff-nth-mono by blast

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

have  $k+1 \leq \text{card } \{0..k\}$  by simp
also have  $\dots \leq \text{card } \{i. i < \text{length } xs \wedge xs ! i \leq xs ! k\}$ 
  apply (rule card-mono, simp)
  apply (rule subsetI, simp)
  using a b l-xs order-le-less-trans by auto
also have  $\dots \leq \text{card } \{i. i < \text{length } xs \wedge xs ! i \leq x\}$ 
  apply (rule card-mono, simp)
  apply (rule subsetI, simp) using b
  by force
also have  $\dots = \text{count-le } x \ M$ 
  apply (simp add:count-le-def M-xs)
  apply (subst mset-filter[symmetric], subst size-mset)
  by (subst length-filter-conv-card, simp)
also have  $\dots \leq k$ 
  using assms by simp
finally show False by simp
qed

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

  assume  $\neg(\text{nth-mset } k \ M \leq x)$ 
  hence  $x < \text{nth-mset } k \ M$  by simp
  hence  $x < xs ! k$ 
    by (simp add:nth-mset-def xs-def[symmetric])
  hence a: $\bigwedge i. i < \text{length } xs \wedge xs ! i \leq x \implies i < k$ 
    using s-xs l-xs sorted-iff-nth-mono leI by fastforce
  have count-le  $x \ M \leq \text{card } \{i. i < \text{length } xs \wedge xs ! i \leq x\}$ 
    apply (simp add:count-le-def M-xs)
    apply (subst mset-filter[symmetric], subst size-mset)
    apply (subst length-filter-conv-card)

```

```

    by (rule card-mono, simp, simp)
  also have ...  $\leq$  card  $\{i. i < k\}$ 
    apply (rule card-mono, simp)
    by (rule subsetI, simp add:a)
  also have ... = k by simp
  finally have count-le x M  $\leq$  k by simp
  thus False using assms by simp
qed

```

lemma *nth-mset-commute-mono*:

```

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

```

lemma *nth-mset-max*:

```

  assumes size A > k
  assumes  $\bigwedge x. x \leq \text{nth-mset } k \ A \implies \text{count } A \ x \leq 1$ 
  shows nth-mset k A = Max (least (k+1) (set-mset A)) and card (least (k+1)
  (set-mset A)) = k+1
proof -
  define xs where xs = sorted-list-of-multiset A
  have k-bound: k < length xs apply (simp add:xs-def)
    by (metis size-mset mset-sorted-list-of-multiset assms(1))

  have A-def: A = mset xs by (simp add:xs-def)
  have s-xs: sorted xs by (simp add:xs-def sorted-sorted-list-of-multiset)
  have a-2:  $\bigwedge x. x \leq xs \ ! \ k \implies \text{count-list } xs \ x \leq 1$ 
    using assms(2) apply (simp add:xs-def[symmetric] nth-mset-def)
    by (simp add:A-def count-mset)

  have inj-xs: inj-on  $(\lambda k. xs \ ! \ k) \ \{0..k\}$ 
    apply (rule inj-onI)
    apply simp
    by (metis (full-types) count-list-ge-2-iff k-bound a-2
    le-neq-implies-less linorder-not-le order-le-less-trans s-xs sorted-iff-nth-mono)

  have rank-conv-2:  $\bigwedge y. y < \text{length } xs \implies \text{rank-of } (xs \ ! \ y) \ (\text{set } xs) < k+1 \implies y$ 
  < k+1
  proof (rule ccontr)
    fix y
    assume b:y < length xs
    assume  $\neg y < k + 1$ 

```

```

hence  $a:k + 1 \leq y$  by simp

have  $d:\text{Suc } k < \text{length } xs$  using  $a$   $b$  by simp

have  $k+1 = \text{card } (!) xs \text{ ' } \{0..k\}$ 
  by (subst card-image[OF inj-xs], simp)
also have  $\dots \leq \text{rank-of } (xs ! (k+1)) (\text{set } xs)$ 
  apply (simp add:rank-of-def)
  apply (rule card-mono, simp)
  apply (rule image-subsetI, simp)
  apply (rule conjI) using  $k\text{-bound}$  apply simp
  by (metis count-list-ge-2-iff a-2 not-le le-imp-less-Suc s-xs sorted-iff-nth-mono
d order-less-le)
also have  $\dots \leq \text{rank-of } (xs ! y) (\text{set } xs)$ 
  apply (simp add:rank-of-def)
  apply (rule card-mono, simp)
  apply (rule subsetI, simp)
  by (metis Suc-eq-plus1 a b s-xs order-less-le-trans sorted-iff-nth-mono)
also assume  $\dots < k+1$ 
finally show False by force
qed

have  $\text{rank-conv-1}: \bigwedge y. y < k + 1 \implies \text{rank-of } (xs ! y) (\text{set } xs) < k+1$ 
proof -
  fix  $y$ 
  have  $\text{rank-of } (xs ! y) (\text{set } xs) \leq \text{card } ((\lambda k. xs ! k) \text{ ' } \{k. k < \text{length } xs \wedge xs ! k$ 
     $< xs ! y\})$ 
    apply (simp add:rank-of-def)
    apply (rule card-mono, simp)
    apply (rule subsetI, simp)
    by (metis (no-types, lifting) imageI in-set-conv-nth mem-Collect-eq)
  also have  $\dots \leq \text{card } \{k. k < \text{length } xs \wedge xs ! k < xs ! y\}$ 
    by (rule card-image-le, simp)
  also have  $\dots \leq \text{card } \{k. k < y\}$ 
    apply (rule card-mono, simp)
    apply (rule subsetI, simp)
    apply (rule ccontr, simp add:not-less)
    by (meson leD sorted-iff-nth-mono s-xs)
  also have  $\dots = y$  by simp
  also assume  $y < k + 1$ 
  finally show  $\text{rank-of } (xs ! y) (\text{set } xs) < k+1$  by simp
qed

have  $\text{rank-conv}: \bigwedge y. y < \text{length } xs \implies \text{rank-of } (xs ! y) (\text{set } xs) < k+1 \longleftrightarrow y$ 
 $< k+1$ 
  using rank-conv-1 rank-conv-2 by blast

have  $\text{max-1}: \bigwedge y. y \in \text{least } (k+1) (\text{set } xs) \implies y \leq xs ! k$ 
proof -

```



```

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

have max-2: xs ! k ∈ least (k+1) (set xs)
  apply (simp add: least-def)
  using k-bound rank-conv by simp

have r-1: Max (least (k+1) (set xs)) = xs ! k
  apply (rule Max-eqI, rule finite-subset[OF least-subset], simp)
  apply (metis max-1)
  by (metis max-2)

have k + 1 = card ((λi. xs ! i) ‘ {0..k})
  by (subst card-image[OF inj-xs], simp)
also have ... ≤ card (least (k+1) (set xs))
  apply (rule card-mono, rule finite-subset[OF least-subset], simp)
  apply (rule image-subsetI)
  apply (simp add: least-def)
  using rank-conv k-bound by simp
finally have card (least (k+1) (set xs)) ≥ k+1 by simp
moreover have card (least (k+1) (set xs)) ≤ k+1
  by (subst card-least, simp, simp)
ultimately have r-2: card (least (k+1) (set xs)) = k+1 by simp

show nth-mset k A = Max (least (k+1) (set-mset A))
  apply (simp add: nth-mset-def xs-def[symmetric] r-1[symmetric])
  by (simp add: A-def)

show card (least (k+1) (set-mset A)) = k+1
  using r-2 by (simp add: A-def)
qed

end

```

11 Counting Polynomials

theory *PolynomialCounting*

```

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  $F\ n = \{x. x \in \text{carrier } (\text{poly-ring } F) \wedge (\text{degree } x < n \vee x = [])\}$ 

```

```

lemma bounded-degree-polynomials-length:
  bounded-degree-polynomials  $F\ n = \{x. x \in \text{carrier } (\text{poly-ring } F) \wedge \text{length } x \leq n\}$ 
apply (rule order-antisym)
apply (rule subsetI, simp add: bounded-degree-polynomials-def)
apply (metis Suc-pred leI less-Suc-eq-0-disj less-Suc-eq-le list.size(3))
apply (rule subsetI, simp add: bounded-degree-polynomials-def)
by (metis diff-less length-greater-0-conv lessI less-imp-diff-less order.not-eq-order-implies-strict)

```

```

lemma fin-degree-bounded:
  assumes ring  $F$ 
  assumes finite (carrier  $F$ )
  shows finite (bounded-degree-polynomials  $F\ n$ )
proof -
  have bounded-degree-polynomials  $F\ n \subseteq \{p. \text{set } p \subseteq \text{carrier } F \wedge \text{length } p \leq n\}$ 
  apply (rule subsetI)
  apply (simp add: bounded-degree-polynomials-length) using assms(1)
  by (meson ring.polynomial-incl univ-poly-carrier)
  thus ?thesis apply (rule finite-subset)
  using assms(2) finite-lists-length-le by auto
qed

```

```

lemma fin-fixed-degree:
  assumes ring  $F$ 
  assumes finite (carrier  $F$ )
  shows finite  $\{p. p \in \text{carrier } (\text{poly-ring } F) \wedge \text{length } p = n\}$ 
proof -
  have  $\{p. p \in \text{carrier } (\text{poly-ring } F) \wedge \text{length } p = n\} \subseteq \text{bounded-degree-polynomials } F\ n$ 
  by (rule subsetI, simp add: bounded-degree-polynomials-length)
  then show ?thesis
  using fin-degree-bounded assms rev-finite-subset by blast
qed

```

```

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

```

proof –

```

define A where A = {p. p ∈ (carrier (poly-ring F)) ∧ length p = Suc n}
have b:A = {p. polynomial_F (carrier F) p ∧ length p = Suc n}
  apply(rule order-antisym, rule subsetI)
  using A-def assms(1) by (simp add: univ-poly-carrier)+
have c:A = {p. set p ⊆ carrier F ∧ hd p ≠ 0_F ∧ length p = Suc n}
  apply (rule order-antisym)
  apply (rule subsetI, simp add:b polynomial-def, force)
  by (rule subsetI, simp add:b polynomial-def)
have d:A = {p. ∃ u v. p=u#v ∧ set v ⊆ carrier F ∧ u ∈ carrier F - {0_F} ∧
length v = n}
  apply(rule order-antisym, rule subsetI)
  apply (simp add:c)
  apply (metis Suc-length-conv hd-Cons-tl length-0-conv list.sel(3) list.set-sel(1)
nat.simps(3)
      order-trans set-subset-Cons subsetD)
  apply (rule subsetI, simp add:c) using assms(2) by force
define B where B = {p. set p ⊆ carrier F ∧ length p = n}
have A = (λ(u,v). u # v) ‘ ((carrier F - {0_F}) × B)
  using d B-def by auto
moreover have inj-on (λ(u,v). u # v) ((carrier F - {0_F}) × B)
  by (auto intro!: inj-onI)
ultimately have card A = card ((carrier F - {0_F}) × B)
  using card-image by meson
moreover have card B = (card (carrier F) ^ n) using B-def
  using card-lists-length-eq assms(2) by blast
ultimately have card A = card (carrier F - {0_F}) * (card (carrier F) ^ n)
  by (simp add: card-cartesian-product)
moreover have card (carrier F - {0_F}) = card (carrier F) - 1
  by (meson assms(1) assms(2) card-Diff-singleton ring.ring-simprules(2))
ultimately show card ({p. p ∈ carrier (poly-ring F) ∧ length p = Suc n}) =
  (card (carrier F) - 1) * (card (carrier F) ^ n) using A-def by simp
qed

```

lemma *fixed-degree-polynomials-count*:

```

assumes ring F
assumes finite (carrier F)
shows card ({p. p ∈ carrier (poly-ring F) ∧ length p = n}) =
  (if n ≥ 1 then (card (carrier F) - 1) * (card (carrier F) ^ (n-1)) else 1)

```

proof –

```

have a: [] ∈ carrier (poly-ring F)
  by (simp add: univ-poly-zero-closed)
show ?thesis
  apply (cases n)
  using assms a apply (simp)
  apply (metis (mono-tags, lifting) One-nat-def empty-Collect-eq is-singletonI'
      is-singleton-altdef mem-Collect-eq)
  using assms by (simp add: nonzero-length-polynomials-count)
qed

```

lemma *bounded-degree-polynomials-count*:
assumes *ring F*
assumes *finite (carrier F)*
shows $\text{card } (\text{bounded-degree-polynomials } F \ n) = \text{card } (\text{carrier } F) ^ n$
proof –
have $0_F \in \text{carrier } F$ **using** *assms(1)* **by** (*simp add: ring.ring-simprules(2)*)
hence $b: \text{card } (\text{carrier } F) > 0$
using *assms(2)* *card-gt-0-iff* **by** *blast*
have $a: \text{bounded-degree-polynomials } F \ n = (\bigcup m \leq n. \{p. p \in \text{carrier } (\text{poly-ring } F) \wedge \text{length } p = m\})$
apply (*simp add: bounded-degree-polynomials-length, rule order-antisym*)
by (*rule subsetI, simp*)
have $\text{card } (\text{bounded-degree-polynomials } F \ n) = (\sum m \leq n. \text{card } \{p. p \in \text{carrier } (\text{poly-ring } F) \wedge \text{length } p = m\})$
apply (*simp only: a*)
apply (*rule card-UN-disjoint, blast*)
using *fin-fixed-degree assms* **apply** *blast*
by *blast*
hence $\text{card } (\text{bounded-degree-polynomials } F \ n) = (\sum m \leq n. \text{if } m \geq 1 \text{ then } (\text{card } (\text{carrier } F) - 1) * \text{card } (\text{carrier } F) ^{(m-1)} \text{ else } 1)$
using *fixed-degree-polynomials-count assms* **by** *fastforce*
moreover **have** $(\sum m \leq n. \text{if } m \geq 1 \text{ then } (\text{card } (\text{carrier } F) - 1) * (\text{card } (\text{carrier } F) ^{(m-1)}) \text{ else } 1) = \text{card } (\text{carrier } F) ^ n$
apply (*induction n, simp, simp add: algebra-simps*) **using** b **by** *force*
ultimately show *?thesis* **by** *auto*
qed

lemma *non-empty-bounded-degree-polynomials*:
assumes *ring F*
shows $\text{bounded-degree-polynomials } F \ k \neq \{\}$
proof –
have $0_{\text{poly-ring } F} \in \text{bounded-degree-polynomials } F \ k$
using *assms*
by (*simp add: bounded-degree-polynomials-def univ-poly-zero univ-poly-zero-closed*)
thus *?thesis* **by** *auto*
qed

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 than 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 $(\text{card } (\text{carrier } F))^n$ 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* $F K p =$
 $(\text{restrict } (\text{ring.eval } F p) K, \lambda k. \text{ring.coeff } F p (k + \text{card } K))$

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

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 $K \subseteq \text{carrier } F$
shows *split-poly* $F K \subseteq \text{bounded-degree-polynomials } F (\text{card } K + n) \subseteq$
 $(K \rightarrow_E \text{carrier } F) \times \{f. \text{range } f \subseteq \text{carrier } F \wedge (\forall k \geq n. f k = \mathbf{0}_F)\}$
apply (*rule image-subsetI*)
apply (*simp add: split-poly-def Pi-def bounded-degree-polynomials-length*)
apply (*rule conjI, rule allI, rule impI*)
apply (*metis assms(1) assms(2) field.is-ring mem-Collect-eq partial-object.select-convs(1)*
 $\text{ring.carrier-is-subring ring.eval-in-carrier ring.polynomial-in-carrier sub-}$
 set-iff
 univ-poly-def)
apply (*rule conjI, rule subsetI*)
apply (*metis (no-types, lifting) assms(1) field.is-ring imageE mem-Collect-eq*
 $\text{partial-object.select-convs(1) ring.carrier-is-subring ring.coeff-in-carrier}$
 $\text{ring.polynomial-in-carrier univ-poly-def}$)
by (*simp add: assms(1) field.is-ring ring.coeff-length*)

lemma *poly-neg-coeff*:

assumes *domain* F
assumes $x \in \text{carrier } (\text{poly-ring } F)$
shows $\text{ring.coeff } F (\ominus_{\text{poly-ring } F} x) k = \ominus_F \text{ring.coeff } F x k$
proof –
interpret *ring poly-ring* F
using *assms cring-def domain.univ-poly-is-ring domain-def ring.carrier-is-subring*

by *blast*
have $\mathbf{0}_{\text{poly-ring } F} = x \ominus_{\text{poly-ring } F} x$ **by** (*metis* *assms*(2) *r-right-minus-eq*)
hence $\text{ring.coeff } F (\mathbf{0}_{\text{poly-ring } F}) k = \text{ring.coeff } F x k \oplus_F \text{ring.coeff } F (\ominus_{\text{poly-ring } F} x) k$
by (*metis* *assms* *cring-def* *domain.univ-poly-a-inv-length* *domain-def* *dual-order.refl* *minus-eq*
 $\text{ring.carrier-is-subring}$ $\text{ring.poly-add-coeff-aux}$ *univ-poly-add*)
thus *?thesis*
by (*metis* *abelian-group.minus-equality* *add.l-inv-ex* *assms*(1) *assms*(2) *cring-def*
 $\text{domain.axioms}(1)$ *is-abelian-group* *mem-Collect-eq* *partial-object.select-convs*(1)
 $\text{ring.carrier-is-subring}$ $\text{ring.coeff.simps}(1)$ $\text{ring.coeff-in-carrier}$ $\text{ring.polynomial-in-carrier}$
 $\text{ring.ring-simprules}(20)$ ring-def *univ-poly-def* *univ-poly-zero*)
qed

lemma *poly-subtract-coeff*:
assumes *domain* *F*
assumes $x \in \text{carrier } (\text{poly-ring } F)$
assumes $y \in \text{carrier } (\text{poly-ring } F)$
shows $\text{ring.coeff } F (x \ominus_{\text{poly-ring } F} y) k = \text{ring.coeff } F x k \ominus_F \text{ring.coeff } F y k$
apply (*simp* *add:a-minus-def* *poly-neg-coeff[symmetric]*)
using *assms* $\text{ring.poly-add-coeff}$
by (*metis* *abelian-group.a-inv-closed* *cring-def* *domain.univ-poly-is-abelian-group* *domain-def*
 poly-neg-coeff $\text{ring.carrier-is-subring}$ $\text{ring.polynomial-incl}$ *univ-poly-add* *univ-poly-carrier*)

lemma *poly-subtract-eval*:
assumes *domain* *F*
assumes $i \in \text{carrier } F$
assumes $x \in \text{carrier } (\text{poly-ring } F)$
assumes $y \in \text{carrier } (\text{poly-ring } F)$
shows $\text{ring.eval } F (x \ominus_{\text{poly-ring } F} y) i = \text{ring.eval } F x i \ominus_F \text{ring.eval } F y i$
proof –
have $\text{subring } (\text{carrier } F) F$
using *assms*(1) *cring-def* *domain-def* $\text{ring.carrier-is-subring}$ **by** *blast*
hence $\text{ring-hom-cring } (\text{poly-ring } F) F (\lambda p. (\text{ring.eval } F p) i)$
by (*simp* *add: assms*(1) *assms*(2) $\text{domain.eval-cring-hom}$)
then show *?thesis* **by** (*meson* $\text{ring-hom-cring.hom-sub}$ *assms*(3) *assms*(4))
qed

lemma *poly-degree-bound-from-coeff*:
assumes *ring* *F*
assumes $x \in \text{carrier } (\text{poly-ring } F)$
assumes $\bigwedge k. k \geq n \implies \text{ring.coeff } F x k = \mathbf{0}_F$
shows $\text{degree } x < n \vee x = \mathbf{0}_{\text{poly-ring } F}$
proof (*rule* *ccontr*)
assume $a: \neg(\text{degree } x < n \vee x = \mathbf{0}_{\text{poly-ring } F})$
hence $b: \text{lead-coeff } x \neq \mathbf{0}_F$

by (metis assms(2) polynomial-def univ-poly-carrier univ-poly-zero)
 hence ring.coeff F x (degree x) $\neq \mathbf{0}_F$
 by (metis a assms(1) ring.lead-coeff-simp univ-poly-zero)
 moreover have degree $x \geq n$ by (meson a not-le)
 ultimately show False using assms(3) by blast
 qed

lemma max-roots:

assumes field R
 assumes $p \in \text{carrier } (\text{poly-ring } R)$
 assumes $K \subseteq \text{carrier } R$
 assumes finite K
 assumes degree $p < \text{card } K$
 assumes $\bigwedge x. x \in K \implies \text{ring.eval } R \ p \ x = \mathbf{0}_R$
 shows $p = \mathbf{0}_{\text{poly-ring } R}$
 proof (rule ccontr)
 assume $p \neq \mathbf{0}_{\text{poly-ring } R}$
 hence $a:p \neq []$ by (simp add: univ-poly-zero)
 have $\bigwedge x. \text{count } (\text{mset-set } K) \ x \leq \text{count } (\text{ring.roots } R \ p) \ x$
 proof -
 fix x
 show $\text{count } (\text{mset-set } K) \ x \leq \text{count } (\text{ring.roots } R \ p) \ x$
 proof (cases $x \in K$)
 case True
 hence ring.is-root $R \ p \ x$ using assms(3) assms(6)
 by (meson a assms(1) field.is-ring ring.is-root-def subsetD)
 hence $x \in \text{set-mset } (\text{ring.roots } R \ p)$
 using assms(2) assms(1) domain.roots-mem-iff-is-root field-def by force
 hence $1 \leq \text{count } (\text{ring.roots } R \ p) \ x$ by simp
 moreover have $\text{count } (\text{mset-set } K) \ x = 1$ using True assms(4) by simp
 ultimately show ?thesis by presburger
 next
 case False
 hence $\text{count } (\text{mset-set } K) \ x = 0$ by simp
 then show ?thesis by presburger
 qed
 qed
 hence $\text{mset-set } K \subseteq \# \text{ ring.roots } R \ p$
 by (simp add: subseteq-mset-def)
 hence $\text{card } K \leq \text{size } (\text{ring.roots } R \ p)$
 by (metis size-mset-mono size-mset-set)
 moreover have $\text{size } (\text{ring.roots } R \ p) \leq \text{degree } p$
 using a field.size-roots-le-degree assms by auto
 ultimately show False using assms(5)
 by (meson leD less-le-trans)
 qed

lemma split-poly-inj:

assumes field F

```

assumes finite K
assumes  $K \subseteq \text{carrier } F$ 
shows inj-on (split-poly F K) (carrier (poly-ring F))
proof
  have ring-F: ring F using assms(1) field.is-ring by blast
  have domain-F: domain F using assms(1) field-def by blast
  fix x
  fix y
  assume  $a1: x \in \text{carrier } (\text{poly-ring } F)$ 
  assume  $a2: y \in \text{carrier } (\text{poly-ring } F)$ 
  assume  $a3: \text{split-poly } F \ K \ x = \text{split-poly } F \ K \ y$ 

  have x-y-carrier:  $x \ominus_{\text{poly-ring } F} y \in \text{carrier } (\text{poly-ring } F)$  using a1 a2
  by (simp add: assms(1) domain.univ-poly-is-ring field.axioms(1) ring.carrier-is-subring

    ring.ring-simprules(4) ring-F)
  have  $\bigwedge k. \text{ring.coeff } F \ x \ (k + \text{card } K) = \text{ring.coeff } F \ y \ (k + \text{card } K)$ 
  using a3 apply (simp add: split-poly-def) by meson
  hence  $\bigwedge k. \text{ring.coeff } F \ (x \ominus_{\text{poly-ring } F} y) \ (k + \text{card } K) = \mathbf{0}_F$ 
  apply (simp add: domain-F a1 a2 poly-subtract-coeff)
  by (meson a2 ring.carrier-is-subring ring.coeff-in-carrier
    ring.polynomial-in-carrier ring.r-right-minus-eq ring-F univ-poly-carrier)
  hence  $\text{degree } (x \ominus_{\text{poly-ring } F} y) < \text{card } K \vee (x \ominus_{\text{poly-ring } F} y) = \mathbf{0}_{\text{poly-ring } F}$ 
  by (metis add.commute le-Suc-ex poly-degree-bound-from-coeff x-y-carrier ring-F)
  moreover have  $\bigwedge k. k \in K \implies \text{ring.eval } F \ x \ k = \text{ring.eval } F \ y \ k$ 
  using a3 apply (simp add: split-poly-def restrict-def) by meson
  hence  $\bigwedge k. k \in K \implies \text{ring.eval } F \ x \ k \ominus_F \text{ring.eval } F \ y \ k = \mathbf{0}_F$ 
  by (metis (no-types, opaque-lifting) a2 assms(3) ring.eval-in-carrier ring.polynomial-incl

    ring.r-right-minus-eq ring-F subsetD univ-poly-carrier)
  hence  $\bigwedge k. k \in K \implies \text{ring.eval } F \ (x \ominus_{\text{poly-ring } F} y) \ k = \mathbf{0}_F$ 
  using domain-F a1 a2 assms(3) poly-subtract-eval by (metis (no-types, opaque-lifting)
subsetD)
  ultimately have  $x \ominus_{\text{poly-ring } F} y = \mathbf{0}_{\text{poly-ring } F}$ 
  using max-roots x-y-carrier assms by blast
  then show  $x = y$ 
  by (meson assms(1) a1 a2 domain.univ-poly-is-ring field-def ring.carrier-is-subring

    ring.r-right-minus-eq ring-F)
qed

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 –
  have a: ring F using assms(1) by (simp add: field.is-ring)

```



```

  show ?A using a bounded-degree-polynomials-count assms by blast
  show ?B using a fin-degree-bounded assms by blast
qed

lemma
  assumes finite (B :: 'b set)
  assumes y ∈ B
  shows
    card-mostly-constant-maps:
    card {f. range f ⊆ B ∧ (∀ x. x ≥ n ⟶ f x = y)} = card B ^ n (is card ?A =
    ?B) and
    finite-mostly-constant-maps:
    finite {f. range f ⊆ B ∧ (∀ x. x ≥ n ⟶ f x = y)}
proof -
  define C where C = {k. k < n} →E B
  define forward where forward = (λ(f :: nat ⇒ 'b). restrict f {k. k < n})
  define backward where backward = (λf k. if k < n then f k else y)

  have forward-inject:inj-on forward ?A
    apply (rule inj-onI, rule ext, simp add:forward-def restrict-def)
    by (metis not-le)

  have forward-image:forward ' ?A ⊆ C
    apply (rule image-subsetI, simp add:forward-def C-def) by blast
  have finite-C:finite C
    by (simp add:C-def finite-PiE assms(1))

  have card-ineq-1: card ?A ≤ card C
    using card-image card-mono forward-inject forward-image finite-C by (metis
    (no-types, lifting))

  show finite ?A
    using inj-on-finite forward-inject forward-image finite-C by blast
  moreover have inj-on backward C
    apply (rule inj-onI, rule ext, simp add:backward-def C-def)
    by (metis (no-types, lifting) PiE-ext mem-Collect-eq)
  moreover have backward ' C ⊆ ?A
    apply (rule image-subsetI, simp add:backward-def C-def)
    apply (rule conjI, rule image-subsetI) apply blast
    by (rule image-subsetI, simp add:assms)
  ultimately have card-ineq-2: card C ≤ card ?A by (metis (no-types, lifting)
  card-image card-mono)

  have card ?A = card C using card-ineq-1 card-ineq-2 by auto
  moreover have card C = card B ^ n using C-def assms(1) by (simp add:
  card-PiE)
  ultimately show card ?A = ?B by auto
qed

```

lemma *split-poly-surj*:
assumes *field* F
assumes *finite* (*carrier* F)
assumes $K \subseteq \text{carrier } F$
shows *split-poly* $F \ K \text{ ‘ bounded-degree-polynomials } F \ (\text{card } K + n) =$
 $(K \rightarrow_E \text{carrier } F) \times \{f. \text{range } f \subseteq \text{carrier } F \wedge (\forall k \geq n. f \ k = \mathbf{0}_F)\}$
(is *split-poly* $F \ K \text{ ‘ ?}A = \text{?}B$ **)**
proof –
define M **where** $M = \text{split-poly } F \ K \text{ ‘ ?}A$
have $a: \mathbf{0}_F \in \text{carrier } F$ **using** *assms*(1)
by (*simp add: field.is-ring ring.ring-simprules*(2))
have $b: \text{finite } K$ **using** *assms*(2) *assms*(3) *finite-subset* **by** *blast*
moreover **have** $?A \subseteq \text{carrier } (\text{poly-ring } F)$
by (*simp add: Collect-mono-iff bounded-degree-polynomials-def*)
ultimately **have** *inj-on* (*split-poly* $F \ K$) $?A$
by (*meson split-poly-inj assms*(1) *assms*(3) *inj-on-subset*)
moreover **have** *finite* $?A$ **using** *finite-poly-count assms*(2) *assms*(1) **by** *blast*
ultimately **have** $\text{card } ?A = \text{card } M$ **by** (*simp add: M-def card-image*)
hence $\text{card } M = \text{card } (\text{carrier } F) \frown (\text{card } K + n)$
using *poly-count assms*(2) *assms*(1) **by** *metis*
moreover **have** $M \subseteq ?B$ **using** *split-poly-image M-def assms* **by** *blast*
moreover **have** $\text{card } ?B = \text{card } (\text{carrier } F) \frown (\text{card } K + n)$
by (*simp add: a assms b card-mostly-constant-maps card-PiE power-add card-cartesian-product*)

moreover **have** *finite* $?B$ **using** *assms*(2) $a \ b$
by (*simp add: finite-mostly-constant-maps finite-PiE*)
ultimately **have** $M = ?B$ **by** (*simp add: card-seteq*)
thus *?thesis* **using** *M-def* **by** *auto*
qed

lemma *inv-subsetI*:
assumes $\bigwedge x. x \in A \implies f \ x \in B \implies x \in C$
shows $f \text{ – ‘ } B \cap A \subseteq C$
using *assms* **by** *force*

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

proof –
define z **where** $z = \text{restrict } f \ K$
define M **where** $M = \{f. \text{range } f \subseteq \text{carrier } F \wedge (\forall k \geq n. f \ k = \mathbf{0}_F)\}$

have $a: \mathbf{0}_F \in \text{carrier } F$ **using** *assms*(1)

```

    by (simp add: field.is-ring ring.ring-simprules(2))

  have finite K using assms(2) assms(3) finite-subset by blast
  hence inj-on-bounded: inj-on (split-poly F K) (bounded-degree-polynomials F
(card K + n))
  using split-poly-inj assms(1) assms(3) inj-on-subset bounded-degree-polynomials-length

    by (metis (mono-tags) Collect-subset)
  moreover have  $z \in (K \rightarrow_E \text{carrier } F)$  apply (simp add: z-def)
    using assms by blast
  hence  $\{z\} \times M \subseteq \text{split-poly } F K \text{ ' (bounded-degree-polynomials } F (\text{card } K + n))$ 
    apply (simp add: split-poly-surj assms M-def z-def)
    by fastforce
  ultimately have  $\text{card } ((\text{split-poly } F K - ' (\{z\} \times M)) \cap \text{bounded-degree-polynomials } F (\text{card } K + n))$ 
    =  $\text{card } (\{z\} \times M)$  by (meson card-vimage-inj-on)
  moreover have  $(\text{split-poly } F K - ' (\{z\} \times M)) \cap \text{bounded-degree-polynomials } F (\text{card } K + n) \subseteq ?A$ 
    apply (rule inv-subsetI)
    apply (simp add: split-poly-def z-def restrict-def)
    by (meson)
  moreover have finite ?A by (simp add: finite-poly-count assms)
  ultimately have card-ineq-1:  $\text{card } (\{z\} \times M) \leq \text{card } ?A$ 
    by (metis (mono-tags, lifting) card-mono)

  have split-poly F K ' ?A  $\subseteq \{z\} \times M$ 
    apply (rule image-subsetI)
    apply (simp add: split-poly-def z-def M-def)
    apply (rule conjI, fastforce)
    apply (simp add: bounded-degree-polynomials-length)
    apply (rule conjI)
    apply (meson assms(1) field.is-ring image-subsetI ring.coeff-in-carrier ring.polynomial-incl
      univ-poly-carrier)
    by (simp add: assms(1) field.is-ring ring.coeff-length)
  moreover have inj-on (split-poly F K) ?A using inj-on-subset inj-on-bounded
  by fastforce
  moreover have finite  $(\{z\} \times M)$  by (simp add: M-def finite-mostly-constant-maps
    assms(2) a)
  ultimately have card-ineq-2:  $\text{card } ?A \leq \text{card } (\{z\} \times M)$  by (meson card-inj-on-le)

  have  $\text{card } ?A = \text{card } (\{z\} \times M)$  using card-ineq-1 card-ineq-2 by auto
  moreover have  $\text{card } (\{z\} \times M) = \text{card } (\text{carrier } F) ^ n$ 
    by (simp add: card-cartesian-product M-def a card-mostly-constant-maps assms(2)
  )
  ultimately show ?thesis by presburger
qed

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 ∈ extensional I then $\prod i \in I. (pmf (M i) (x i) \text{ else } 0)$ else 0)*

by (*simp add:prod-pmf-def pmf-Pi[OF assms(1)] extensional-def*)

lemma *set-prod-pmf*:

assumes *finite I*

shows *set-pmf (prod-pmf I M) = PiE I (set-pmf ∘ M)*

apply (*simp add:set-pmf-eq pmf-prod-pmf[OF assms(1)] prod-zero-iff[OF assms(1)]*)

apply (*simp add:set-pmf-iff[symmetric] PiE-def Pi-def*)

by *blast*

lemma *set-pmf-iff'*: *x ∉ set-pmf M ⟷ pmf M x = 0*

using *set-pmf-iff* **by** *metis*

lemma *prob-prod-pmf*:

assumes *finite I*

shows *measure (measure-pmf (prod-pmf I M)) (Pi I A) = ($\prod i \in I. \text{measure } (M i) (A i)$)*

apply (*simp add:prod-pmf-def*)

by (*subst measure-Pi-pmf-Pi[OF assms(1)], simp*)

lemma *prob-prod-pmf'*:

assumes *finite I*

assumes *J ⊆ I*

shows *measure (measure-pmf (prod-pmf I M)) (Pi J A) = ($\prod i \in J. \text{measure } (M i) (A i)$)*

proof –

have *a:Pi J A = Pi I (λi. if i ∈ J then A i else UNIV)*

apply (*simp add:Pi-def*)

apply (*rule Collect-cong*)

using *assms(2)* **by** *blast*

show *?thesis*

apply (*simp add:if-distrib a prob-prod-pmf[OF assms(1)] prod.If-cases[OF assms(1)]*)

apply (rule arg-cong2[**where** $f=prod$], simp)
using assms(2) **by** blast
qed

lemma prob-prod-pmf-slice:
assumes finite I
assumes $i \in I$
shows $measure (measure-pmf (prod-pmf I M)) \{\omega. P (\omega i)\} = measure (M i) \{\omega. P \omega\}$
using prob-prod-pmf'[OF assms(1), **where** $J=\{i\}$ **and** $M=M$ **and** $A=\lambda-. Collect P$]
by (simp add:assms Pi-def)

lemma range-inter: $range ((\cap) F) = Pow F$
apply (rule order-antisym, rule subsetI, simp add:image-def, blast)
by (rule subsetI, simp add:image-def, blast)

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 –
have $sigma-sets M ((\lambda k. \{k\}) ' M) = Pow M$
using assms sigma-sets-singletons **by** auto
hence $Pow M \subseteq sigma-sets M (insert \{\} ((\lambda k. \{k\}) ' M))$
by (metis sigma-sets-subseteq subset-insertI)
moreover have $(insert \{\} ((\lambda k. \{k\}) ' M)) \subseteq Pow M$ **by** blast
hence $sigma-sets M (insert \{\} ((\lambda k. \{k\}) ' M)) \subseteq Pow M$
by (meson sigma-algebra.sigma-sets-subset sigma-algebra-Pow)
ultimately show ?thesis **by** force
qed

lemma indep-vars-pmf:
assumes $\bigwedge a J. J \subseteq I \implies finite J \implies$
 $\mathcal{P}(\omega \text{ in } measure-pmf M. \forall i \in J. X i \omega = a i) = (\prod i \in J. \mathcal{P}(\omega \text{ in } measure-pmf M. X i \omega = a i))$
shows $prob-space.indep-vars (measure-pmf M) (\lambda i. measure-pmf (M' i)) X I$
proof –
define G **where** $G = (\lambda i. \{\{\}\} \cup (\lambda x. \{x\}) ' (X i ' set-pmf M))$
define F **where** $F = (\lambda i. \{X i -' a \cap set-pmf M | a. a \in G i\})$

have $g: \bigwedge i. i \in I \implies sigma-sets (X i ' set-pmf M) (G i) = Pow (X i ' set-pmf M)$
by (simp add:G-def, metis countable-image countable-set-pmf sigma-sets-singletons-and-empty)

have $e: \bigwedge i. i \in I \implies F i \subseteq Pow (set-pmf M)$
by (simp add:F-def, rule subsetI, simp, blast)

```

have a:  $distr (restrict\ space (measure\ pmf\ M) (set\ pmf\ M)) (measure\ pmf\ M) id$ 
=  $measure\ pmf\ M$ 
  apply (rule  $measure\ eqI$ ,  $simp$ ,  $simp$ )
  apply (subst  $emeasure\ distr$ )
  apply (simp add:  $measurable\ def\ sets\ restrict\ space$ )
  apply blast
  apply simp
  apply (simp add:  $emeasure\ restrict\ space$ )
  by (metis  $emeasure\ Int\ set\ pmf$ )

have b:  $prob\ space (restrict\ space (measure\ pmf\ M) (set\ pmf\ M))$ 
  apply (rule  $prob\ spaceI$ )
  apply simp
  apply (subst  $emeasure\ restrict\ space$ ,  $simp$ ,  $simp$ )
  using  $emeasure\ pmf$  by blast

have d:  $\bigwedge i. i \in I \implies \{u. \exists A. u = X\ i - ' A \cap set\ pmf\ M\} = sigma\ sets (set\ pmf\ M) (F\ i)$ 
proof -
  fix i
  assume d1:  $i \in I$ 
  have d2:  $\bigwedge A. X\ i - ' A \cap set\ pmf\ M = X\ i - ' (A \cap X\ i - ' set\ pmf\ M) \cap set\ pmf\ M$ 
  M
    apply (rule  $order\ antisym$ )
    by (rule  $subsetI$ ,  $simp$ ) +
  show  $\{u. \exists A. u = X\ i - ' A \cap set\ pmf\ M\} = sigma\ sets (set\ pmf\ M) (F\ i)$ 
    apply (simp add:  $F\ def$ )
    apply (subst  $sigma\ sets\ vimage\ commute[symmetric, where\ \Omega' = X\ i - ' set\ pmf\ M]$ ,  $blast$ )
    using d1 apply (simp add:  $g$ )
    apply (rule  $order\ antisym$ )
    apply (rule  $subsetI$ ,  $simp$ ,  $meson\ inf\ le2\ d2$ )
    by (rule  $subsetI$ ,  $simp$ ,  $blast$ )
qed

have h:  $\bigwedge J\ A. J \subseteq I \implies J \neq \{\} \implies finite\ J \implies A \in Pi\ J\ F \implies$ 
 $Sigma\ Algebra.measure (restrict\ space (measure\ pmf\ M) (set\ pmf\ M))$ 
 $(\bigcap (A - ' J)) =$ 
 $(\prod j \in J. Sigma\ Algebra.measure (restrict\ space (measure\ pmf\ M)$ 
 $(set\ pmf\ M)) (A\ j)) (A\ j))$ 
proof -
  fix J A
  assume h1:  $J \subseteq I$ 
  assume h2:  $J \neq \{\}$ 
  assume h3:  $finite\ J$ 
  assume h4:  $A \in Pi\ J\ F$ 

  have h5:  $\bigwedge j. j \in J \implies A\ j \subseteq set\ pmf\ M$ 
  by (metis  $PiE\ PowD\ h1\ subsetD\ e\ h4$ )

```

```

obtain  $a$  where  $h6: \bigwedge j. j \in J \implies A\ j = X\ j - 'a\ j \cap \text{set-pmf } M \wedge a\ j \in G\ j$ 
using  $h4$  by (simp add:Pi-def F-def, metis)

show  $\text{Sigma-Algebra.measure } (\text{restrict-space } (\text{measure-pmf } M) (\text{set-pmf } M)) (\bigcap (A - 'J)) =$ 
 $(\prod_{j \in J. \text{Sigma-Algebra.measure } (\text{restrict-space } (\text{measure-pmf } M) (\text{set-pmf } M)) (A\ j))$ 
proof (cases  $\exists j \in J. A\ j = \{\}$ )
  case True
    hence  $\bigcap (A - 'J) = \{\}$  by blast
    then show ?thesis
      using  $h3$  True apply simp
      by (metis measure-empty)
  next
    case False
      then have  $\bigwedge j. j \in J \implies a\ j \neq \{\}$  using  $h6$  by auto
      hence  $\bigwedge j. j \in J \implies a\ j \in (\lambda x. \{x\}) - 'X\ j - \text{set-pmf } M$  using  $h6$  by (simp
add:G-def)
      then obtain  $b$  where  $h7: \bigwedge j. j \in J \implies a\ j = \{b\ j\}$  by (simp add:image-def, metis)

      have  $\text{Sigma-Algebra.measure } (\text{restrict-space } (\text{measure-pmf } M) (\text{set-pmf } M)) (\bigcap (A - 'J)) =$ 
 $\text{Sigma-Algebra.measure } (\text{measure-pmf } M) (\bigcap_{j \in J. A\ j})$ 
      apply (subst measure-restrict-space, simp)
      using  $h5\ h2$  apply blast
      by simp
      also have  $\dots = \text{Sigma-Algebra.measure } (\text{measure-pmf } M) (\{\omega. \forall j \in J. X\ j\ \omega = b\ j\})$ 
      using  $h2\ h6\ h7$  apply (simp add:vimage-def measure-Int-set-pmf)
      by (rule arg-cong2 [where f=measure], simp, blast)
      also have  $\dots = (\prod_{j \in J. \text{Sigma-Algebra.measure } (\text{measure-pmf } M) (A\ j))$ 
      using  $h6\ h7\ h2\ \text{assms}(1)[OF\ h1\ h3]$  by (simp add:vimage-def measure-Int-set-pmf)
      also have  $\dots = (\prod_{j \in J. \text{Sigma-Algebra.measure } (\text{restrict-space } (\text{measure-pmf } M) (\text{set-pmf } M)) (A\ j))$ 
      by (rule prod.cong, simp, subst measure-restrict-space, simp, metis h5, simp)
      finally show ?thesis by blast
    qed
  qed

have  $i: \bigwedge i. i \in I \implies \text{Int-stable } (F\ i)$ 
proof (rule Int-stableI)
  fix  $i\ a\ b$ 
  assume  $i \in I$ 
  assume  $a \in F\ i$ 
  moreover assume  $b \in F\ i$ 
  ultimately show  $a \cap b \in (F\ i)$ 
  apply (cases  $a \cap b = \{\}$ , simp add:F-def G-def, blast)

```

```

    by (simp add:F-def G-def, blast)
qed

  have c: prob-space.indep-sets (restrict-space (measure-pmf M) (set-pmf M)) (λi.
{u. ∃ A. u = X i - 'A ∩ set-pmf M}) I
    apply (simp add: d cong:prob-space.indep-sets-cong[OF b])
    apply (rule prob-space.indep-sets-sigma[where M=restrict-space (measure-pmf
M) (set-pmf M), simplified])
    apply (metis b)
    apply (subst prob-space.indep-sets-def, metis b, simp add:sets-restrict-space
range-inter e)
    apply (metis h)
    by (metis i)

show ?thesis
  apply (subst a [symmetric])
  apply (rule indep-vars-distr)
  apply (simp add:measurable-def sets-restrict-space)
  apply blast
  apply simp
  apply simp
  apply (subst prob-space.indep-vars-def2)
  apply (metis b)
  apply (simp add:measurable-def sets-restrict-space range-inter)
  by (metis c, metis b)
qed

lemma indep-vars-restrict:
  fixes M :: 'a ⇒ 'b pmf
  fixes J :: 'c set
  assumes disjoint-family-on f J
  assumes J ≠ {}
  assumes ∧i. i ∈ J ⇒ f i ⊆ I
  assumes finite I
  shows prob-space.indep-vars (measure-pmf (prod-pmf I M)) (λi. measure-pmf
(prod-pmf (f i) M)) (λi ω. restrict ω (f i)) J
proof (rule indep-vars-pmf[simplified])
  fix a :: 'c ⇒ 'a ⇒ 'b
  fix J'
  assume e:J' ⊆ J
  assume c:finite J'
  show measure-pmf.prob (prod-pmf I M) {ω. ∀ i∈J'. restrict ω (f i) = a i} =
    (∏ i∈J'. measure-pmf.prob (prod-pmf I M) {ω. restrict ω (f i) = a i})
  proof (cases ∀ j ∈ J'. a j ∈ extensional (f j))
    case True
    define b where b = (λi. if i ∈ (⋃ (f ' J')) then a (THE j. i ∈ f j ∧ j ∈ J') i
else undefined)
    have b-def:∧i. i ∈ J' ⇒ a i = restrict b (f i)
    proof -

```



```

fix i
assume b-def-1: i ∈ J'
have b-def-2:  $\bigwedge x. x \in f i \implies i = (THE j. x \in f j \wedge j \in J')$ 
  using disjoint-family-on-mono[OF e assms(1)] b-def-1
  apply (simp add: disjoint-family-on-def)
  by (metis (mono-tags, lifting) IntI empty-iff the-equality)
show a i = restrict b (f i)
  apply (rule extensionalityI[where A = f i]) using b-def-1 True apply blast
  apply (rule restrict-extensional)
  apply (simp add: restrict-apply' b-def b-def-2[symmetric])
  using b-def-1 by force
qed
have a:  $\{\omega. \forall i \in J'. \text{restrict } \omega (f i) = a i\} = Pi (\bigcup (f ' J')) (\lambda i. \{b i\})$ 
  apply (simp add: b-def)
  apply (rule order-antisym)
  apply (rule subsetI, simp add: Pi-def, metis restrict-apply')
  by (rule subsetI, simp add: Pi-def, meson assms(3) e restrict-ext singletonD
subsetD)
have b:  $\bigwedge i. i \in J' \implies \{\omega. \text{restrict } \omega (f i) = a i\} = Pi (f i) (\lambda i. \{b i\})$ 
  apply (simp add: b-def)
  apply (rule order-antisym)
  apply (rule subsetI, simp add: Pi-def, metis restrict-apply')
  by (rule subsetI, simp add: Pi-def, meson assms(3) e restrict-ext singletonD
subsetD)
show ?thesis
  apply (simp add: a b)
  apply (subst prob-prod-pmf'[OF assms(4)], meson UN-least e in-mono assms(3))
  apply (subst prod.UNION-disjoint, metis c)
  apply (metis in-mono e assms(3) assms(4) finite-subset)
  apply (metis e disjoint-family-on-def assms(1) subset-eq)
  apply (rule prod.cong, simp)
  apply (subst prob-prod-pmf'[OF assms(4)]) using e assms(3) apply blast
  by simp
next
case False
then obtain j where j-def:  $j \in J'$  and  $a j \notin \text{extensional } (f j)$  by blast
hence  $\bigwedge \omega. \text{restrict } \omega (f j) \neq a j$  by (metis restrict-extensional)
then show ?thesis
  by (metis (mono-tags, lifting) Collect-empty-eq j-def c measure-empty prod-zero-iff)
qed
qed

lemma indep-vars-restrict-intro:
  fixes M :: 'a  $\Rightarrow$  'b pmf
  fixes J :: 'c set
  assumes  $\bigwedge i. i \in J \implies X i \omega = X i (\text{restrict } \omega (f i))$ 
  assumes disjoint-family-on f J
  assumes  $J \neq \{\}$ 
  assumes  $\bigwedge i. i \in J \implies f i \subseteq I$ 

```

```

assumes finite I
assumes  $\bigwedge \omega \ i. \ i \in J \implies X \ i \ \omega \in \text{space} \ (M' \ i)$ 
shows prob-space.indep-vars (measure-pmf (prod-pmf I M)) M' ( $\lambda i \ \omega. \ X \ i \ \omega$ ) J
proof –
  have prob-space.indep-vars (measure-pmf (prod-pmf I M)) M' ( $\lambda i \ \omega. \ X \ i \ (\text{restrict } \omega \ (f \ i))$ ) J (is ?A)
    apply (rule prob-space.indep-vars-compose2 [where  $X = \lambda i \ \omega. \ \text{restrict } \omega \ (f \ i)$ ])
    apply (metis prob-space-measure-pmf)
    apply (rule indep-vars-restrict, metis assms(2), metis assms(3), metis assms(4),
  metis assms(5))
    apply simp using assms(6) by blast
  moreover have  $?A = ?thesis$ 
    apply (rule prob-space.indep-vars-cong, metis prob-space-measure-pmf, simp)
    by (rule ext, metis assms(1), simp)
  ultimately show  $?thesis$  by blast
qed

```

```

lemma has-bochner-integral-prod-pmfI:
  fixes  $f :: 'a \Rightarrow 'b \Rightarrow ('c :: \{\text{second-countable-topology}, \text{banach}, \text{real-normed-field}\})$ 
  assumes finite I
  assumes  $\bigwedge i. \ i \in I \implies \text{has-bochner-integral} \ (\text{measure-pmf} \ (M \ i)) \ (f \ i) \ (r \ i)$ 
  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 –
  define  $M'$  where  $M' = (\lambda i. \ \text{if } i \in I \text{ then } \text{restrict-space} \ (\text{measure-pmf} \ (M \ i)) \ (\text{set-pmf} \ (M \ i)) \text{ else } \text{count-space} \ \{\text{undefined}\})$ 

  have  $a: \bigwedge i. \ i \in I \implies \text{finite-measure} \ (\text{restrict-space} \ (\text{measure-pmf} \ (M \ i)) \ (\text{set-pmf} \ (M \ i)))$ 
    apply (rule finite-measureI)
    by (simp add:emeasure-restrict-space)

```

```

interpret product-sigma-finite M'
  apply (simp add:product-sigma-finite-def M'-def)
  by (metis a finite-measure.axioms(1) finite.emptyI finite-insert sigma-finite-measure-count-space-finite)

```

```

have  $\bigwedge i. \ i \in I \implies \text{has-bochner-integral} \ (M' \ i) \ (f \ i) \ (r \ i)$ 
  apply (simp add:M'-def has-bochner-integral-restrict-space)
  apply (rule has-bochner-integralI-AE [OF assms(2)], simp, simp)
  by (subst AE-measure-pmf-iff, simp)

```

```

hence  $b: \text{has-bochner-integral} \ (PiM \ I \ M') \ (\lambda x. \ (\prod i \in I. \ f \ i \ (x \ i))) \ (\prod i \in I. \ r \ i)$ 
  apply (subst has-bochner-integral-iff)
  apply (rule conjI)
  apply (rule product-integrable-prod [OF assms(1)])
  apply (simp add: has-bochner-integral-iff)
  apply (subst product-integral-prod [OF assms(1)])
  apply (simp add: has-bochner-integral-iff)
  apply (rule prod.cong, simp)

```

```

by (simp add: has-bochner-integral-iff)

have d:sets (PiM I M') = Pow (PiE I (set-pmf ∘ M))
  apply (simp add:sets-PiM M'-def comp-def cong:PiM-cong)
  apply (rule order-antisym)
  apply (rule subsetI)
  apply (simp)
  apply (rule sigma-sets-into-sp [where A=prod-algebra I (λx. restrict-space
(measure-pmf (M x)) (set-pmf (M x))))))
  apply (metis (mono-tags, lifting) prod-algebra-sets-into-space space-restrict-space
PiE-cong UNIV-I sets-measure-pmf space-restrict-space2)
  apply simp
  apply (subst sigma-sets-singletons[symmetric])
  apply (rule countable-PiE, metis assms(1), metis countable-set-pmf)
  apply (rule sigma-sets-subseteq)
  apply (rule image-subsetI)
  apply (subst PiE-singleton[symmetric, where A=I], simp add:PiE-def)
  apply (rule prod-algebraI-finite, metis assms(1))
  apply (simp add:sets-restrict-space PiE-iff image-def)
by blast

have c:PiM I M' = restrict-space (measure-pmf (prod-pmf I M)) (PiE I (set-pmf
∘ M))
  apply (rule measure-eqI-countable[where A=PiE I (set-pmf ∘ M)])
  apply (metis d)
  apply (simp add:sets-restrict-space image-def, fastforce)
  apply (rule countable-PiE, metis assms(1), simp add:comp-def)
  apply (subst PiE-singleton[symmetric, where A=I], simp add:PiE-def)
  apply (subst emeasure-PiM, metis assms(1), simp add:M'-def sets-restrict-space,
fastforce)
  apply (subst emeasure-restrict-space, simp, simp)
  apply (simp add:emeasure-pmf-single pmf-prod-pmf[OF assms(1)] PiE-def
prod-ennreal[symmetric] M'-def)
  apply (rule prod.cong, simp)
  apply (subst emeasure-restrict-space, simp, simp add:Pi-iff)
by (simp add:emeasure-pmf-single)

have a:has-bochner-integral (prod-pmf I M) (λx. indicator (PiE I (set-pmf ∘
M)) x *R (∏ i ∈ I. f i (x i))) (∏ i ∈ I. r i)
  apply (subst Lebesgue-Measure.has-bochner-integral-restrict-space[symmetric],
simp)
  by (subst c[symmetric], metis b)

have (λx. ∏ i ∈ I. f i (x i)) ∈ borel-measurable (prod-pmf I M)
  by simp
show has-bochner-integral (prod-pmf I M) (λx. (∏ i ∈ I. f i (x i))) (∏ i ∈ I. r
i)
  apply (rule has-bochner-integralI-AE[OF a], simp)
  apply (subst AE-measure-pmf-iff)

```

using *assms* by (*simp add:set-prod-pmf*)
qed

lemma

fixes $f :: 'a \Rightarrow 'b \Rightarrow ('c :: \{\text{second-countable-topology}, \text{banach}, \text{real-normed-field}\})$
 assumes *finite I*
 assumes $\bigwedge i. i \in I \implies \text{integrable } (\text{measure-pmf } (M\ i))\ (f\ i)$
 shows *prod-pmf-integrable*: $\text{integrable } (\text{prod-pmf } I\ M)\ (\lambda x. (\prod i \in I. f\ i\ (x\ i)))$
 (is ?A) and
 $\text{prod-pmf-integral: } \text{integral}^L\ (\text{prod-pmf } I\ M)\ (\lambda x. (\prod i \in I. f\ i\ (x\ i))) =$
 $(\prod i \in I. \text{integral}^L\ (M\ i)\ (f\ i))$ (is ?B)
proof –
 have $a: \text{has-bochner-integral } (\text{prod-pmf } I\ M)\ (\lambda x. (\prod i \in I. f\ i\ (x\ i)))\ (\prod i \in I.$
 $\text{integral}^L\ (M\ i)\ (f\ i))$
 apply (*rule has-bochner-integral-prod-pmfI[OF assms(1)]*)
 by (*rule has-bochner-integral-integrable[OF assms(2)], simp*)
 show ?A using *a has-bochner-integral-iff* by blast
 show ?B using *a has-bochner-integral-iff* by blast
 qed

lemma *has-bochner-integral-prod-pmf-sliceI*:

fixes $f :: 'a \Rightarrow ('b :: \{\text{second-countable-topology}, \text{banach}, \text{real-normed-field}\})$
 assumes *finite I*
 assumes $i \in I$
 assumes $\text{has-bochner-integral } (\text{measure-pmf } (M\ i))\ (f)\ r$
 shows $\text{has-bochner-integral } (\text{prod-pmf } I\ M)\ (\lambda x. (f\ (x\ i)))\ r$
proof –
 define g where $g = (\lambda j\ \omega. \text{if } j = i \text{ then } f\ \omega \text{ else } 1)$

 have $b: \bigwedge M. \text{has-bochner-integral } (\text{measure-pmf } M)\ (\lambda \omega. 1 :: 'b)\ 1$
 apply (*subst has-bochner-integral-iff, rule conjI, simp*)
 by (*subst lebesgue-integral-const, simp*)

have $a: \bigwedge j. j \in I \implies \text{has-bochner-integral } (\text{measure-pmf } (M\ j))\ (g\ j)\ (\text{if } j = i$
 then r else 1)
 using *assms(3)* by (*simp add:g-def b*)
 have $\text{has-bochner-integral } (\text{prod-pmf } I\ M)\ (\lambda x. (\prod j \in I. g\ j\ (x\ j)))\ (\prod j \in I. \text{if}$
 $j = i \text{ then } r \text{ else } 1)$
 by (*rule has-bochner-integral-prod-pmfI[OF assms(1)], metis a*)
 thus ?thesis
 using *assms(2)* by (*simp add:g-def prod.If-cases[OF assms(1)]*)
 qed

lemma

fixes $f :: 'a \Rightarrow ('b :: \{\text{second-countable-topology}, \text{banach}, \text{real-normed-field}\})$
 assumes *finite I*
 assumes $i \in I$
 assumes $\text{integrable } (\text{measure-pmf } (M\ i))\ f$
 shows *integrable-prod-pmf-slice*: $\text{integrable } (\text{prod-pmf } I\ M)\ (\lambda x. (f\ (x\ i)))$ (is ?A)

and

integral-prod-pmf-slice: $\text{integral}^L (\text{prod-pmf } I \ M) (\lambda x. (f \ (x \ i))) = \text{integral}^L (M \ i) \ f$ (is ?B)

proof –

have *a*:*has-bochner-integral* (*prod-pmf* *I* *M*) ($\lambda x. (f \ (x \ i))$) (*integral*^{*L*} (*M* *i*) *f*)
apply (*rule* *has-bochner-integral-prod-pmf-sliceI*[*OF* *assms*(1) *assms*(2)])
using *assms*(3) **by** (*simp* *add*: *has-bochner-integral-iff*)
show ?*A* **using** *a* *has-bochner-integral-iff* **by** *blast*
show ?*B* **using** *a* *has-bochner-integral-iff* **by** *blast*

qed

lemma *variance-prod-pmf-slice*:

fixes *f* :: 'a \Rightarrow real

assumes *i* $\in I$ *finite* *I*

assumes *integrable* (*measure-pmf* (*M* *i*)) ($\lambda \omega. f \ \omega^{\wedge 2}$)

shows *prob-space.variance* (*prod-pmf* *I* *M*) ($\lambda \omega. f \ (\omega \ i)$) = *prob-space.variance* (*M* *i*) *f*

proof –

have *a*:*integrable* (*measure-pmf* (*M* *i*)) *f*
apply (*rule* *measure-pmf.square-integrable-imp-integrable*)
using *assms*(3) **by** *auto*

show ?*thesis*

apply (*subst* *measure-pmf.variance-eq*)
apply (*rule* *integrable-prod-pmf-slice*[*OF* *assms*(2) *assms*(1)], *metis* *a*)
apply (*rule* *integrable-prod-pmf-slice*[*OF* *assms*(2) *assms*(1)], *metis* *assms*(3))
apply (*subst* *measure-pmf.variance-eq*[*OF* *a* *assms*(3)])
apply (*subst* *integral-prod-pmf-slice*[*OF* *assms*(2) *assms*(1)], *metis* *assms*(3))
apply (*subst* *integral-prod-pmf-slice*[*OF* *assms*(2) *assms*(1)], *metis* *a*)
by *simp*

qed

lemma *PiE-default-undefined-eq*: *PiE-dflt* *I* *undefined* *M* = *PiE* *I* *M*

apply (*rule* *set-eqI*)

apply (*simp* *add*:*PiE-dflt-def* *PiE-def* *extensional-def* *Pi-def*) **by** *blast*

lemma *pmf-of-set-prod*:

assumes *finite* *I*

assumes $\bigwedge x. x \in I \Longrightarrow \text{finite} \ (M \ x)$

assumes $\bigwedge x. x \in I \Longrightarrow M \ x \neq \{\}$

shows *pmf-of-set* (*PiE* *I* *M*) = *prod-pmf* *I* ($\lambda i. \text{pmf-of-set} \ (M \ i)$)

by (*simp* *add*:*prod-pmf-def* *PiE-default-undefined-eq* *Pi-pmf-of-set*[*OF* *assms*(1) *assms*(2) *assms*(3)])

lemma *extensionality-iff*:

assumes *f* \in *extensional* *I*

```

shows (( $\lambda i \in I. g\ i$ ) =  $f$ ) = ( $\forall i \in I. g\ i = f\ i$ )
using assms apply (simp add:extensional-def restrict-def) by auto

lemma of-bool-prod:
  assumes finite I
  shows of-bool ( $\forall i \in I. P\ i$ ) = ( $\prod i \in I. (of\text{-}bool\ (P\ i) :: 'a :: field)$ )
  using assms by (induction I rule:finite-induct, simp, simp)

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  $\gg$  ( $\lambda x. return\text{-}pmf\ (\lambda i \in I. f\ (x\ i))$ ) = prod-pmf I ( $\lambda i. M\ i \gg (\lambda x. return\text{-}pmf\ (f\ x))$ )
proof (rule pmf-eqI)
  fix i :: 'a  $\Rightarrow$  'c

  have a: $\bigwedge x. i \in extensional\ I \implies (of\text{-}bool\ ((\lambda j \in I. f\ (x\ j)) = i) :: real) = (\prod j \in I. of\text{-}bool\ (f\ (x\ j) = i\ j))$ 
  apply (subst extensionality-iff, simp)
  by (rule of-bool-prod[OF assms(1)])

  have b: $\bigwedge x. i \notin extensional\ I \implies of\text{-}bool\ ((\lambda j \in I. f\ (x\ j)) = i) = 0$ 
  by auto

  show pmf (prod-pmf I M  $\gg$  ( $\lambda x. return\text{-}pmf\ (\lambda i \in I. f\ (x\ i))$ )) i = pmf (prod-pmf I ( $\lambda i. M\ i \gg (\lambda x. return\text{-}pmf\ (f\ x))$ )) i
  apply (subst pmf-bind)
  apply (subst pmf-prod-pmf) defer
  apply (subst pmf-bind)
  apply (simp add:indicator-def)
  apply (rule conjI, rule impI)
  apply (subst a, simp)
  apply (subst prod-pmf-integral[OF assms(1)])
  apply (rule finite-measure.integrable-const-bound[where B=1], simp, simp, simp, simp)
  by (simp add:b, metis assms(1))
qed

lemma pair-pmfI:
  A  $\gg$  ( $\lambda a. B \gg (\lambda b. return\text{-}pmf\ (f\ a\ b))$ ) = pair-pmf A B  $\gg$  ( $\lambda (a,b). return\text{-}pmf\ (f\ a\ b)$ )
  apply (simp add:pair-pmf-def)
  apply (subst bind-assoc-pmf)
  apply (subst bind-assoc-pmf)
  by (simp add:bind-return-pmf)

lemma pmf-pair':

```

$\text{pmf } (\text{pair-pmf } M \ N) \ x = \text{pmf } M \ (\text{fst } x) * \text{pmf } N \ (\text{snd } x)$
by (*cases x,simp add:pmf-pair*)

lemma *pair-pmf-ptw*:

assumes *finite I*

shows $\text{pair-pmf } (\text{prod-pmf } I \ A :: (('i \Rightarrow 'a) \ \text{pmf})) \ (\text{prod-pmf } I \ B :: (('i \Rightarrow 'b) \ \text{pmf})) =$

$\text{prod-pmf } I \ (\lambda i. \ \text{pair-pmf } (A \ i) \ (B \ i)) \gg=$
 $(\lambda f. \ \text{return-pmf } (\text{restrict } (\text{fst} \circ f) \ I, \ \text{restrict } (\text{snd} \circ f) \ I))$
(is ?lhs = ?rhs)

proof –

define *h* **where** $h = (\lambda f \ x.$

if $x \in I$ **then**

f *x*

else (

if $(f \ x) = \text{undefined}$ **then**

$(\text{undefined} :: 'a, \text{undefined} :: 'b)$

else (

if $(f \ x) = (\text{undefined}, \text{undefined})$ **then**

undefined

else

f *x*))

have *h-h-id*: $\bigwedge f. \ h \ (h \ f) = f$

apply (*rule ext*)

by (*simp add:h-def*)

have *b*: $\bigwedge i \ g. \ i \in I \implies h \ g \ i = g \ i$

by (*simp add:h-def*)

have *a*: *inj* $(\lambda f. \ (\text{fst} \circ h \ f, \ \text{snd} \circ h \ f))$

proof (*rule injI*)

fix *x y*

assume $(\text{fst} \circ h \ x, \ \text{snd} \circ h \ x) = (\text{fst} \circ h \ y, \ \text{snd} \circ h \ y)$

hence *a1*: $h \ x = h \ y$

by (*simp, metis convol-expand-snd*)

show $x = y$

apply (*rule ext*)

using *a1* **apply** (*simp add:h-def*)

by (*metis (no-types, opaque-lifting)*)

qed

have *c*: $\bigwedge g. \ (\text{fst} \circ h \ g \in \text{extensional } I \wedge \text{snd} \circ h \ g \in \text{extensional } I) = (g \in \text{extensional } I)$

apply (*rule order-antisym*)

apply (*simp add:h-def extensional-def*)

apply (*metis prod.collapse*)

by (*simp add:h-def extensional-def*)

```

have pair-pmf (prod-pmf I A :: (('i ⇒ 'a) pmf)) (prod-pmf I B :: (('i ⇒ 'b)
pmf)) = prod-pmf I (λi. pair-pmf (A i) (B i)) >>=
  (λf. return-pmf (fst ∘ h f, snd ∘ h f))
proof (rule pmf-eqI)
fix f
define g where g = h (λi. (fst f i, snd f i))
hence g-rev: f = (λf. (fst ∘ h f, snd ∘ h f)) g
by (simp add:comp-def h-h-id)
show pmf (pair-pmf (prod-pmf I A) (prod-pmf I B)) f =
  pmf (prod-pmf I (λi. pair-pmf (A i) (B i)) >>= (λf. return-pmf (fst ∘ h f,
snd ∘ h f))) f
apply (subst map-pmf-def[symmetric], simp add: g-rev, subst pmf-map-inj',
metis a)
apply (simp add:pmf-pair' pmf-prod-pmf[OF assms(1)] b prod.distrib)
using c by blast
qed
also have ... = ?rhs
apply (rule bind-pmf-cong ,simp)
apply (simp add: h-def comp-def set-prod-pmf[OF assms(1)] PiE-iff exten-
sional-def restrict-def)
apply (rule conjI)
by(rule ext, simp)+
finally show ?thesis
by blast
qed
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 = (
  (∀ x ∈ U. ∀ h ∈ H. f h x ∈ V) ∧ finite V ∧ V ≠ {} ∧
  (∀ x ∈ U. ∀ v ∈ V. P(h in pmf-of-set H. f h x = v) = 1 / real (card V)) ∧
  (∀ x ⊆ U. card x ≤ k ∧ finite x ⟶ 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, \dots, x_k the functions $h(x_1), \dots, h(x_m)$ 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** $\text{hash } F \ x \ \omega = \text{ring.eval } F \ \omega \ x$

lemma *hash-range*:

assumes *ring* F
assumes $\omega \in \text{bounded-degree-polynomials } F \ n$
assumes $x \in \text{carrier } F$
shows $\text{hash } F \ x \ \omega \in \text{carrier } F$
using *assms*
apply (*simp add:hash-def bounded-degree-polynomials-def*)
by (*metis ring.eval-in-carrier ring.polynomial-incl univ-poly-carrier*)

lemma *hash-range-2*:

assumes *ring* F
assumes $\omega \in \text{bounded-degree-polynomials } F \ n$
shows $(\lambda x. \text{hash } F \ x \ \omega) \text{ ' carrier } F \subseteq \text{carrier } F$
apply (*rule image-subsetI*)
by (*metis hash-range assms*)

lemma *poly-cards*:

assumes *field* $F \wedge \text{finite } (\text{carrier } F)$
assumes $K \subseteq \text{carrier } F$
assumes $\text{card } K \leq n$
assumes $y \text{ ' } K \subseteq (\text{carrier } F)$
shows $\text{card } \{\omega \in \text{bounded-degree-polynomials } F \ n. (\forall k \in K. \text{ring.eval } F \ \omega \ k = y \ k)\} =$
 $\text{card } (\text{carrier } F) \wedge (n - \text{card } K)$
using *interpolating-polynomials-count* [**where** $n = n - \text{card } K$ **and** $f = y$ **and** $F = F$
and $K = K$] *assms*
by *fastforce*

lemma *poly-cards-single*:

assumes *field* $F \wedge \text{finite } (\text{carrier } F)$
assumes $k \in \text{carrier } F$
assumes $1 \leq n$
assumes $y \in \text{carrier } F$
shows $\text{card } \{\omega \in \text{bounded-degree-polynomials } F \ n. \text{ring.eval } F \ \omega \ k = y\} =$
 $\text{card } (\text{carrier } F) \wedge (n - 1)$
using *poly-cards* [*OF assms(1)*, **where** $K = \{k\}$ **and** $y = \lambda -. y$, *simplified*] *assms(3)*
assms(4) [*simplified*]
by (*simp add:assms*)

lemma *expand-subset-filter*: $\{x \in A. P \ x\} = A \cap \{x. P \ x\}$

by *force*

```

lemma hash-prob:
  assumes field  $F \wedge \text{finite } (\text{carrier } F)$ 
  assumes  $K \subseteq \text{carrier } F$ 
  assumes  $\text{card } K \leq n$ 
  assumes  $y \in K \subseteq \text{carrier } F$ 
  shows  $\mathcal{P}(\omega \text{ in pmf-of-set } (\text{bounded-degree-polynomials } F \ n). (\forall x \in K. \text{hash } F \ x \ \omega = y \ x)) = 1 / (\text{real } (\text{card } (\text{carrier } F)))^{\text{card } K}$ 
proof –
  have  $0_F \in \text{carrier } F$ 
  using assms(1) field.is-ring ring.ring-simprules(2) by blast

  hence  $a:\text{card } (\text{carrier } F) > 0$ 
  apply (subst card-gt-0-iff)
  using assms(1) by blast

show ?thesis
  apply (subst measure-pmf-of-set)
  apply (metis non-empty-bounded-degree-polynomials field.is-ring assms(1))
  apply (metis fin-degree-bounded field.is-ring assms(1))
  apply (simp add:hash-def expand-subset-filter[symmetric])
  apply (subst poly-cards[OF assms(1) assms(2) assms(3) assms(4)])
  apply (subst bounded-degree-polynomials-count, metis field.is-ring assms(1),
metis assms(1))
  apply (subst frac-eq-eq)
  apply (simp add:a, simp add:a, simp)
  by (metis assms(3) le-add-diff-inverse2 power-add)
qed

lemma hash-prob-single:
  assumes field  $F \wedge \text{finite } (\text{carrier } F)$ 
  assumes  $x \in \text{carrier } F$ 
  assumes  $1 \leq n$ 
  assumes  $y \in \text{carrier } F$ 
  shows  $\mathcal{P}(\omega \text{ in pmf-of-set } (\text{bounded-degree-polynomials } F \ n). \text{hash } F \ x \ \omega = y) = 1 / (\text{real } (\text{card } (\text{carrier } F)))$ 
  using hash-prob[OF assms(1), where  $K=\{x\}$  and  $y=\lambda-. y$ , simplified] assms
  by (metis (no-types, lifting) Collect-cong One-nat-def UNIV-I space-measure-pmf)

lemma hash-indep-pmf:
  assumes field  $F \wedge \text{finite } (\text{carrier } F)$ 
  assumes  $J \subseteq \text{carrier } F$ 
  assumes finite  $J$ 
  assumes  $\text{card } J \leq n$ 
  assumes  $1 \leq n$ 
  shows prob-space.indep-vars (pmf-of-set (bounded-degree-polynomials  $F \ n$ ))
    ( $\lambda-. \text{pmf-of-set } (\text{carrier } F)$ ) (hash  $F$ )  $J$ 
proof –
  have  $0_{\text{poly-ring } F} \in \text{bounded-degree-polynomials } F \ n$ 
  apply (simp add:bounded-degree-polynomials-def)

```

```

apply (rule conjI)
apply (simp add: univ-poly-zero univ-poly-zero-closed)
using univ-poly-zero by blast
hence b: bounded-degree-polynomials  $F\ n \neq \{\}$ 
by blast
have c: finite (bounded-degree-polynomials  $F\ n$ )
by (metis finite-poly-count assms(1))
have d:  $\bigwedge A\ P.\ A \cap \{\omega.\ P\ \omega\} = \{\omega \in A.\ P\ \omega\}$ 
by blast

have fin-carr: finite (carrier  $F$ ) using assms(1) by blast
have e:ring  $F$  using assms(1) field.is-ring by blast
have f:  $0 < \text{card}\ (\text{carrier}\ F)$ 
by (metis assms(1) card-0-eq e empty-iff gr0I ring.ring-simprules(2))

define  $\Omega$  where  $\Omega = (\text{pmf-of-set}\ (\text{bounded-degree-polynomials}\ F\ n))$ 
have a: $\bigwedge a\ J'.$ 
   $J' \subseteq J \implies$ 
  finite  $J' \implies$ 
   $\text{measure}\ \Omega\ \{\omega.\ \forall x \in J'.\ \text{hash}\ F\ x\ \omega = a\ x\} =$ 
   $(\prod_{x \in J'.\ \text{measure}\ \Omega\ \{\omega.\ \text{hash}\ F\ x\ \omega = a\ x\}})$ 
proof -
  fix a
  fix  $J'$ 
  assume a-1:  $J' \subseteq J$ 
  assume a-11: finite  $J'$ 
  have a-2:  $\text{card}\ J' \leq n$  by (metis card-mono order-trans a-1 assms(3) assms(4))
  have a-3:  $J' \subseteq \text{carrier}\ F$  by (metis order-trans a-1 assms(2))
  have a-4:  $1 \leq n$  using assms by blast

  show  $\text{measure-pmf.prob}\ \Omega\ \{\omega.\ \forall x \in J'.\ \text{hash}\ F\ x\ \omega = a\ x\} =$ 
   $(\prod_{x \in J'.\ \text{measure-pmf.prob}\ \Omega\ \{\omega.\ \text{hash}\ F\ x\ \omega = a\ x\}})$ 
  proof (cases a '  $J' \subseteq \text{carrier}\ F$ )
    case True
    have a-5:  $\bigwedge x.\ x \in J' \implies x \in \text{carrier}\ F$  using a-1 assms(2) order-trans by
    force
    have a-6:  $\bigwedge x.\ x \in J' \implies a\ x \in \text{carrier}\ F$  using True by force
    show ?thesis
    apply (simp add:  $\Omega$ -def measure-pmf-of-set[OF b c] d hash-def)
    apply (subst poly-cards[OF assms(1) a-3 a-2], metis True)
    apply (simp add: bounded-degree-polynomials-count[OF e fin-carr] poly-cards-single[OF
    assms(1) a-5 a-4 a-6] power-divide)
    apply (subst frac-eq-eq, simp add:f, simp add:f)
    apply (simp add: power-add[symmetric] power-mult[symmetric])
    apply (rule arg-cong2[where f= $\lambda x\ y.\ x \wedge y$ ], simp)
    using a-2 a-4 mult-eq-if by force
  next
  case False
  then obtain j where a-8:  $j \in J'$  and a-9:  $a\ j \notin \text{carrier}\ F$  by blast

```

```

    have a-7:  $\bigwedge x \omega. \omega \in \text{bounded-degree-polynomials } F \ n \implies x \in \text{carrier } F \implies$ 
    hash  $F \ x \ \omega \in \text{carrier } F$ 
    apply (simp add: bounded-degree-polynomials-def hash-def)
    by (metis e ring.eval-in-carrier ring.polynomial-incl univ-poly-carrier)
    have a-10:  $\{\omega \in \text{bounded-degree-polynomials } F \ n. \forall x \in J'. \text{hash } F \ x \ \omega = a \ x\}$ 
    = {}
    apply (rule order-antisym)
    apply (rule subsetI, simp, metis a-7 a-8 a-9 a-3 in-mono)
    by (rule subsetI, simp)
    have a-12:  $\{\omega \in \text{bounded-degree-polynomials } F \ n. \text{hash } F \ j \ \omega = a \ j\} = \{\}$ 
    apply (rule order-antisym)
    apply (rule subsetI, simp, metis a-7 a-8 a-9 a-3 in-mono)
    by (rule subsetI, simp)
    then show ?thesis
    apply (simp add:  $\Omega$ -def measure-pmf-of-set[OF b c] d a-10)
    apply (rule prod-zero, metis a-11)
    apply (rule bexI[where x=j])
    by (simp add: a-12 a-8)+
  qed
qed
show ?thesis
  apply (rule indep-vars-pmf)
  using a by (simp add:  $\Omega$ -def)
qed

```

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 $k \ M' \ X' \ I = (\forall J \subseteq I. \text{card } J \leq k \implies \text{finite } J \implies \text{indep-vars } M' \ X' \ J)$

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
  ( $\lambda$ -. pmf-of-set (carrier  $F$ )) (hash  $F$ ) (carrier  $F$ )
  apply (simp add: measure-pmf.k-wise-indep-vars-def)
  using hash-indep-pmf[OF assms(1) - - assms(2)] by blast

```

lemma *hash-inj-if-degree-1*:

```

  assumes field  $F \wedge$  finite (carrier  $F$ )
  assumes  $\omega \in \text{bounded-degree-polynomials } F \ n$ 
  assumes degree  $\omega = 1$ 
  shows inj-on ( $\lambda x. \text{hash } F \ x \ \omega$ ) (carrier  $F$ )
proof (rule inj-onI)
  fix x y
  assume a1:  $x \in \text{carrier } F$ 
  assume a2:  $y \in \text{carrier } F$ 

```

```

assume  $a3$ :  $\text{hash } F \ x \ \omega = \text{hash } F \ y \ \omega$ 

interpret  $\text{field } F$ 
  by ( $\text{metis } \text{assms}(1)$ )

obtain  $u \ v$  where  $\omega\text{-def}$ :  $\omega = [u, v]$  using  $\text{assms}(3)$ 
  apply ( $\text{cases } \omega, \text{simp}$ )
  by ( $\text{cases } (\text{tl } \omega), \text{simp}, \text{simp}$ )

have  $u\text{-carr}$ :  $u \in \text{carrier } F - \{0_F\}$ 
  using  $\omega\text{-def } \text{assms}$  apply ( $\text{simp } \text{add}:\text{bounded-degree-polynomials-def}$ )
  by ( $\text{metis } \text{field.is-ring } \text{list.sel}(1) \text{ ring.degree-oneE } \text{assms}(1) \text{ assms}(3)$ )

have  $v\text{-carr}$ :  $v \in \text{carrier } F$ 
  using  $\omega\text{-def } \text{assms}(2)$  apply ( $\text{simp } \text{add}:\text{bounded-degree-polynomials-def}$ )
  by ( $\text{metis } \text{assms}(1) \text{ assms}(3) \text{ field.is-ring } \text{list.inject } \text{ring.degree-oneE}$ )

have  $u \otimes_F x \oplus_F v = u \otimes_F y \oplus_F v$ 
  using  $a1 \ a2 \ a3 \ u\text{-carr } v\text{-carr}$  by ( $\text{simp } \text{add}:\text{hash-def } \omega\text{-def}$ )

thus  $x = y$ 
  using  $u\text{-carr } a1 \ a2 \ v\text{-carr}$ 
  by ( $\text{simp } \text{add}:\text{local.field-Units}$ )
qed

lemma (in  $\text{prob-space}$ )  $k\text{-wise-subset}$ :
  assumes  $k\text{-wise-indep-vars } k \ M' \ X' \ I$ 
  assumes  $J \subseteq I$ 
  shows  $k\text{-wise-indep-vars } k \ M' \ X' \ J$ 
  using  $\text{assms}$  by ( $\text{simp } \text{add}:\text{k-wise-indep-vars-def}$ )

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  $\text{fin-bounded-degree-polynomials}$ :
  assumes  $p > 0$ 
  shows  $\text{finite } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ n)$ 
  apply ( $\text{rule } \text{fin-degree-bounded}$ )
  apply ( $\text{metis } \text{ZFact-is-crng } \text{crng-def}$ )
  by ( $\text{rule } \text{zfact-finite}[OF \ \text{assms}]$ )

```

```

lemma ne-bounded-degree-polynomials:
  shows bounded-degree-polynomials (ZFact (int p)) n  $\neq \{\}$ 
  apply (rule non-empty-bounded-degree-polynomials)
  by (metis ZFact-is-cring cring-def)

lemma card-bounded-degree-polynomials:
  assumes  $p > 0$ 
  shows card (bounded-degree-polynomials (ZFact (int p)) n) =  $p^n$ 
  apply (subst bounded-degree-polynomials-count)
  apply (metis ZFact-is-cring cring-def)
  apply (rule zfact-finite[OF assms])
  by (subst zfact-card, metis assms, simp)

fun hash :: nat  $\Rightarrow$  nat  $\Rightarrow$  int set list  $\Rightarrow$  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 –
  have UniversalHashFamily.hash (ZFact (int p)) (zfact-embed p x)  $\omega \in$  carrier
    (ZFact (int p))
  apply (rule UniversalHashFamily.hash-range[OF - assms(2)])
  apply (metis ZFact-is-cring cring-def)
  by (metis zfact-embed-ran[OF assms(1)] assms(3) atLeast0LessThan image-eqI
    lessThan-iff)
  thus ?thesis
  using the-inv-into-into[OF zfact-embed-inj[OF assms(1)], where B={0.. $p$ }]
    zfact-embed-ran[OF assms(1)]
  by (simp add:hash.simps)
qed

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 ( $\lambda x.$  hash p x  $\omega$ ) {0.. $p$ }
proof –
  have p-ge-0:  $p > 0$  using assms(1)
  by (simp add: prime-gt-0-nat)

  have ring-p: ring (ZFact (int p))
  by (metis ZFact-is-cring cring-def)

```

```

have inj-on (the-inv-into  $\{0..<p\}$  (zfact-embed p))  $\circ$  ( $\lambda x.$  (UniversalHashFamily.hash
(ZFact (int p)) x  $\omega$ ))  $\circ$  (zfact-embed p))  $\{0..<p\}$ 
  apply (rule comp-inj-on[OF zfact-embed-inj[OF p-ge-0]])
  apply (subst zfact-embed-ran[OF p-ge-0])
  apply (rule comp-inj-on)
  apply (rule UniversalHashFamily.hash-inj-if-degree-1[OF - assms(2) assms(3)])
  apply (metis zfact-prime-is-field[OF assms(1)] zfact-finite[OF p-ge-0])
  apply (rule inj-on-subset[OF - UniversalHashFamily.hash-range-2[OF ring-p
assms(2)])])
  apply (subst zfact-embed-ran[OF p-ge-0, symmetric])
  by (rule inj-on-the-inv-into[OF zfact-embed-inj[OF p-ge-0]])

thus ?thesis
  by (simp add:hash.simps comp-def)
qed

```

```

lemma hash-prob:
  assumes prime p
  assumes  $K \subseteq \{0..<p\}$ 
  assumes  $y \text{ ' } K \subseteq \{0..<p\}$ 
  assumes  $\text{card } K \leq n$ 
  shows  $\mathcal{P}(\omega \text{ in measure-pmf (pmf-of-set (bounded-degree-polynomials (ZFact (int p)) n)).$ 
 $(\forall x \in K. \text{hash } p \ x \ \omega = (y \ x))) = 1 \ / \ \text{real } p^{\wedge \text{card } K}$ 
proof -
  define  $y'$  where  $y' = \text{zfact-embed } p \circ y \circ (\text{the-inv-into } K \ (\text{zfact-embed } p))$ 
  define  $\Omega$  where  $\Omega = \text{pmf-of-set (bounded-degree-polynomials (ZFact (int p)) n)$ 

  have p-ge-0:  $p > 0$  using prime-gt-0-nat[OF assms(1)] by simp

  have  $\bigwedge x. x \in \text{zfact-embed } p \text{ ' } K \implies \text{the-inv-into } K \ (\text{zfact-embed } p) \ x \in K$ 
    apply (rule the-inv-into-into)
    apply (metis zfact-embed-inj[OF p-ge-0] assms(2) inj-on-subset)
    by auto

  hence ran-y:  $\bigwedge x. x \in \text{zfact-embed } p \text{ ' } K \implies y \ (\text{the-inv-into } K \ (\text{zfact-embed } p) \ x) \in \{0..<p\}$ 
    using assms(3) by blast

  have ran-y':  $y' \text{ ' } (\text{zfact-embed } p \text{ ' } K) \subseteq \text{carrier (ZFact (int p))}$ 
    apply (rule image-subsetI)
    apply (simp add:y'-def)
    by (metis zfact-embed-ran[OF p-ge-0] imageI ran-y)

  have K-embed:  $\text{zfact-embed } p \text{ ' } K \subseteq \text{carrier (ZFact (int p))}$ 
    using zfact-embed-ran[OF p-ge-0] assms(2) by auto

  have ring-zfact: ring (ZFact (int p))
    using ZFact-is-crng crng-def by blast

```

```

have  $\mathcal{P}(\omega \text{ in measure-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \text{ } n))).$ 
   $(\forall x \in K. \text{ hash } p \ x \ \omega = (y \ x))) = \mathcal{P}(\omega \text{ in measure-pmf } \Omega. (\forall x \in K. \text{ hash } p \ x \ \omega = (y \ x)))$ 
  by (simp add:  $\Omega$ -def)
also have ... =
   $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. (\forall x \in \text{zfact-embed } p \text{ } K. \text{ UniversalHashFamily.hash } (\text{ZFact } (\text{int } p)) \ x \ \omega = y' \ x))$ 
  apply (rule pmf-eq)
  apply (simp add:y'-def hash.simps  $\Omega$ -def)
  apply (subst (asm) set-pmf-of-set, metis ne-bounded-degree-polynomials,
    metis fin-bounded-degree-polynomials[OF p-ge-0]))
  apply (rule ball-cong, simp)
  apply (subst the-inv-into-f-f)
  apply (metis zfact-embed-inj[OF p-ge-0] assms(2) inj-on-subset)
  apply (simp)
  apply (subst eq-commute)
  apply (rule order-antisym)
  apply (simp, rule impI)
  apply (subst f-the-inv-into-f[OF zfact-embed-inj[OF p-ge-0]])
  apply (subst zfact-embed-ran[OF p-ge-0])
  apply (rule UniversalHashFamily.hash-range[OF ring-zfact, where n=n],
simp)
  apply (meson K-embed image-subset-iff)
  apply (simp)
  apply (simp, rule impI)
  apply (subst the-inv-into-f-f[OF zfact-embed-inj[OF p-ge-0]])
  apply (metis assms(3) image-subset-iff)
  by simp
also have ... =
   $1 / \text{real } (\text{card } (\text{carrier } (\text{ZFact } (\text{int } p)))) \sim (\text{card } (\text{zfact-embed } p \text{ } K))$ 
  apply (simp only:  $\Omega$ -def)
  apply (rule UniversalHashFamily.hash-prob[where K=zfact-embed p ' K and
F=ZFact (int p) and n=n and y=y'])
  apply (metis zfact-prime-is-field[OF assms(1)] zfact-finite[OF p-ge-0])
  apply (metis zfact-embed-ran[OF p-ge-0] assms(2) image-mono)
  apply (rule order-trans[OF card-image-le], rule finite-subset[OF assms(2)],
simp, metis assms(4))
  using K-embed ran-y' by blast
  also have ... =  $1 / \text{real } p \sim (\text{card } K)$ 
  apply (subst card-image, meson inj-on-subset zfact-embed-inj[OF p-ge-0] assms(2))
  apply (subst zfact-card[OF p-ge-0])
  by simp
finally show ?thesis by simp
qed

lemma hash-prob-2:
  assumes prime p

```



```

assumes inj-on  $x \ K$ 
assumes  $x \in K \subseteq \{0..<p\}$ 
assumes  $y \in K \subseteq \{0..<p\}$ 
assumes  $\text{card } K \leq n$ 
shows  $\mathcal{P}(\omega \text{ in } \text{measure-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ n)))$ .
   $(\forall k \in K. \text{hash } p \ (x \ k) \ \omega = (y \ k))) = 1 / \text{real } p^{\text{card } K}$  (is  $?lhs = ?rhs$ )
proof –
  define  $y'$  where  $y' = y \circ (\text{the-inv-into } K \ x)$ 
  have  $?lhs = \mathcal{P}(\omega \text{ in } \text{measure-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ n)))$ .
     $(\forall k \in x \in K. \text{hash } p \ k \ \omega = y' \ k))$ 
    apply (rule pmf-eq)
    apply (simp add:y'-def)
    apply (rule ball-cong, simp)
    by (subst the-inv-into-f-f[OF assms(2)], simp, simp)
  also have  $\dots = 1 / \text{real } p^{\text{card } (x \in K)}$ 
    apply (rule hash-prob[OF assms(1) assms(3)])
    using assms apply (simp add: y'-def subset-eq the-inv-into-f-f)
    by (metis card-image assms(2) assms(5))
  also have  $\dots = ?rhs$ 
    by (subst card-image[OF assms(2)], simp)
  finally show  $?thesis$  by simp
qed

```

lemma *hash-prob-range*:

```

assumes prime  $p$ 
assumes  $x < p$ 
assumes  $n > 0$ 
shows  $\mathcal{P}(\omega \text{ in } \text{measure-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ n)))$ .
   $\text{hash } p \ x \ \omega \in A = \text{card } (A \cap \{0..<p\}) / p$ 
proof –
  define  $\Omega$  where  $\Omega = \text{measure-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ n))$ 
  have  $p\text{-ge-0}: p > 0$  using assms(1) by (simp add: prime-gt-0-nat)

  have  $\mathcal{P}(\omega \text{ in } \Omega. \text{hash } p \ x \ \omega \in A) = \text{measure } \Omega \ (\bigcup k \in A \cap \{0..<p\}. \{\omega. \text{hash } p \ x \ \omega = k\})$ 
    apply (simp only:Ω-def)
    apply (rule pmf-eq, simp)
    apply (subst (asm) set-pmf-of-set[OF ne-bounded-degree-polynomials fin-bounded-degree-polynomials[OF p-ge-0]])
    using hash-range[OF p-ge-0 - assms(2)] by simp
  also have  $\dots = (\sum k \in (A \cap \{0..<p\}). \text{measure } \Omega \ \{\omega. \text{hash } p \ x \ \omega = k\})$ 
    apply (rule measure-finite-Union, simp, simp add:Ω-def)
    apply (simp add:disjoint-family-on-def, fastforce)
    by (simp add:Ω-def)
  also have  $\dots = (\sum k \in (A \cap \{0..<p\}). \mathcal{P}(\omega \text{ in } \Omega. \forall x' \in \{x\}. \text{hash } p \ x' \ \omega = k$ 

```

```

))
  by (simp add:Ω-def)
also have ... = (∑ k ∈ (A ∩ {0..

}). 1 / real p ^ card {x})
  apply (rule sum.cong, simp)
  apply (simp only:Ω-def)
  apply (rule hash-prob[OF assms(1)], simp add:assms, simp)
  using assms(3) by simp
also have ... = card (A ∩ {0..

) / real p
  by simp
finally show ?thesis
  by (simp only:Ω-def)
qed

lemma hash-k-wise-indep:
  assumes prime p
  assumes 1 ≤ 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..

) (hash p) {0..


```

14.1 Encoding

```

fun zfactS where zfactS p x = (

```

```

    if  $x \in \text{zfact-embed } p \text{ ' } \{0..<p\}$  then
       $N_S \text{ (the-inv-into } \{0..<p\} \text{ (zfact-embed } p) \text{ } x)$ 
    else
      None
  )

lemma zfact-encoding :
  is-encoding (zfactS p)
proof –
  have  $p > 0 \implies \text{is-encoding } (\lambda x. \text{zfact}_S p \text{ } x)$ 
  apply simp
  apply (rule encoding-compose[where  $f=N_S$ ])
  apply (metis nat-encoding, simp)
  by (metis inj-on-the-inv-into zfact-embed-inj)
  moreover have is-encoding (zfactS 0)
  by (simp add:is-encoding-def)
  ultimately show ?thesis by blast
qed

lemma bounded-degree-polynomial-bit-count:
  assumes  $p > 0$ 
  assumes  $x \in \text{bounded-degree-polynomials } (Z\text{Fact } p) \text{ } n$ 
  shows bit-count (listS (zfactS p) x)  $\leq \text{ereal } (\text{real } n * (2 * \log 2 \text{ } p + 2) + 1)$ 
proof –
  have  $b:\text{real } (\text{length } x) \leq \text{real } n$ 
  using assms(2)
  apply (simp add:bounded-degree-polynomials-def)
  apply (cases  $x=[]$ , simp, simp)
  by linarith

  have  $a:\bigwedge y. y \in \text{set } x \implies y \in \text{zfact-embed } p \text{ ' } \{0..<p\}$ 
  using assms(2)
  apply (simp add:bounded-degree-polynomials-def)
  by (metis length-greater-0-conv length-pos-if-in-set polynomial-def subsetD univ-poly-carrier
  zfact-embed-ran[OF assms(1)])

  have bit-count (listS (zfactS p) x)  $\leq \text{ereal } (\text{real } (\text{length } x)) * ( \text{ereal } (2 * \log 2$ 
   $(1 + \text{real } (p-1)) + 1) + 1) + 1$ 
  apply (rule list-bit-count-est)
  apply (simp add:a del:N_S.simps)
  apply (rule nat-bit-count-est)
  by (metis a the-inv-into-into[OF zfact-embed-inj[OF assms(1)], where  $B=\{0..<p\}$ ,
  simplified]
  Suc-pred assms(1) less-Suc-eq-le)
  also have  $\dots \leq \text{ereal } (\text{real } n) * (2 + \text{ereal } (2 * \log 2 \text{ } p) ) + 1$ 
  apply simp
  apply (rule mult-mono, metis b)
  apply (rule add-mono)
  using assms(1) by simp+

```

```

    also have ... = ereal (real n * (2 * log 2 p + 2) + 1)
    by simp
    finally show ?thesis by simp
qed

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 (λx. f1 x + f2 x) ∈ O[F'](g)
  by (rule sum-in-bigo[OF assms(2) assms(1)])

```

```

lemma landau-sum:
  assumes eventually (λx. g1 x ≥ (0::real)) F'
  assumes eventually (λx. g2 x ≥ 0) F'
  assumes f1 ∈ O[F'](g1)
  assumes f2 ∈ O[F'](g2)
  shows (λx. f1 x + f2 x) ∈ O[F'](λx. g1 x + g2 x)

```

proof –

```

  obtain c1 where a1: c1 > 0 and b1: eventually (λx. abs (f1 x) ≤ c1 * abs (g1
x)) F'

```

```

    using assms(3) by (simp add:bigo-def, blast)

```

```

  obtain c2 where a2: c2 > 0 and b2: eventually (λx. abs (f2 x) ≤ c2 * abs (g2
x)) F'

```

```

    using assms(4) by (simp add:bigo-def, blast)

```

```

  have eventually (λx. abs (f1 x + f2 x) ≤ (max c1 c2) * abs (g1 x + g2 x)) F'

```

```

  proof (rule eventually-mono[OF eventually-conj[OF b1 eventually-conj[OF b2
eventually-conj[OF assms(1) assms(2)]]]])

```

```

    fix x

```

```

    assume a: |f1 x| ≤ c1 * |g1 x| ∧ |f2 x| ≤ c2 * |g2 x| ∧ 0 ≤ g1 x ∧ 0 ≤ g2 x

```

```

    have |f1 x + f2 x| ≤ |f1 x| + |f2 x| using abs-triangle-ineq by blast

```

```

    also have ... ≤ c1 * |g1 x| + c2 * |g2 x| using a add-mono by blast

```

```

    also have ... ≤ max c1 c2 * |g1 x| + max c1 c2 * |g2 x|

```

```

    apply (rule add-mono)

```

```

    apply (rule mult-right-mono, simp)

```

```

    apply (metis a a1 abs-le-zero-iff abs-zero linorder-not-less order-trans semir-
ing-norm(63) zero-le-mult-iff)

```

```

    apply (rule mult-right-mono, simp)
    by (metis a a2 abs-le-zero-iff abs-zero linorder-not-less order-trans semir-
ing-norm(63) zero-le-mult-iff)
    also have ... ≤ max c1 c2 * (|g1 x + g2 x|)
    apply (subst distrib-left[symmetric])
    apply (rule mult-left-mono)
    using a a1 a2 by auto
    finally show |f1 x + f2 x| ≤ max c1 c2 * |g1 x + g2 x| by (simp add: algebra-simps)
qed
thus ?thesis
  apply (simp add: bigo-def)
  apply (rule exI[where x = max c1 c2])
  using a1 a2 by linarith
qed

```

```

lemma landau-sum-1:
  assumes eventually (λx. g1 x ≥ (0::real)) F'
  assumes eventually (λx. g2 x ≥ 0) F'
  assumes f ∈ O[F'](g1)
  shows f ∈ O[F'](λx. g1 x + g2 x)
proof -
  have f = (λx. f x + 0)
  by simp
  also have ... ∈ O[F'](λx. g1 x + g2 x)
  by (rule landau-sum[OF assms(1) assms(2) assms(3) zero-in-bigo])
  finally show ?thesis by simp
qed

```

```

lemma landau-sum-2:
  assumes eventually (λx. g1 x ≥ (0::real)) F'
  assumes eventually (λx. g2 x ≥ 0) F'
  assumes f ∈ O[F'](g2)
  shows f ∈ O[F'](λx. g1 x + g2 x)
proof -
  have f = (λx. 0 + f x)
  by simp
  also have ... ∈ O[F'](λx. g1 x + g2 x)
  by (rule landau-sum[OF assms(1) assms(2) zero-in-bigo assms(3)])
  finally show ?thesis by simp
qed

```

```

lemma landau-ln-3:
  assumes eventually (λx. (1::real) ≤ f x) F'
  assumes f ∈ O[F'](g)
  shows (λx. ln (f x)) ∈ O[F'](g)
proof -
  have a: (λx. ln (f x)) ∈ O[F'](f)
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF assms(1)])

```

```

    apply (subst abs-of-nonneg, subst ln-ge-zero-iff, simp, simp, simp)
    using ln-less-self
    by (meson ln-bound order.strict-trans2 zero-less-one)
  show ?thesis
    by (rule landau-o.big-trans[OF a assms(2)])
qed

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

```

```

    apply (subst abs-of-nonneg)
    apply (rule ln-ge-zero) using c assms(1) by simp+
    finally show abs (ln (f x)) ≤ d * abs (ln (g x)) by simp
qed
show ?thesis
  apply (simp add:bigo-def)
  apply (rule exI[where x=d])
  apply (rule conjI, simp add:d-def)
  apply (meson add-pos-nonneg assms(1) less-le-not-le less-numeral-extra(1)
ln-ge-zero max.cobounded1 zero-le-divide-iff)
  by (metis d)
qed

```

```

lemma landau-real-nat:
  fixes f :: 'a ⇒ int
  assumes (λx. of-int (f x)) ∈ O[F](g)
  shows (λx. real (nat (f x))) ∈ O[F](g)
proof -
  obtain c where a: c > 0 and b: eventually (λx. abs (of-int (f x)) ≤ c * abs (g
x)) F'
  using assms(1) by (simp add:bigo-def, blast)

  show ?thesis
    apply (simp add:bigo-def)
    apply (rule exI[where x=c])
    apply (rule conjI[OF a])
    apply (rule eventually-mono[OF b])
    by simp
qed

```

```

lemma landau-ceil:
  assumes (λ-. 1) ∈ O[F](g)
  assumes f ∈ O[F](g)
  shows (λx. real-of-int ⌈f x⌉) ∈ O[F](g)
  apply (rule landau-o.big-trans[where g=λx. 1 + abs (f x)])
  apply (rule landau-o.big-mono)
  apply (rule always-eventually, rule allI, simp, linarith)
  by (rule sum-in-bigo[OF assms(1)], simp add:assms)

```

```

lemma landau-nat-ceil:
  assumes (λ-. 1) ∈ O[F](g)
  assumes f ∈ O[F](g)
  shows (λx. real (nat ⌈f x⌉)) ∈ O[F](g)
  apply (rule landau-real-nat)
  by (rule landau-ceil[OF assms(1) assms(2)])

```

```

lemma landau-const-inv:
  assumes c > (0::real)
  assumes (λx. 1 / f x) ∈ O[F](g)

```

```

shows ( $\lambda x. c / f x \in O[F'](g)$ )
proof –
  obtain  $d$  where  $a: d > 0$  and  $b: \text{eventually } (\lambda x. \text{abs } (1 / f x) \leq d * \text{abs } (g x))$ 
   $F'$ 
    using  $\text{assms}(2)$  by ( $\text{simp add: bigo-def, blast}$ )
    have  $c: \text{eventually } (\lambda x. |c| / |f x| \leq (c) * d * \text{abs } (g x))$   $F'$ 
    apply ( $\text{rule eventually-mono}[OF b]$ )
    using  $\text{assms}(1)$ 
    apply  $\text{simp}$ 
    by ( $\text{metis Groups.mult-ac}(2) \text{ Groups.mult-ac}(3) \text{ divide-inverse inverse-eq-divide}$ 
 $\text{less-imp-le mult-le-cancel-left not-less}$ )
    show  $?thesis$ 
    apply ( $\text{simp add: bigo-def}$ )
    apply ( $\text{rule exI}[\text{where } x=c*d]$ )
    apply ( $\text{rule conjI, rule mult-pos-pos}[OF \text{assms}(1) a]$ )
    by ( $\text{rule c}$ )
qed

```

```

lemma eventually-nonneg-div:
  assumes  $\text{eventually } (\lambda x. (0::\text{real}) \leq f x)$   $F'$ 
  assumes  $\text{eventually } (\lambda x. 0 < g x)$   $F'$ 
  shows  $\text{eventually } (\lambda x. 0 \leq f x / g x)$   $F'$ 
  apply ( $\text{rule eventually-mono}[OF \text{eventually-conj}[OF \text{assms}(1) \text{assms}(2)]]$ )
  by  $\text{simp}$ 

```

```

lemma eventually-nonneg-add:
  assumes  $\text{eventually } (\lambda x. (0::\text{real}) \leq f x)$   $F'$ 
  assumes  $\text{eventually } (\lambda x. 0 \leq g x)$   $F'$ 
  shows  $\text{eventually } (\lambda x. 0 \leq f x + g x)$   $F'$ 
  apply ( $\text{rule eventually-mono}[OF \text{eventually-conj}[OF \text{assms}(1) \text{assms}(2)]]$ )
  by  $\text{simp}$ 

```

```

lemma eventually-ln-ge-iff:
  assumes  $\text{eventually } (\lambda x. (\exp (c::\text{real})) \leq f x)$   $F'$ 
  shows  $\text{eventually } (\lambda x. c \leq \ln (f x))$   $F'$ 
  apply ( $\text{rule eventually-mono}[OF \text{assms}(1)]$ )
  by ( $\text{meson ln-ge-iff exp-gt-zero order-less-le-trans}$ )

```

```

lemma div-commute:  $(a::\text{real}) / b = (1/b) * a$  by  $\text{simp}$ 

```

```

lemma eventually-prod1':
  assumes  $B \neq \text{bot}$ 
  shows  $(\forall_F x \text{ in } A \times_F B. P (\text{fst } x)) \longleftrightarrow (\forall_F x \text{ in } A. P x)$ 
  apply ( $\text{subst } (2) \text{ eventually-prod1}[OF \text{assms}(1), \text{symmetric}]$ )
  apply ( $\text{rule arg-cong2}[\text{where } f=\text{eventually}]$ )
  by ( $\text{rule ext, simp add: case-prod-beta, simp}$ )

```

```

lemma eventually-prod2':
  assumes  $A \neq \text{bot}$ 

```



```

shows ( $\forall_F x \text{ in } A \times_F B. P \text{ (snd } x)$ )  $\longleftrightarrow$  ( $\forall_F x \text{ in } B. P \text{ } x$ )
apply (subst (2) eventually-prod2[OF assms(1), symmetric])
apply (rule arg-cong2[where f=eventually])
by (rule ext, simp add:case-prod-beta, simp)

instantiation rat :: linorder-topology
begin

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

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

lemma inv-at-right-0-inf:
   $\forall_F x \text{ in at-right } 0. c \leq 1 \text{ / real-of-rat } x$ 
apply (rule eventually-at-rightI[where b=1/rat-of-int (max  $\lceil c \rceil$  1)])
apply (rule order-trans[where y=real-of-int (max  $\lceil c \rceil$  1)], linarith)
apply (subst pos-le-divide-eq, simp)
apply simp
apply (subst (asm) pos-less-divide-eq, simp)
apply (metis less-eq-real-def mult.commute of-rat-less-1-iff of-rat-mult of-rat-of-int-eq)
by simp

end

```

16 Frequency Moment 0

```

theory Frequency-Moment-0
  imports Main Primes-Ext Float-Ext Median OrderStatistics UniversalHashFamilyOfPrime 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 to the Isabelle proof here, there is also an informal handwritten proof in Appendix A.

```

type-synonym f0-state = nat  $\times$  nat  $\times$  nat  $\times$  nat  $\times$  (nat  $\Rightarrow$  (int set list))  $\times$  (nat  $\Rightarrow$  float set)

```

```

fun f0-init :: rat  $\Rightarrow$  rat  $\Rightarrow$  nat  $\Rightarrow$  f0-state pmf where
  f0-init  $\delta \in n =$ 
    do {
      let s = nat  $\lceil -18 * \ln (\text{real-of-rat } \delta) \rceil$ ;

```

```

    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 (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)))
    ) s)

```

```

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:
  assumes ε ∈ {0<.. $1$ }
  assumes δ ∈ {0<.. $1$ }
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  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 (subst sketch-def, induction as rule:rev-induct)
  case Nil
  then show ?case
    apply (simp add:s-def[symmetric] p-def[symmetric] map-pmf-def[symmetric]
t-def[symmetric] r-def[symmetric])
    apply (rule arg-cong2[where f=map-pmf])
    apply (rule ext)
    apply simp
    by (rule ext, simp add:f0-sketch-def least-def, simp)
next
  case (snoc x xs)
  then show ?case
    apply (simp add:map-pmf-def)
    apply (subst bind-assoc-pmf)

```

```

    apply (subst bind-return-pmf)
    apply (rule arg-cong2[where f=bind-pmf], simp)
    apply (simp)
    apply (rule ext, rule arg-cong[where f=return-pmf], simp)
    apply (rule ext)
    apply (simp add:f0-sketch-def)
    by (subst least-insert, simp, simp)

```

qed

lemma (in prob-space) prob-sub-additive:

```

  assumes Collect P ∈ sets M
  assumes Collect Q ∈ sets M
  shows  $\mathcal{P}(\omega \text{ in } M. P \ \omega \vee Q \ \omega) \leq \mathcal{P}(\omega \text{ in } M. P \ \omega) + \mathcal{P}(\omega \text{ in } M. Q \ \omega)$ 
proof -
  have  $\mathcal{P}(\omega \text{ in } M. P \ \omega \vee Q \ \omega) = \text{measure } M \ (\{\omega \in \text{space } M. P \ \omega\} \cup \{\omega \in \text{space } M. Q \ \omega\})$ 
  apply (rule arg-cong2[where f=measure], simp)
  by (subst set-eq-iff, rule allI, blast)
  also have ... ≤ measure M  $\{\omega \in \text{space } M. P \ \omega\} + \text{measure } M \ \{\omega \in \text{space } M. Q \ \omega\}$ 
  apply (rule measure-subadditive)
  apply (metis (no-types, lifting) Collect-cong mem-Collect-eq sets.sets-into-space subsetD assms(1))
  apply (metis (no-types, lifting) Collect-cong mem-Collect-eq sets.sets-into-space subsetD assms(2))
  by simp+
  finally show ?thesis by simp
qed

```

lemma (in prob-space) prob-sub-additiveI:

```

  assumes Collect P ∈ sets M
  assumes Collect Q ∈ sets M
  assumes  $\mathcal{P}(\omega \text{ in } M. P \ \omega) \leq r1$ 
  assumes  $\mathcal{P}(\omega \text{ in } M. Q \ \omega) \leq r2$ 
  shows  $\mathcal{P}(\omega \text{ in } M. P \ \omega \vee Q \ \omega) \leq r1 + r2$ 
proof -
  have  $\mathcal{P}(\omega \text{ in } M. P \ \omega \vee Q \ \omega) \leq \mathcal{P}(\omega \text{ in } M. P \ \omega) + \mathcal{P}(\omega \text{ in } M. Q \ \omega)$ 
  by (rule prob-sub-additive[OF assms(1) assms(2)])
  also have ... ≤ r1 + r2
  by (rule add-mono, metis assms(3), metis assms(4))
  finally show ?thesis by simp
qed

```

lemma (in prob-space) prob-mono:

```

  assumes Collect Q ∈ sets M
  assumes  $\bigwedge \omega. \omega \in \text{space } M \implies P \ \omega \implies Q \ \omega$ 
  shows  $\mathcal{P}(\omega \text{ in } M. P \ \omega) \leq \mathcal{P}(\omega \text{ in } M. Q \ \omega)$ 
  apply (rule finite-measure.finite-measure-mono)
  apply simp

```

apply (rule subsetI, simp add:assms(2))
by (metis (no-types, lifting) assms(1) Collect-cong mem-Collect-eq sets.sets-into-space subsetD)

lemma in-events-pmf: $A \in \text{measure-pmf.events } \Omega$
by simp

lemma pmf-add:

assumes $\bigwedge x. x \in P \implies x \in \text{set-pmf } \Omega \implies x \in Q \vee x \in R$
shows $\text{measure } (\text{measure-pmf } \Omega) P \leq \text{measure } (\text{measure-pmf } \Omega) Q + \text{measure } (\text{measure-pmf } \Omega) R$

proof –

have $\text{measure } (\text{measure-pmf } \Omega) P \leq \text{measure } (\text{measure-pmf } \Omega) (Q \cup R)$

apply (rule pmf-mono-1)

using assms **by** blast

also have $\dots \leq \text{measure } (\text{measure-pmf } \Omega) Q + \text{measure } (\text{measure-pmf } \Omega) R$

by (rule measure-subadditive, simp+)

finally show ?thesis **by** simp

qed

lemma pmf-mono:

assumes $\bigwedge x. x \in P \implies x \in Q$

shows $\text{measure } (\text{measure-pmf } \Omega) P \leq \text{measure } (\text{measure-pmf } \Omega) Q$

apply (rule pmf-mono-1) **using** assms **by** auto

lemma abs-ge-iff: $((x::\text{real}) \leq \text{abs } y) = (x \leq y \vee x \leq -y)$
by linarith

lemma two-powr-0: $2^{\text{powr } (0::\text{real})} = 1$
by simp

lemma count-nat-abs-diff-2:

fixes $x :: \text{nat}$

fixes $q :: \text{real}$

assumes $q \geq 0$

defines $A \equiv \{(k::\text{nat}). \text{abs } (\text{real } x - \text{real } k) \leq q \wedge k \neq x\}$

shows $\text{real } (\text{card } A) \leq 2 * q$ **and** finite A

proof –

have $a: \text{of-nat } x \in \{\lceil \text{real } x - q \rceil .. \lfloor \text{real } x + q \rfloor\}$

using assms

by (simp add: ceiling-le-iff)

have $\text{card } A = \text{card } (\text{int } 'A)$

by (rule card-image[symmetric], simp)

also have $\dots \leq \text{card } (\{\lceil \text{real } x - q \rceil .. \lfloor \text{real } x + q \rfloor\} - \{\text{of-nat } x\})$

apply (rule card-mono, simp)

apply (rule image-subsetI)

apply (simp add:A-def abs-le-iff)

by linarith

```

also have ... = card { $\lceil \text{real } x - q \rceil .. \lfloor \text{real } x + q \rfloor$ } - 1
  by (rule card-Diff-singleton, rule a)
also have ... = int (card { $\lceil \text{real } x - q \rceil .. \lfloor \text{real } x + q \rfloor$ }) - int 1
  apply (rule of-nat-diff)
  by (metis a card-0-eq empty-iff finite-atLeastAtMost-int less-one linorder-not-le)
also have ...  $\leq \lfloor q + \text{real } x \rfloor + 1 - \lceil \text{real } x - q \rceil - 1$ 
  using assms
  apply simp
  by linarith
also have ...  $\leq 2 * q$ 
  by linarith
finally show card A  $\leq 2 * q$ 
  by simp
show finite A
  apply (simp add:A-def)
  apply (rule finite-subset[where  $B = \{0..x + \text{nat } \lceil q \rceil\}$ ])
  apply (rule subsetI, simp add:abs-le-iff)
  using assms apply linarith by simp
qed

```

lemma *f0-collision-prob*:

```

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

```

proof -

```

have  $p\text{-ge-0}$ :  $p > 0$ 
  using assms prime-gt-0-nat by blast

```

```

have  $c\text{-ge-0}$ :  $c \geq 0$  using assms by simp

```

```

have  $\text{two-pow-r-le-1}$ :  $2^{\text{powr } (-\text{real } r)} \leq 1$ 
  by (subst two-powr-0[symmetric], rule powr-mono, simp, simp)

```

```

have  $f\text{-M}$ : finite M
  by (rule finite-subset[where  $B = \{0..<p\}$ ], metis assms(3), simp)

```

```

have  $a2$ :  $\bigwedge x. x < p \implies \omega \in \text{bounded-degree-polynomials } (\text{ZFact } p) \ 2 \implies \text{hash}$ 
 $p \ x \ \omega < p$ 
  using hash-range[OF p-ge-0] by simp

```

```

have  $\bigwedge \omega. \text{degree } \omega \geq 1 \implies \omega \in \text{bounded-degree-polynomials } (\text{ZFact } p) \text{ } 2 \implies$ 
 $\text{degree } \omega = 1$ 
  apply (simp add: bounded-degree-polynomials-def)
  by (metis One-nat-def Suc-1 le-less-Suc-eq less-imp-diff-less list.size(3) pos2)
hence a3:  $\bigwedge \omega \ x \ y. x < p \implies y < p \implies x \neq y \implies \text{degree } \omega \geq 1 \implies$ 
 $\omega \in \text{bounded-degree-polynomials } (\text{ZFact } p) \text{ } 2 \implies$ 
 $\text{hash } p \ x \ \omega \neq \text{hash } p \ y \ \omega$ 
  using hash-inj-if-degree-1[OF assms(1)]
  by (meson atLeastLessThan-iff inj-on-def less-nat-zero-code linorder-not-less)

have a1:
   $\bigwedge x \ y. x < y \implies x \in M \implies y \in M \implies \text{measure } \Omega$ 
 $\{\omega. \text{degree } \omega \geq 1 \wedge \text{truncate-down } r (\text{hash } p \ x \ \omega) \leq c \wedge$ 
 $\text{truncate-down } r (\text{hash } p \ x \ \omega) = \text{truncate-down } r (\text{hash } p \ y \ \omega)\} \leq$ 
 $12 * c^2 * 2^{\text{powr } (-\text{real } r)} / (\text{real } p)^2$ 
proof -
  fix x y
  assume a1-1:  $x \in M$ 
  assume a1-2:  $y \in M$ 
  assume a1-3:  $x < y$ 

  have a1-4:  $\bigwedge u \ v. \text{truncate-down } r (\text{real } u) \leq c \implies$ 
 $\text{truncate-down } r (\text{real } u) = \text{truncate-down } r (\text{real } v) \implies$ 
 $\text{real } u \leq 2 * c \wedge |\text{real } u - \text{real } v| \leq 2 * c * 2^{\text{powr } (-\text{real } r)}$ 
proof -
  fix u v
  assume a-1:  $\text{truncate-down } r (\text{real } u) \leq c$ 
  assume a-2:  $\text{truncate-down } r (\text{real } u) = \text{truncate-down } r (\text{real } v)$ 
  have a-3:  $2 * 2^{\text{powr } (-\text{real } r)} = 2^{\text{powr } (1 - \text{real } r)}$ 
  by (simp add: divide-powr-uminus powr-diff)

  have a-4-1:  $1 \leq 2 * (1 - 2^{\text{powr } (-\text{real } r)})$ 
  apply (simp, subst a-3, subst (2) two-powr-0[symmetric])
  apply (rule powr-mono)
  using assms(5) by simp+

  have a-4:  $(c * 1) / (1 - 2^{\text{powr } (-\text{real } r)}) \leq c * 2$ 
  apply (subst pos-divide-le-eq, simp)
  apply (subst two-powr-0[symmetric])
  apply (rule powr-less-mono) using assms(5) apply simp
  apply simp
  using a-4-1
  by (metis (no-types, opaque-lifting) c-ge-0 mult.left-commute mult.right-neutral
  mult-left-mono)

  have a-5:  $\bigwedge x. \text{truncate-down } r (\text{real } x) \leq c \implies \text{real } x \leq c * 2$ 
  apply (rule order-trans[OF - a-4])
  apply (subst pos-le-divide-eq)
  apply (simp, subst two-powr-0[symmetric])

```

```

    apply (rule powr-less-mono) using assms(5) apply simp
  apply simp
    using truncate-down-pos[OF of-nat-0-le-iff] order-trans apply simp by
blast

  have a-6:  $\text{real } u \leq c * 2$ 
    using a-1 a-5 by simp
  have a-7:  $\text{real } v \leq c * 2$ 
    using a-1 a-2 a-5 by simp
  have  $|\text{real } u - \text{real } v| \leq (\max |\text{real } u| |\text{real } v|) * 2 \text{ powr } (-\text{real } r)$ 
    apply (rule truncate-down-eq) using a-2 by simp
  also have  $\dots \leq (c * 2) * 2 \text{ powr } (-\text{real } r)$ 
    apply (rule mult-right-mono) using a-6 a-7 by simp+
  finally have a-8:  $|\text{real } u - \text{real } v| \leq 2 * c * 2 \text{ powr } (-\text{real } r)$ 
    by simp

  show  $\text{real } u \leq 2 * c \wedge |\text{real } u - \text{real } v| \leq 2 * c * 2 \text{ powr } (-\text{real } r)$ 
    using a-6 a-8 by simp
qed

have measure  $\Omega \{ \omega. \text{degree } \omega \geq 1 \wedge \text{truncate-down } r (\text{hash } p \ x \ \omega) \leq c \wedge$ 
 $\text{truncate-down } r (\text{hash } p \ x \ \omega) = \text{truncate-down } r (\text{hash } p \ y \ \omega) \}$   $\leq$ 
 $\text{measure } \Omega (\bigcup i \in \{(u,v) \in \{0..<p\} \times \{0..<p\}. u \neq v \wedge$ 
 $\text{truncate-down } r \ u \leq c \wedge \text{truncate-down } r \ u = \text{truncate-down } r \ v\}.$ 
 $\{ \omega. \text{hash } p \ x \ \omega = \text{fst } i \wedge \text{hash } p \ y \ \omega = \text{snd } i \})$ 
  apply (rule pmf-mono-1)
  apply (simp add:  $\Omega$ -def)
  apply (subst (asm) set-pmf-of-set)
    apply (rule ne-bounded-degree-polynomials)
  apply (rule fin-bounded-degree-polynomials[OF p-ge-0])
  by (metis assms(3) a2 a3 a1-1 a1-2 a1-3 One-nat-def less-not-refl3 atLeast-
LessThan-iff subset-eq)
  also have  $\dots \leq (\sum i \in \{(u,v) \in \{0..<p\} \times \{0..<p\}. u \neq v \wedge$ 
 $\text{truncate-down } r \ u \leq c \wedge \text{truncate-down } r \ u = \text{truncate-down } r \ v\}.$ 
 $\text{measure } \Omega \{ \omega. \text{hash } p \ x \ \omega = \text{fst } i \wedge \text{hash } p \ y \ \omega = \text{snd } i \})$ 
    apply (rule measure-UNION-le)
    apply (rule finite-subset[where  $B = \{0..<p\} \times \{0..<p\}$ ], rule subsetI, simp
add:case-prod-beta mem-Times-iff, simp)
    by simp
  also have  $\dots \leq (\sum i \in \{(u,v) \in \{0..<p\} \times \{0..<p\}. u \neq v \wedge$ 
 $\text{truncate-down } r \ u \leq c \wedge \text{truncate-down } r \ u = \text{truncate-down } r \ v\}.$ 
 $\mathcal{P}(\omega \text{ in } \Omega. (\forall u \in \text{UNIV}. \text{hash } p \ (\text{if } u \text{ then } x \text{ else } y) \ \omega = (\text{if } u \text{ then } (\text{fst } i) \text{ else}$ 
 $(\text{snd } i))))))$ 
    apply (rule sum-mono)
    apply (rule pmf-mono)
    by (simp add:case-prod-beta)
  also have  $\dots \leq (\sum i \in \{(u,v) \in \{0..<p\} \times \{0..<p\}. u \neq v \wedge$ 
 $\text{truncate-down } r \ u \leq c \wedge \text{truncate-down } r \ u = \text{truncate-down } r \ v\}. 1/(\text{real}$ 
 $p)^2)$ 

```

```

apply (rule sum-mono)
apply (simp only:Ω-def)
apply (subst hash-prob-2[OF assms(1)])
  using a1-3 apply (simp add: inj-on-def)
  using a1-1 assms(3) a1-3 a1-2 apply auto[1]
by force+
also have ... = 1/(real p)2 *
  card {(u,v) ∈ {0..by simp
also have ... ≤ 1/(real p)2 *
  card {(u,v) ∈ {0..apply (rule mult-left-mono, rule of-nat-mono, rule card-mono)
  apply (rule finite-subset[where B={0..apply (rule subsetI, simp add:case-prod-beta)
by (metis a1-4, simp)
also have ... ≤ 1/(real p)2 * card (⋃ u' ∈ {u. u < p ∧ real u ≤ 2 * c}. {(u::nat,v::nat). u = u' ∧ abs (real u - real v) ≤ 2 * c * 2 powr (-real r) ∧ v < p ∧ v ≠ u'})
apply (rule mult-left-mono)
apply (rule of-nat-mono)
apply (rule card-mono, simp add:case-prod-beta)
apply (rule allI, rule impI)
apply (rule finite-subset[where B={0..apply (rule subsetI, simp add:case-prod-beta)
by simp
also have ... ≤ 1/(real p)2 * (∑ u' ∈ {u. u < p ∧ real u ≤ 2 * c}. card {(u::nat,v::nat). u = u' ∧ abs (real u - real v) ≤ 2 * c * 2 powr (-real r) ∧ v < p ∧ v ≠ u'})
apply (rule mult-left-mono)
apply (rule of-nat-mono)
by (rule card-UN-le, simp, simp)
also have ... = 1/(real p)2 * (∑ u' ∈ {u. u < p ∧ real u ≤ 2 * c}. card ((λx. (u',x)) ' {(v::nat). abs (real u' - real v) ≤ 2 * c * 2 powr (-real r) ∧ v < p ∧ v ≠ u'}))
apply (simp, rule disjI2, rule sum.cong, simp)
apply (simp, rule arg-cong[where f=card], subst set-eq-iff)
by blast
also have ... ≤ 1/(real p)2 * (∑ u' ∈ {u. u < p ∧ real u ≤ 2 * c}. card {(v::nat). abs (real u' - real v) ≤ 2 * c * 2 powr (-real r) ∧ v < p ∧ v ≠ u'})
apply (rule mult-left-mono)
apply (rule of-nat-mono, rule sum-mono, rule card-image-le, simp)
by simp
also have ... ≤ 1/(real p)2 * (∑ u' ∈ {u. u < p ∧ real u ≤ 2 * c}. card {(v::nat). abs (real u' - real v) ≤ 2 * c * 2 powr (-real r) ∧ v ≠ u'})

```



```

    apply (rule mult-left-mono)
    apply (rule of-nat-mono, rule sum-mono, rule card-mono)
    apply (rule count-nat-abs-diff-2(2), simp)
    by (rule subsetI, simp, simp)
  also have ... ≤ 1/(real p)2 * (∑ u' ∈ {u. u < p ∧ real u ≤ 2 * c}.
    2 * (2 * c * 2powr (-real r)))
    apply (rule mult-left-mono)
    apply (subst of-nat-sum)
    apply (rule sum-mono)
    apply (rule count-nat-abs-diff-2(1), simp)
    by simp
  also have ... ≤ 1/(real p)2 * (real (card {u. u ≤ nat (⌊2 * c⌋)}) * (2 * (2 *
c * 2powr (-real r))))
    apply (rule mult-left-mono)
    apply (subst sum-constant)
    apply (rule mult-right-mono)
    apply (rule of-nat-mono, rule card-mono, simp)
    apply (rule subsetI, simp) using c-ge-0 le-nat-floor apply blast
    apply (simp add: c-ge-0)
    by simp
  also have ... ≤ 1/(real p)2 * ((3 * c) * (2 * (2 * c * 2powr (-real r))))
    apply (rule mult-left-mono)
    apply (rule mult-right-mono)
    apply simp using assms(4) apply linarith
    by (simp add: c-ge-0)+
  also have ... = 12 * c2 * 2powr (-real r) / (real p)2
    by (simp add: ac-simps power2-eq-square)
  finally show measure Ω {ω. degree ω ≥ 1 ∧ truncate-down r (hash p x ω) ≤
c ∧
    truncate-down r (hash p x ω) = truncate-down r (hash p y ω)} ≤ 12 * c2 *
2powr (-real r) / (real p)2
    by simp
qed

have P(ω in measure-pmf Ω. ?l ω ∧ degree ω ≥ 1) ≤
  measure Ω (⋃ i ∈ {(x,y) ∈ M × M. x < y}. {ω.
    degree ω ≥ 1 ∧ truncate-down r (hash p (fst i) ω) ≤ c ∧
    truncate-down r (hash p (fst i) ω) = truncate-down r (hash p (snd i) ω)})
    apply (rule pmf-mono)
    apply (simp)
    by (metis linorder-neqE-nat)
  also have ... ≤ (∑ i ∈ {(x,y) ∈ M × M. x < y}. measure Ω
    {ω. degree ω ≥ 1 ∧ truncate-down r (hash p (fst i) ω) ≤ c ∧
    truncate-down r (hash p (fst i) ω) = truncate-down r (hash p (snd i) ω)})
    apply (rule measure-UNION-le)
    apply (rule finite-subset[where B=M × M], rule subsetI, simp add: case-prod-beta
mem-Times-iff)
    apply (rule finite-cartesian-product[OF f-M f-M])
    by simp

```

```

also have ...  $\leq (\sum i \in \{(x,y) \in M \times M. x < y\}. 12 * c^2 * 2^{\text{powr } (-\text{real } r)}) / (\text{real } p)^2$ 
apply (rule sum-mono)
using a1 by (simp add:case-prod-beta)
also have ...  $= (12 * c^2 * 2^{\text{powr } (-\text{real } r)} / (\text{real } p)^2) * \text{card } \{(x,y) \in M \times M. x < y\}$ 
by simp
also have ...  $\leq (12 * c^2 * 2^{\text{powr } (-\text{real } r)} / (\text{real } p)^2) * ((\text{real } (\text{card } M))^2 / \text{real } 2)$ 
apply (rule mult-left-mono)
apply (subst pos-le-divide-eq, simp)
apply (subst mult.commute)
apply (subst of-nat-mult[symmetric])
apply (subst card-ordered-pairs, rule finite-subset[OF assms(3)], simp)
apply (subst of-nat-power[symmetric], rule of-nat-mono)
apply (simp add:power2-eq-square)
by (simp add:c-ge-0)
also have ...  $= 6 * (\text{real } (\text{card } M))^2 * c^2 * 2^{\text{powr } (-\text{real } r)} / (\text{real } p)^2$ 
by (simp add:algebra-simps)
finally have a:  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. ?l \ \omega \wedge \text{degree } \omega \geq 1) \leq ?r1$  by simp

have b1: bounded-degree-polynomials (ZFact (int p)) 2  $\cap \{\omega. \text{length } \omega \leq \text{Suc } 0\}$ 
 $= \text{bounded-degree-polynomials } (ZFact (int p)) \ 1$ 
apply (rule order-antisym)
apply (rule subsetI, simp add:bounded-degree-polynomials-def)
by (rule subsetI, simp add:bounded-degree-polynomials-def, fastforce)

have b:  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. \text{degree } \omega < 1) \leq ?r2$ 
apply (simp add: $\Omega$ -def)
apply (subst measure-pmf-of-set)
apply (rule ne-bounded-degree-polynomials)
apply (rule fin-bounded-degree-polynomials[OF p-ge-0])
apply (subst card-bounded-degree-polynomials[OF p-ge-0], subst b1)
apply (subst card-bounded-degree-polynomials[OF p-ge-0])
apply (simp add:zfact-card[OF p-ge-0])
by (subst pos-divide-le-eq, simp add:p-ge-0, simp add:power2-eq-square)

have  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. ?l \ \omega) \leq$ 
 $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. ?l \ \omega \wedge \text{degree } \omega \geq 1) + \mathcal{P}(\omega \text{ in measure-pmf } \Omega. \text{degree } \omega < 1)$ 
by (rule pmf-add, simp, linarith)
also have ...  $\leq ?r1 + ?r2$  by (rule add-mono, metis a, metis b)
finally show ?thesis by simp
qed

lemma inters-compr:  $A \cap \{x. P \ x\} = \{x \in A. P \ x\}$ 
by blast

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

```

```

by (cases x, simp, simp)

theorem f0-alg-correct:
  assumes  $\varepsilon \in \{0 < .. < 1\}$ 
  assumes  $\delta \in \{0 < .. < 1\}$ 
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  defines  $M \equiv \text{fold } (\lambda a \text{ state. state } \gg f0\text{-update } a) \text{ as } (f0\text{-init } \delta \ \varepsilon \ n) \gg f0\text{-result}$ 
  shows  $\mathcal{P}(\omega \text{ in measure-pmf } M. |\omega - F \ 0 \ as| \leq \delta * F \ 0 \ as) \geq 1 - \text{of-rat } \varepsilon$ 
proof -
  define s where  $s = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 
  define t where  $t = \text{nat } \lceil 80 / (\text{real-of-rat } \delta)^2 \rceil$ 
  define p where  $p = \text{find-prime-above } (\text{max } n \ 19)$ 
  define r where  $r = \text{nat } (4 * \lceil \log 2 (1 / \text{real-of-rat } \delta) \rceil + 24)$ 
  define g where  $g = (\lambda S. \text{if card } S < t \text{ then rat-of-nat } (\text{card } S) \text{ else of-nat } t * \text{rat-of-nat } p / \text{rat-of-float } (\text{Max } S))$ 
  define g' where  $g' = (\lambda S. \text{if card } S < t \text{ then real } (\text{card } S) \text{ else real } t * \text{real } p / \text{Max } S)$ 
  define h where  $h = (\lambda \omega. \text{least } t ((\lambda x. \text{truncate-down } r (\text{hash } p \ x \ \omega)) \text{ ' set } as))$ 
  define  $\Omega_0$  where  $\Omega_0 = \text{prod-pmf } \{0..<s\} (\lambda \cdot. \text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ 2))$ 
  define  $\Omega_1$  where  $\Omega_1 = \text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ 2)$ 
  define m where  $m = \text{card } (\text{set } as)$ 

  define f where  $f = (\lambda r \ \omega. \text{card } \{x \in \text{set } as. \text{int } (\text{hash } p \ x \ \omega) \leq r\})$ 
  define  $\delta'$  where  $\delta' = 3 * \text{real-of-rat } \delta / 4$ 
  define a where  $a = \lfloor \text{real } t * p / (m * (1 + \delta')) \rfloor$ 
  define b where  $b = \lceil \text{real } t * p / (m * (1 - \delta')) - 1 \rceil$ 

  define has-no-collision where  $\text{has-no-collision} = (\lambda \omega. \forall x \in \text{set } as. \forall y \in \text{set } as. (\text{truncate-down } r (\text{hash } p \ x \ \omega) = \text{truncate-down } r (\text{hash } p \ y \ \omega) \longrightarrow x = y) \vee \text{truncate-down } r (\text{hash } p \ x \ \omega) > b)$ 

  have s-ge-0:  $s > 0$ 
  using assms(1) by (simp add:s-def)

  have t-ge-0:  $t > 0$ 
  using assms by (simp add:t-def)

  have  $\delta$ -ge-0:  $\delta > 0$  using assms by simp
  have  $\delta$ -le-1:  $\delta < 1$  using assms by simp

  have r-bound:  $4 * \log 2 (1 / \text{real-of-rat } \delta) + 24 \leq r$ 
  apply (simp add:r-def)
  apply (subst of-nat-nat)
  apply (rule add-nonneg-nonneg)
  apply (rule mult-nonneg-nonneg, simp)
  apply (subst zero-le-ceiling, subst log-divide, simp, simp, simp, simp add: $\delta$ -ge-0, simp)

```

```

    apply (subst log-less-one-cancel-iff, simp, simp add:δ-ge-0)
  by (rule order-less-le-trans[where y=1], simp add:δ-le-1, simp+)

have 1 ≤ 0 + (24::real) by simp
also have ... ≤ 4 * log 2 (1 / real-of-rat δ) + 24
  apply (rule add-mono, simp)
  apply (subst zero-le-log-cancel-iff)
  using assms by simp+
also have ... ≤ r using r-bound by simp
finally have r-ge-0: 1 ≤ r by simp

have 2 powr (−real r) ≤ 2 powr (−(4 * log 2 (1 / real-of-rat δ) + 24))
  apply (rule powr-mono) using r-bound apply linarith by simp
also have ... = 2 powr (−4 * log 2 (1 / real-of-rat δ) − 24)
  by (rule arg-cong2[where f=(powr)], simp, simp add:algebra-simps)
also have ... ≤ 2 powr (−1 * log 2 (1 / real-of-rat δ) − 4)
  apply (rule powr-mono)
  apply (rule diff-mono)
  using assms(2) by simp+
also have ... = real-of-rat δ / 16
  apply (subst powr-diff)
  apply (subst log-divide, simp, simp, simp, simp add:δ-ge-0, simp)
  by (subst powr-log-cancel, simp, simp, simp add:δ-ge-0, simp)
also have ... < real-of-rat δ / 8
  by (subst pos-divide-less-eq, simp, simp add:δ-ge-0)
finally have r-le-δ: 2 powr (−real r) < (real-of-rat δ) / 8
  by simp

have r-le-t2: 18 * 96 * (real t)2 * 2 powr (−real r) ≤
  18 * 96 * (80 / (real-of-rat δ)2 + 1)2 * 2 powr (−4 * log 2 (1 / real-of-rat δ)
− 24)
  apply (rule mult-mono)
  apply (rule mult-left-mono)
  apply (rule power-mono)
  apply (simp add:t-def) using t-def t-ge-0 apply linarith
  apply simp
  apply simp
  apply (rule powr-mono) using r-bound apply linarith
  by simp+
also have ... ≤ 18 * 96 * (80 / (real-of-rat δ)2 + 1 / (real-of-rat δ)2) * (2
powr (−4 * log 2 (1 / real-of-rat δ)) * 2 powr (−24))
  apply (rule mult-mono)
  apply (rule mult-left-mono)
  apply (rule power-mono)
  apply (rule add-mono, simp) using assms(2) apply (simp add: power-le-one)
  by (simp add:powr-diff)+
also have ... = 18 * 96 * (812 / (real-of-rat δ)4) * (2 powr (log 2 ((real-of-rat
δ)4)) * 2 powr (−24))
  apply (rule arg-cong2[where f=(*)])

```

```

    apply (rule arg-cong2[where f=(*)], simp)
  apply (simp add:power2-eq-square power4-eq-xxxx)
  apply (rule arg-cong2[where f=(*)])
  apply (rule arg-cong2[where f=(powr)], simp)
  apply (subst log-nat-power, simp add:δ-ge-0)
  apply (subst log-divide, simp, simp, simp, simp add:δ-ge-0)
  by simp+
also have ... = 18 * 96 * 812 * 2 powr (-24)
  apply (subst powr-log-cancel, simp, simp, simp) using δ-ge-0 apply blast
  apply (simp add:algebra-simps) using δ-ge-0 by blast
also have ... ≤ 1
  by simp
finally have r-le-t2: 18 * 96 * (real t)2 * 2 powr (-real r) ≤ 1
  by simp

have δ'-ge-0: δ' > 0 using assms by (simp add:δ'-def)
have δ'-le-1: δ' < 1
  apply (rule order-less-le-trans[where y=3/4])
  using assms by (simp add:δ'-def)+

have t ≤ 80 / (real-of-rat δ)2 + 1
  using t-def t-ge-0 by linarith
also have ... = 45 / (δ')2 + 1
  by (simp add:δ'-def algebra-simps power2-eq-square)
also have ... ≤ 45 / δ2 + 1 / δ2
  apply (rule add-mono, simp)
  apply (subst pos-le-divide-eq, simp add:δ'-def)
  using assms apply force
  apply (simp add:δ'-def algebra-simps)
  apply (subst power-le-one-iff)
  using assms apply simp
  apply (subst pos-divide-le-eq, simp, simp)
  apply (rule order-trans[where y=3])
  using assms(2) by simp+
also have ... = 46 / δ2
  by simp
finally have t-le-δ': t ≤ 46 / δ2 by simp

have 45 / δ2 = 80 / (real-of-rat δ)2
  by (simp add:δ'-def power2-eq-square)
also have ... ≤ t
  using t-ge-0 t-def of-nat-ceiling by blast
finally have t-ge-δ': 45 / δ2 ≤ t by simp

have p-prime: Factorial-Ring.prime p
  using p-def find-prime-above-is-prime by simp
have p-ge-18: p ≥ 18
  apply (rule order-trans[where y=19], simp)
  using p-def find-prime-above-lower-bound max.bounded-iff by blast

```

```

hence  $p\text{-ge-}0$ :  $p > 0$  by simp

have  $m \leq \text{card } \{0..<n\}$ 
  apply (subst m-def)
  apply (rule card-mono, simp)
  apply (rule subsetI)
  using assms(3) by simp
also have  $\dots \leq p$ 
  by (metis p-def find-prime-above-lower-bound card-atLeastLessThan diff-zero
max-def order-trans)
finally have  $m\text{-le-}p$ :  $m \leq p$  by simp

have  $xs\text{-le-}p$ :  $\bigwedge x. x \in \text{set } as \implies x < p$ 
  apply (rule order-less-le-trans[where  $y=n$ ])
  using assms(3) apply simp
  by (metis p-def find-prime-above-lower-bound max-def order-trans)

have  $m\text{-eq-}F\text{-}0$ :  $\text{real } m = \text{of-rat } (F\ 0\ as)$ 
  by (simp add: m-def F-def)

have  $\text{fin-}\omega\text{-}1$ :  $\text{finite } (\text{set-pmf } \Omega_1)$ 
  apply (simp add:  $\Omega_1$ -def)
  by (metis fin-bounded-degree-polynomials[OF  $p\text{-ge-}0$ ] ne-bounded-degree-polynomials
set-pmf-of-set)

have  $\text{exp-var-}f$ :  $\bigwedge a. a \geq -1 \implies a < \text{int } p \implies$ 
   $\text{prob-space.expectation } \Omega_1 (\lambda\omega. \text{real } (f\ a\ \omega)) = \text{real } m * (\text{real-of-int } a+1) / p \wedge$ 
   $\text{prob-space.variance } \Omega_1 (\lambda\omega. \text{real } (f\ a\ \omega)) \leq \text{real } m * (\text{real-of-int } a+1) / p$ 
proof -
  fix  $a :: \text{int}$ 
  assume  $a\text{-ge-}m1$ :  $a \geq -1$ 
  assume  $a\text{-le-}p$ :  $a < \text{int } p$ 
  have  $xs\text{-subs-}p$ :  $\text{set } as \subseteq \{0..<p\}$ 
    using  $xs\text{-le-}p$ 
    by (simp add: subset-iff)

  have  $\text{exp-single}$ :  $\bigwedge x. x \in \text{set } as \implies \text{prob-space.expectation } \Omega_1 (\lambda\omega. \text{of-bool } (\text{int } (\text{hash } p\ x\ \omega) \leq a)) =$ 
     $(\text{real-of-int } a+1) / \text{real } p$ 
  proof -
    fix  $x$ 
    assume  $x \in \text{set } as$ 
    hence  $x\text{-le-}p$ :  $x < p$  using  $xs\text{-le-}p$  by simp
    have  $\text{prob-space.expectation } \Omega_1 (\lambda\omega. \text{of-bool } (\text{int } (\text{hash } p\ x\ \omega) \leq a)) =$ 
       $\text{measure } \Omega_1 (\{\omega. \text{int } (\text{hash } p\ x\ \omega) \leq a\} \cap \text{space } \Omega_1)$ 
    apply (subst Bochner-Integration.integral-indicator[where  $M=\text{measure-pmf}$ 
 $\Omega_1$ , symmetric])
    apply (rule arg-cong2[where  $f=\text{integral}^L$ ], simp)
    by (rule ext, simp)

```

```

also have ... =  $\mathcal{P}(\omega \text{ in } \Omega_1. \text{hash } p \ x \ \omega \in \{k. \text{int } k \leq a\})$ 
  by simp
also have ... =  $\text{card } (\{k. \text{int } k \leq a\} \cap \{0..<p\}) / \text{real } p$ 
  apply (simp only:  $\Omega_1$ -def)
  by (rule hash-prob-range[OF p-prime x-le-p], simp)
also have ... =  $\text{card } \{0..<\text{nat } (a+1)\} / \text{real } p$ 
  apply (rule arg-cong2[where f=(/)])
  apply (rule arg-cong[where f=real], rule arg-cong[where f=card])
  apply (subst set-eq-iff, rule allI)
  apply (cases a ≥ 0)
  using a-le-p apply (simp, linarith)
  by simp+
also have ... =  $(\text{real-of-int } a+1) / \text{real } p$ 
  using a-ge-m1 by simp
finally show prob-space.expectation  $\Omega_1$  ( $\lambda\omega. \text{of-bool } (\text{int } (\text{hash } p \ x \ \omega) \leq a)$ )
=
   $(\text{real-of-int } a+1) / \text{real } p$ 
  by simp
qed
have prob-space.expectation  $\Omega_1$  ( $\lambda\omega. \text{real } (f \ a \ \omega)$ ) =
  prob-space.expectation  $\Omega_1$  ( $\lambda\omega. (\sum x \in \text{set as. of-bool } (\text{int } (\text{hash } p \ x \ \omega) \leq a))$ )
  by (simp add: f-def inters-compr)
also have ... =  $(\sum x \in \text{set as. prob-space.expectation } \Omega_1 (\lambda\omega. \text{of-bool } (\text{int } (\text{hash } p \ x \ \omega) \leq a)))$ 
  apply (rule Bochner-Integration.integral-sum)
  by (rule integrable-measure-pmf-finite[OF fin-omega-1])
also have ... =  $(\sum x \in \text{set as. } (a+1) / \text{real } p)$ 
  by (rule sum.cong, simp, subst exp-single, simp, simp)
also have ... =  $\text{real } m * (\text{real-of-int } a + 1) / \text{real } p$ 
  by (simp add: m-def)
finally have r-1: prob-space.expectation  $\Omega_1$  ( $\lambda\omega. \text{real } (f \ a \ \omega)$ ) =  $\text{real } m * (\text{real-of-int } a+1) / p$ 
  by simp

have prob-space.variance  $\Omega_1$  ( $\lambda\omega. \text{real } (f \ a \ \omega)$ ) =
  prob-space.variance  $\Omega_1$  ( $\lambda\omega. (\sum x \in \text{set as. of-bool } (\text{int } (\text{hash } p \ x \ \omega) \leq a))$ )
  by (simp add: f-def inters-compr)
also have ... =  $(\sum x \in \text{set as. prob-space.variance } \Omega_1 (\lambda\omega. \text{of-bool } (\text{int } (\text{hash } p \ x \ \omega) \leq a)))$ 
  apply (rule prob-space.var-sum-pairwise-indep-2, simp add: prob-space-measure-pmf,
  simp, simp)
  apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
  apply (rule prob-space.indep-vars-compose2[where Y= $\lambda i \ x. \text{of-bool } (\text{int } x \leq a)$  and  $M' = \lambda \cdot. \text{measure-pmf } (\text{pmf-of-set } \{0..<p\})$ ])
  apply (simp add: prob-space-measure-pmf)
  using hash-k-wise-indep[OF p-prime, where n=2] xs-subs-p
  apply (simp add: measure-pmf.k-wise-indep-vars-def  $\Omega_1$ -def)
  apply (metis le-refl order-trans subset-eq-atLeast0-lessThan-finite)
  by simp

```

```

also have ... ≤ (∑ x ∈ set as. (a+1)/real p)
  apply (rule sum-mono)
  apply (subst prob-space.variance-eq[OF prob-space.measure-pmf])
  apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
  apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
  apply (simp add:of-bool-square)
  apply (subst exp-single, simp)
  by simp
also have ... = real m * (real-of-int a + 1) / real p
  by (simp add:m-def)
finally have r-2: prob-space.variance Ω1 (λω. real (f a ω)) ≤ real m * (real-of-int
a+1) / p
  by simp
show
  prob-space.expectation Ω1 (λω. real (f a ω)) = real m * (real-of-int a+1) / p
^
  prob-space.variance Ω1 (λω. real (f a ω)) ≤ real m * (real-of-int a+1) / p
  using r-1 r-2 by auto
qed

have exp-f: ∧a. a ≥ -1 ⇒ a < int p ⇒ prob-space.expectation Ω1 (λω. real
(f a ω)) =
  real m * (real-of-int a+1) / p using exp-var-f by blast

have var-f: ∧a. a ≥ -1 ⇒ a < int p ⇒ prob-space.variance Ω1 (λω. real (f
a ω)) ≤
  real m * (real-of-int a+1) / p using exp-var-f by blast

have b: P(ω in measure-pmf Ω1.
  of-rat δ * real-of-rat (F 0 as) < |g' (h ω) - of-rat (F 0 as)|) ≤ 1/3
proof (cases card (set as) ≥ t)
case True
  hence t-le-m: t ≤ card (set as) by simp
  have m-ge-0: real m > 0
    using m-def True t-ge-0 by simp

  have b-le-tpm : b ≤ real t * real p / (real m * (1 - δ'))
    by (simp add:b-def)
  also have ... ≤ real t * real p / (real m * (1/4))
    apply (rule divide-left-mono)
    apply (rule mult-left-mono)
    using assms apply (simp add:δ'-def)
    using m-ge-0 δ'-le-1 by (auto intro!:mult-pos-pos)
  finally have b-le-tpm: b ≤ 4 * real t * real p / real m
    by (simp add:algebra-simps)

have a-ge-0: a ≥ 0
  apply (simp add:a-def)
  apply (rule divide-nonneg-nonneg, simp)

```



```

    using  $\delta'$ -ge-0 by simp
  have b-ge-0:  $b > 0$ 
  apply (simp add: b-def)
  apply (subst pos-less-divide-eq)
  apply (rule mult-pos-pos)
  using True m-def t-ge-0 apply simp
  using  $\delta'$ -le-1 apply simp
  apply simp
  apply (subst mult.commute)
  apply (rule order-less-le-trans[where  $y = \text{real } m$ ]) using  $\delta'$ -ge-0 m-ge-0 apply
simp
  apply (rule order-trans[where  $y = 1 * \text{real } p$ ]) using m-le-p apply simp
  apply (rule mult-right-mono) using t-ge-0 apply simp
  by simp
  hence b-ge-1:  $\text{real-of-int } b \geq 1$ 
  by linarith

  have a-le-p:  $a < \text{real } p$ 
  apply (rule order-le-less-trans[where  $y = \text{real } t * \text{real } p / (\text{real } m * (1 + \delta'))$ ])
  apply (simp add: a-def)
  apply (subst pos-divide-less-eq) using m-ge-0  $\delta'$ -ge-0 apply force
  apply (subst mult.commute)
  apply (rule mult-strict-left-mono)
  apply (rule order-le-less-trans[where  $y = \text{real } m$ ]) using True m-def apply
linarith
  using  $\delta'$ -ge-0 m-ge-0 apply force
  using p-ge-0 by simp
  hence a-le-p:  $a < \text{int } p$ 
  by linarith

  have  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. f \ a \ \omega \geq t) \leq$ 
 $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \text{abs } (\text{real } (f \ a \ \omega) - \text{prob-space.expectation } (\text{measure-pmf } \Omega_1) (\lambda \omega. \text{real } (f \ a \ \omega))))$ 
 $\geq 3 * \text{sqrt } (m * (\text{real-of-int } a + 1) / p)$ 
  proof (rule prob-space.prob-mono[OF prob-space.measure-pmf in-events-pmf])
  fix  $\omega$ 
  assume  $\omega \in \text{space } (\text{measure-pmf } \Omega_1)$ 
  assume t-le:  $t \leq f \ a \ \omega$ 
  have  $\text{real } m * (\text{of-int } a + 1) / p = \text{real } m * (\text{of-int } a) / p + \text{real } m / p$ 
  by (simp add: algebra-simps add-divide-distrib)
  also have  $\dots \leq \text{real } m * (\text{real } t * \text{real } p / (\text{real } m * (1 + \delta'))) / \text{real } p + 1$ 
  apply (rule add-mono)
  apply (rule divide-right-mono)
  apply (rule mult-mono, simp, simp add: a-def, simp, simp add: a-ge-0)
  apply (simp)
  using m-le-p by (simp add: p-ge-0)
  also have  $\dots \leq \text{real } t / (1 + \delta') + 1$ 
  apply (rule add-mono)
  apply (subst pos-le-divide-eq) using  $\delta'$ -ge-0 apply simp

```

```

    by simp+
  finally have a-le-1: real m * (of-int a + 1) / p ≤ t / (1 + δ') + 1
    by simp
  have a-le: 3 * sqrt (real m * (of-int a + 1) / real p) + real m * (of-int a +
1) / real p ≤
    3 * sqrt (t / (1 + δ') + 1) + (t / (1 + δ') + 1)
  apply (rule add-mono)
  apply (rule mult-left-mono)
  apply (subst real-sqrt-le-iff, simp add: a-le-1)
  apply simp
  by (simp add: a-le-1)
also have ... ≤ 3 * sqrt (real t + 1) + ((t - δ' * t / (1 + δ')) + 1)
  apply (rule add-mono)
  apply (rule mult-left-mono)
  apply (subst real-sqrt-le-iff, simp)
  apply (subst pos-divide-le-eq) using δ'-ge-0 apply simp
  using δ'-ge-0 apply (simp add: t-ge-0)
  apply simp
  apply (rule add-mono)
  apply (subst pos-divide-le-eq) using δ'-ge-0 apply simp
  apply (subst left-diff-distrib, simp, simp add: algebra-simps)
  using δ'-ge-0 by simp+
also have ... ≤ 3 * sqrt (46 / δ2 + 1 / δ2) + (t - δ' * t / 2) + 1 / δ'
  apply (subst add.assoc[symmetric])
  apply (rule add-mono)
  apply (rule add-mono)
  apply (rule mult-left-mono)
  apply (subst real-sqrt-le-iff)
  apply (rule add-mono, metis t-le-δ')
  apply (subst pos-le-divide-eq) using δ'-ge-0 apply simp
  apply (metis δ'-le-1 δ'-ge-0 less-eq-real-def mult-1 power-le-one)
  apply simp
  apply simp
  apply (subst pos-le-divide-eq) using δ'-ge-0 apply simp
  using δ'-le-1 δ'-ge-0
  apply (metis add-mono less-eq-real-def mult-eq-0-iff mult-left-mono of-nat-0-le-iff
one-add-one)
    using δ'-le-1 δ'-ge-0 by simp
  also have ... ≤ (21 / δ' + (t - 45 / (2 * δ'))) + 1 / δ'
  apply (rule add-mono)
  apply (rule add-mono)
  apply (simp add: real-sqrt-divide, subst abs-of-nonneg) using δ'-ge-0 apply
linarith
    using δ'-ge-0 apply (simp add: divide-le-cancel)
    apply (rule real-le-lsqr, simp, simp, simp)
    apply simp
    apply (metis δ'-ge-0 t-ge-δ' less-eq-real-def mult-left-mono power2-eq-square
real-divide-square-eq times-divide-eq-right)
    by simp

```

```

    also have ... ≤ t using δ'-ge-0 by simp
    also have ... ≤ f a ω using t-le by simp
    finally have t-le: 3 * sqrt (real m * (of-int a + 1) / real p) ≤ f a ω - real
m * (of-int a + 1) / real p
      by (simp add: algebra-simps)
    show 3 * sqrt (real m * (real-of-int a + 1) / real p) ≤
|real (f a ω) - measure-pmf.expectation Ω1 (λω. real (f a ω))|
      apply (subst exp-f) using a-ge-0 a-le-p True apply (simp, simp)
      apply (subst abs-ge-iff)
      using t-le by blast
  qed
  also have ... ≤ prob-space.variance (measure-pmf Ω1) (λω. real (f a ω))
/ (3 * sqrt (real m * (of-int a + 1) / real p))2
    apply (rule prob-space.Chebyshev-inequality)
      apply (metis prob-space-measure-pmf)
      apply simp
      apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
      apply simp
      using t-ge-0 a-ge-0 p-ge-0 m-ge-0 m-eq-F-0 by auto
  also have ... ≤ 1/9
    apply (subst pos-divide-le-eq) using a-ge-0 p-ge-0 m-ge-0 m-eq-F-0 apply
force
      apply simp
      apply (subst real-sqrt-pow2) using a-ge-0 p-ge-0 m-ge-0 m-eq-F-0 apply force
      apply (rule var-f) using a-ge-0 apply linarith
      using a-le-p by simp
  finally have case-1: P(ω in measure-pmf Ω1. f a ω ≥ t) ≤ 1/9
    by simp

  have case-2: P(ω in measure-pmf Ω1. f b ω < t) ≤ 1/9
  proof (cases b < p)
    case True
      have P(ω in measure-pmf Ω1. f b ω < t) ≤
P(ω in measure-pmf Ω1. abs (real (f b ω) - prob-space.expectation (measure-pmf
Ω1) (λω. real (f b ω)))
      ≥ 3 * sqrt (m * (real-of-int b + 1) / p))
    proof (rule prob-space.prob-mono[OF prob-space-measure-pmf in-events-pmf])
      fix ω
      assume ω ∈ space (measure-pmf Ω1)
      have aux: (real t + 3 * sqrt (real t / (1 - δ') + 1)) * (1 - δ') =
real t - δ' * t + 3 * ((1 - δ') * sqrt (real t / (1 - δ') + 1))
        by (simp add: algebra-simps)
      also have ... = real t - δ' * t + 3 * sqrt ((1 - δ')2 * (real t / (1 - δ') +
1))
        1))
        apply (subst real-sqrt-mult)
        apply (subst real-sqrt-abs)
        apply (subst abs-of-nonneg)
        using δ'-le-1 by simp+
      also have ... = real t - δ' * t + 3 * sqrt (real t * (1 - δ') + (1 - δ')2)

```

```

    by (simp add: power2-eq-square distrib-left)
  also have ... ≤ real t - 45 / δ' + 3 * sqrt ( real t + 1 )
    apply (rule add-mono, simp)
    apply (subst mult.commute, subst pos-divide-le-eq[symmetric])
    using δ'-ge-0 apply blast
    using t-ge-δ' apply (simp add: power2-eq-square)
  apply simp
  apply (rule add-mono)
    using δ'-le-1 δ'-ge-0 by (simp add: power-le-one t-ge-0)+
  also have ... ≤ real t - 45 / δ' + 3 * sqrt ( 46 / δ'^2 + 1 / δ'^2 )
    apply (rule add-mono, simp)
    apply (rule mult-left-mono)
    apply (subst real-sqrt-le-iff)
    apply (rule add-mono, metis t-le-δ')
  apply (meson δ'-ge-0 δ'-le-1 le-divide-eq-1-pos less-eq-real-def power-le-one-iff
zero-less-power)
    by simp
  also have ... = real t + (3 * sqrt(47) - 45) / δ'
    apply (simp add: real-sqrt-divide)
    apply (subst abs-of-nonneg)
    using δ'-ge-0 by (simp add: diff-divide-distrib)+
  also have ... ≤ t
    apply simp
    apply (subst pos-divide-le-eq)
    using δ'-ge-0 apply simp
    apply simp
    by (rule real-le-lsqrt, simp+)
  finally have aux: (real t + 3 * sqrt (real t / (1 - δ') + 1)) * (1 - δ') ≤
real t
    by simp
  assume t-ge: f b ω < t
  have real (f b ω) + 3 * sqrt (real m * (real-of-int b + 1) / real p)
    ≤ real t + 3 * sqrt (real m * real-of-int b / real p + 1)
    apply (rule add-mono)
    using t-ge apply linarith
    using m-le-p by (simp add: algebra-simps add-divide-distrib p-ge-0)
  also have ... ≤ real t + 3 * sqrt (real m * (real t * real p / (real m * (1 -
δ')))) / real p + 1)
    apply (rule add-mono, simp)
    apply (rule mult-left-mono)
    apply (subst real-sqrt-le-iff)
    apply (rule add-mono)
    apply (rule divide-right-mono)
    apply (rule mult-left-mono)
    apply (simp add: b-def)
    by simp+
  also have ... ≤ real t + 3 * sqrt(real t / (1 - δ') + 1)
    apply (simp add: p-ge-0)
    using t-ge-0 t-le-m m-def by linarith

```

```

also have ... ≤ real t / (1-δ')
  apply (subst pos-le-divide-eq) using δ'-le-1 aux by simp+
also have ... = real m * (real t * real p / (real m * (1-δ')) / real p
  apply (simp add:p-ge-0)
  using t-ge-0 t-le-m m-def by linarith
also have ... ≤ real m * (real-of-int b + 1) / real p
  apply (rule divide-right-mono)
  apply (rule mult-left-mono)
  by (simp add:b-def)+
finally have t-ge: real (f b ω) + 3 * sqrt (real m * (real-of-int b + 1) / real
p)
  ≤ real m * (real-of-int b + 1) / real p
  by simp
show 3 * sqrt (real m * (real-of-int b + 1) / real p) ≤
|real (f b ω) - measure-pmf.expectation Ω1 (λω. real (f b ω))|
  apply (subst exp-f) using b-ge-0 True apply (simp, simp)
  apply (subst abs-ge-iff)
  using t-ge by force
qed
also have ... ≤ prob-space.variance (measure-pmf Ω1) (λω. real (f b ω))
/ (3 * sqrt (real m * (real-of-int b + 1) / real p))2
  apply (rule prob-space.Chebyshev-inequality)
  apply (metis prob-space-measure-pmf)
  apply simp
  apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
  apply simp
  using t-ge-0 b-ge-0 p-ge-0 m-ge-0 m-eq-F-0 by auto
also have ... ≤ 1/9
  apply (subst pos-divide-le-eq)
  using b-ge-0 p-ge-0 m-ge-0 m-eq-F-0 apply force
  apply simp
  apply (subst real-sqrt-pow2)
  using b-ge-0 p-ge-0 m-ge-0 m-eq-F-0 apply force
  apply (rule var-f) using b-ge-0 apply linarith
  using True by simp
finally show ?thesis
  by simp
next
case False
have  $\mathcal{P}(\omega \text{ in } \Omega_1. f b \omega < t) \leq \mathcal{P}(\omega \text{ in } \Omega_1. \text{False})$ 
proof (rule pmf-mono-1)
  fix ω
  assume a-1:ω ∈ {ω ∈ space (measure-pmf Ω1). f b ω < t}
  assume a-2:ω ∈ set-pmf Ω1
  have a:∧x. x < p ⇒ hash p x ω < p
  using hash-range[OF p-ge-0] a-2
  by (simp add:Ω1-def set-pmf-of-set[OF ne-bounded-degree-polynomials
fin-bounded-degree-polynomials[OF p-ge-0]])
  have t ≤ card (set as)

```

```

    using True by simp
  also have ...  $\leq f\ b\ \omega$ 
    apply (simp add:f-def)
    apply (rule card-mono, simp)
    apply (rule subsetI)
    by (metis (no-types, lifting) False a xs-le-p linorder-linear mem-Collect-eq
of-nat-less-iff order-le-less-trans)
  also have ...  $< t$  using a-1 by simp
  finally have False by auto
  thus  $\omega \in \{\omega \in \text{space } (\text{measure-pmf } \Omega_1). \text{ False}\}$ 
    by simp
qed
also have ...  $= 0$  by auto
finally show ?thesis by simp
qed

have  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \neg \text{has-no-collision } \omega) \leq$ 
 $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \exists x \in \text{set as}. \exists y \in \text{set as}. x \neq y \wedge$ 
 $\text{truncate-down } r\ (\text{real } (\text{hash } p\ x\ \omega)) \leq \text{real-of-int } b \wedge$ 
 $\text{truncate-down } r\ (\text{real } (\text{hash } p\ x\ \omega)) = \text{truncate-down } r\ (\text{real } (\text{hash } p\ y\ \omega)))$ 
  apply (rule pmf-mono-1)
  apply (simp add:has-no-collision-def  $\Omega_1$ -def)
  by force
also have ...  $\leq 6 * (\text{real } (\text{card } (\text{set as})))^2 * (\text{real-of-int } b)^2$ 
 $* 2^{\text{powr} - \text{real } r} / (\text{real } p)^2 + 1 / \text{real } p$ 
  apply (simp only:  $\Omega_1$ -def)
  apply (rule f0-collision-prob[where c=real-of-int b])
    apply (metis p-prime)
    apply (rule subsetI, simp add:xs-le-p)
    apply (metis b-ge-1)
  by (metis r-ge-0)
also have ...  $\leq 6 * (\text{real } m)^2 * (\text{real-of-int } b)^2 * 2^{\text{powr} - \text{real } r} / (\text{real } p)^2 +$ 
 $1 / \text{real } p$ 
  apply (rule add-mono)
  apply (rule divide-right-mono)
  apply (rule mult-right-mono)
  apply (rule mult-mono)
    apply (simp add:m-def)
    apply (rule power-mono, simp)
  using b-ge-0 by simp+
also have ...  $\leq 6 * (\text{real } m)^2 * (4 * \text{real } t * \text{real } p / \text{real } m)^2 * (2^{\text{powr} - \text{real } r} /$ 
 $(\text{real } p)^2 + 1 / \text{real } p$ 
  apply (rule add-mono)
  apply (rule divide-right-mono)
  apply (rule mult-right-mono)
  apply (rule mult-left-mono)
  apply (simp add:b-def)
  using b-def b-ge-1 b-le-tpm apply force
  apply simp

```

```

    apply simp
    apply simp
  by simp
also have ... = 96 * (real t)2 * (2 powr - real r) + 1 / real p
  using p-ge-0 m-ge-0 t-ge-0 by (simp add: algebra-simps power2-eq-square)
also have ... ≤ 1/18 + 1/18
  apply (rule add-mono)
  apply (subst pos-le-divide-eq, simp)
  using r-le-t2 apply simp
  using p-ge-18 by simp
also have ... = 1/9 by (simp)
finally have case-3:  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \neg \text{has-no-collision } \omega) \leq 1/9$ 
  by simp

have  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1.$ 
   $\text{real-of-rat } \delta * \text{real-of-rat } (F \ 0 \ as) < |g' (h \ \omega) - \text{real-of-rat } (F \ 0 \ as)|) \leq$ 
 $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. f \ a \ \omega \geq t \vee f \ b \ \omega < t \vee \neg(\text{has-no-collision } \omega))$ 
proof (rule prob-space.prob-mono[OF prob-space.measure-pmf in-events-pmf],
rule ccontr)
  fix  $\omega$ 
  assume  $\omega \in \text{space } (\text{measure-pmf } \Omega_1)$ 
  assume est:  $\text{real-of-rat } \delta * \text{real-of-rat } (F \ 0 \ as) < |g' (h \ \omega) - \text{real-of-rat } (F \ 0$ 
as)|
  assume  $\neg(t \leq f \ a \ \omega \vee f \ b \ \omega < t \vee \neg \text{has-no-collision } \omega)$ 
  hence lb:  $f \ a \ \omega < t$  and ub:  $f \ b \ \omega \geq t$  and no-col:  $\text{has-no-collision } \omega$  by
simp+

  define y where  $y = \text{nth-mset } (t-1) \ \{\# \text{int } (\text{hash } p \ x \ \omega). \ x \in \# \text{mset-set } (\text{set}$ 
as)\#\}
  define y' where  $y' = \text{nth-mset } (t-1) \ \{\# \text{truncate-down } r \ (\text{hash } p \ x \ \omega). \ x$ 
 $\in \# \text{mset-set } (\text{set } as)\#\}$ 

  have  $a < y$ 
  apply (subst y-def, rule nth-mset-bound-left-excl)
  apply (simp)
  using True t-ge-0 apply linarith
  using lb
  by (simp add: f-def swap-filter-image count-le-def)
  hence rank-t-lb:  $a + 1 \leq y$ 
  by linarith

  have rank-t-ub:  $y \leq b$ 
  apply (subst y-def, rule nth-mset-bound-right)
  apply simp using True t-ge-0 apply linarith
  using ub t-ge-0
  by (simp add: f-def swap-filter-image count-le-def)

  have y-ge-0:  $\text{real-of-int } y \geq 0$  using rank-t-lb a-ge-0 by linarith
  have y'-eq:  $y' = \text{truncate-down } r \ y$ 

```

```

      apply (subst y-def, subst y'-def, subst nth-mset-commute-mono[where
f=( $\lambda x. \text{truncate-down } r \text{ (of-int } x)$ )])
      apply (metis truncate-down-mono mono-def of-int-le-iff)
      apply simp using True t-ge-0 apply linarith
      by (simp add: multiset.map-comp comp-def)
      have real-of-int (a+1) * (1 - 2 powr -real r)  $\leq$  real-of-int y * (1 - 2 powr
(-real r))
      apply (rule mult-right-mono)
      using rank-t-lb of-int-le-iff apply blast
      apply simp
      apply (subst two-powr-0[symmetric])
      by (rule powr-mono, simp, simp)
    also have ...  $\leq$  y'
      apply (subst y'-eq)
      using truncate-down-pos[OF y-ge-0] by simp
    finally have rank-t-lb': (a+1) * (1 - 2 powr (-real r))  $\leq$  y' by simp

    have y'  $\leq$  real-of-int y
      by (subst y'-eq, rule truncate-down-le, simp)
    also have ...  $\leq$  real-of-int b
      using rank-t-ub of-int-le-iff by blast
    finally have rank-t-ub': y'  $\leq$  b
      by simp

    have 0 < (a+1) * (1-2 powr (-real r))
      apply (rule mult-pos-pos)
      using a-ge-0 apply linarith
      apply simp
      apply (subst two-powr-0[symmetric])
      apply (rule powr-less-mono)
      using r-ge-0 by auto
    hence y'-pos: y' > 0 using rank-t-lb' by linarith

    have no-col':  $\bigwedge x. x \leq y' \implies \text{count } \{\# \text{truncate-down } r \text{ (real (hash } p \ x \ \omega))\} \leq 1$ 
    x  $\in \#$  mset-set (set as)#}
      apply (subst count-image-mset, simp add: vimage-def card-le-Suc0-iff-eq)
      using rank-t-ub' no-col apply (subst (asm) has-no-collision-def)
      by force

    have h-1: Max (h  $\omega$ ) = y'
      apply (simp add: h-def y'-def)
      apply (subst nth-mset-max)
      using True t-ge-0 apply simp
      using no-col' apply (simp add: y'-def)
      using t-ge-0
      by simp

    have card (h  $\omega$ ) = card (least ((t-1)+1) (set-mset {#truncate-down r (hash
p x  $\omega$ ). x  $\in \#$  mset-set (set as)#}))

```



```

    using t-ge-0
    by (simp add:h-def)
  also have ... = (t-1) + 1
    apply (rule nth-mset-max(2))
    using True t-ge-0 apply simp
    using no-col' by (simp add:y'-def)
  also have ... = t using t-ge-0 by simp
  finally have h-2: card (h  $\omega$ ) = t
    by simp
  have h-3:  $g' (h \omega) = \text{real } t * \text{real } p / y'$ 
    using h-2 h-1 by (simp add:g'-def)

  have (real t) * real p  $\leq (1 + \delta') * \text{real } m * ((\text{real } t) * \text{real } p / (\text{real } m * (1 + \delta')))$ 
    apply (simp)
    using  $\delta'$ -le-1 m-def True t-ge-0  $\delta'$ -ge-0 by linarith
  also have ...  $\leq (1 + \delta') * m * (a+1)$ 
    apply (rule mult-left-mono)
    apply (simp add:a-def)
    using  $\delta'$ -ge-0 by simp
  also have ...  $< ((1 + \text{real-of-rat } \delta) * (1 - \text{real-of-rat } \delta/8)) * m * (a+1)$ 
    apply (rule mult-strict-right-mono)
    apply (rule mult-strict-right-mono)
    apply (simp add: $\delta'$ -def distrib-left distrib-right left-diff-distrib right-diff-distrib)
    using True m-def t-ge-0 a-ge-0 assms(2) by auto
  also have ...  $\leq ((1 + \text{real-of-rat } \delta) * (1 - 2 \text{ powr } (-r))) * m * (a+1)$ 
    apply (rule mult-right-mono)
    apply (rule mult-right-mono)
    apply (rule mult-left-mono)
    using r-le- $\delta$  assms(2) a-ge-0 by auto
  also have ... =  $(1 + \text{real-of-rat } \delta) * m * ((a+1) * (1 - 2 \text{ powr } (-\text{real } r)))$ 
    by simp
  also have ...  $\leq (1 + \text{real-of-rat } \delta) * m * y'$ 
    apply (rule mult-left-mono, metis rank-t-lb')
    using assms by simp
  finally have  $\text{real } t * \text{real } p < (1 + \text{real-of-rat } \delta) * m * y'$  by simp
  hence f-1:  $g' (h \omega) < (1 + \text{real-of-rat } \delta) * m$ 
    apply (simp add: h-3)
    by (subst pos-divide-less-eq, metis y'-pos, simp)
  have  $(1 - \text{real-of-rat } \delta) * m * y' \leq (1 - \text{real-of-rat } \delta) * m * b$ 
    apply (rule mult-left-mono, metis rank-t-ub')
    using assms by simp
  also have ... =  $((1 - \text{real-of-rat } \delta)) * (\text{real } m * b)$ 
    by simp
  also have ...  $< (1 - \delta') * (\text{real } m * b)$ 
    apply (rule mult-strict-right-mono)
    apply (simp add:  $\delta'$ -def algebra-simps)
    using assms apply simp
    using r-le- $\delta$  m-eq-F-0 m-ge-0 b-ge-0 by simp

```

```

also have ...  $\leq (1-\delta') * (\text{real } m * (\text{real } t * \text{real } p / (\text{real } m * (1-\delta'))))$ 
  apply (rule mult-left-mono)
  apply (rule mult-left-mono)
  apply (simp add:b-def, simp)
  using  $\delta'\text{-ge-0 } \delta'\text{-le-1}$  by force
also have ... =  $\text{real } t * \text{real } p$ 
  apply (simp)
  using  $\delta'\text{-ge-0 } \delta'\text{-le-1 } t\text{-ge-0 } p\text{-ge-0}$  apply simp
  using True m-def order-less-le-trans by blast
finally have  $(1 - \text{real-of-rat } \delta) * m * y' < \text{real } t * \text{real } p$  by simp
hence f-2:  $g'(h \ \omega) > (1 - \text{real-of-rat } \delta) * m$ 
  apply (simp add: h-3)
  by (subst pos-less-divide-eq, metis y'-pos, simp)
have abs  $(g'(h \ \omega) - \text{real-of-rat } (F \ 0 \ as)) < \text{real-of-rat } \delta * (\text{real-of-rat } (F \ 0 \ as))$ 
as))
  apply (subst abs-less-iff) using f-1 f-2
  by (simp add:algebra-simps m-eq-F-0)
thus False
  using est by linarith
qed
also have ...  $\leq 1/9 + (1/9 + 1/9)$ 
  apply (rule prob-space.prob-sub-additiveI, simp add:prob-space-measure-pmf,
simp, simp)
  apply (rule case-1)
  apply (rule prob-space.prob-sub-additiveI, simp add:prob-space-measure-pmf,
simp, simp)
  by (rule case-2, rule case-3)
also have ... =  $1/3$  by simp
finally show ?thesis by simp
next
case False
have  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \text{real-of-rat } \delta * \text{real-of-rat } (F \ 0 \ as) < |g'(h \ \omega) - \text{real-of-rat } (F \ 0 \ as)|) \leq$ 
 $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_1. \exists x \in \text{set } as. \exists y \in \text{set } as. x \neq y \wedge$ 
 $\text{truncate-down } r (\text{real } (\text{hash } p \ x \ \omega)) \leq \text{real } p \wedge$ 
 $\text{truncate-down } r (\text{real } (\text{hash } p \ x \ \omega)) = \text{truncate-down } r (\text{real } (\text{hash } p \ y \ \omega)))$ 
proof (rule pmf-mono-1)
  fix  $\omega$ 
  assume a: $\omega \in \{\omega \in \text{space } (\text{measure-pmf } \Omega_1).$ 
 $\text{real-of-rat } \delta * \text{real-of-rat } (F \ 0 \ as) < |g'(h \ \omega) - \text{real-of-rat } (F \ 0 \ as)|\}$ 
  assume b: $\omega \in \text{set-pmf } \Omega_1$ 
  have a-1:  $\text{card } (\text{set } as) < t$  using False by auto
  have a-2: $\text{card } (h \ \omega) = \text{card } ((\lambda x. \text{truncate-down } r (\text{real } (\text{hash } p \ x \ \omega))) \text{ ` } (\text{set } as))$ 
    apply (simp add:h-def)
    apply (subst card-least, simp)
    apply (rule min.absorb4)
    using card-image-le a-1 order-le-less-trans[OF - a-1] by blast
  have  $\text{card } (h \ \omega) < t$ 

```

```

    by (metis List.finite-set a-1 a-2 card-image-le order-le-less-trans)
  hence  $g' (h \ \omega) = \text{card } (h \ \omega)$  by (simp add:g'-def)
  hence  $\text{card } (h \ \omega) \neq \text{real-of-rat } (F \ 0 \ as)$ 
    using a assms(2) apply simp
  by (metis abs-zero cancel-comm-monoid-add-class.diff-cancel of-nat-less-0-iff
pos-prod-lt zero-less-of-rat-iff)
  hence  $\text{card } (h \ \omega) \neq \text{card } (\text{set } as)$ 
    using m-def m-eq-F-0 by linarith
  hence  $\neg \text{inj-on } (\lambda x. \text{truncate-down } r \ (\text{real } (\text{hash } p \ x \ \omega))) \ (\text{set } as)$ 
    apply (simp add:a-2)
    using card-image by blast
  moreover have  $\bigwedge x. x \in \text{set } as \implies \text{truncate-down } r \ (\text{real } (\text{hash } p \ x \ \omega)) \leq$ 
    real p
  proof -
    fix x
    assume a:x  $\in \text{set } as$ 
    show  $\text{truncate-down } r \ (\text{real } (\text{hash } p \ x \ \omega)) \leq \text{real } p$ 
      apply (rule truncate-down-le)
      using hash-range[OF p-ge-0 - xs-le-p[OF a]] b
      apply (simp add: $\Omega_1$ -def set-pmf-of-set[OF ne-bounded-degree-polynomials
fin-bounded-degree-polynomials[OF p-ge-0]])
      using le-eq-less-or-eq by blast
    qed
  ultimately show  $\omega \in \{\omega \in \text{space } (\text{measure-pmf } \Omega_1). \exists x \in \text{set } as. \exists y \in \text{set}$ 
    as.  $x \neq y \wedge$ 
       $\text{truncate-down } r \ (\text{real } (\text{hash } p \ x \ \omega)) \leq \text{real } p \wedge$ 
       $\text{truncate-down } r \ (\text{real } (\text{hash } p \ x \ \omega)) = \text{truncate-down } r \ (\text{real } (\text{hash } p \ y \ \omega))\}$ 
    apply (simp add:inj-on-def) by blast
  qed
  also have  $\dots \leq 6 * (\text{real } (\text{card } (\text{set } as)))^2 * (\text{real } p)^2 * 2^{\text{powr } - \text{real } r} / (\text{real}$ 
     $p)^2 + 1 / \text{real } p$ 
    apply (simp only: $\Omega_1$ -def)
    apply (rule f0-collision-prob)
    apply (metis p-prime)
    apply (rule subsetI, simp add:xs-le-p)
    using p-ge-0 r-ge-0 by simp+
  also have  $\dots = 6 * (\text{real } (\text{card } (\text{set } as)))^2 * 2^{\text{powr } (- \text{real } r)} + 1 / \text{real } p$ 
    apply (simp add:ac-simps power2-eq-square)
    using p-ge-0 by blast
  also have  $\dots \leq 6 * (\text{real } t)^2 * 2^{\text{powr } (- \text{real } r)} + 1 / \text{real } p$ 
    apply (rule add-mono)
    apply (rule mult-right-mono)
    apply (rule mult-left-mono)
    apply (rule power-mono) using False apply simp
  by simp+
  also have  $\dots \leq 1/6 + 1/6$ 
    apply (rule add-mono)
    apply (subst pos-le-divide-eq, simp)
    using r-le-t2 apply simp

```

```

    using p-ge-18 by simp
    also have ...  $\leq 1/3$  by simp
    finally show ?thesis by simp
qed

have f0-result-elim:  $\bigwedge x. f0\text{-result } (s, t, p, r, x, \lambda i \in \{0..<s\}. f0\text{-sketch } p \ r \ t \ (x \ i) \ as) =$ 
  return-pmf (median ( $\lambda i. g \ (f0\text{-sketch } p \ r \ t \ (x \ i) \ as)$ ) s)
  apply (simp add:g-def)
  apply (rule median-cong)
  by simp

have real-g-2:  $\bigwedge \omega. \text{real-of-float } 'f0\text{-sketch } p \ r \ t \ \omega \ as = h \ \omega$ 
  apply (simp add:g-def g'-def h-def f0-sketch-def)
  apply (subst least-mono-commute, simp)
  apply (meson less-float.rep-eq strict-mono-onI)
  by (simp add:image-comp float-of-inverse[OF truncate-down-float])

have card-eq:  $\bigwedge \omega. \text{card } (f0\text{-sketch } p \ r \ t \ \omega \ as) = \text{card } (h \ \omega)$ 
  apply (subst real-g-2[symmetric])
  apply (rule card-image[symmetric])
  using inj-on-def real-of-float-inject by blast

have real-g:  $\bigwedge \omega. \text{real-of-rat } (g \ (f0\text{-sketch } p \ r \ t \ \omega \ as)) = g' \ (h \ \omega)$ 
  apply (simp add:g-def g'-def card-eq of-rat-divide of-rat-mult of-rat-add real-of-rat-of-float)
  apply (rule impI)
  apply (subst mono-Max-commute[where f=real-of-float])
  using less-eq-float.rep-eq mono-def apply blast
  apply (simp add:f0-sketch-def, simp add:least-def)
  using card-eq[symmetric] card-gt-0-iff t-ge-0 apply (simp, force)
  by (simp add:real-g-2)

have  $1 - \text{real-of-rat } \varepsilon \leq \mathcal{P}(\omega \text{ in measure-pmf } \Omega_0.$ 
  |median ( $\lambda i. g' \ (h \ (\omega \ i))$ ) s - real-of-rat (F 0 as)|  $\leq \text{real-of-rat } \delta * \text{real-of-rat}$ 
  (F 0 as))
  apply (rule prob-space.median-bound-2, simp add:prob-space-measure-pmf)
  using assms apply simp
  apply (subst  $\Omega_0$ -def)
  apply (rule indep-vars-restrict-intro [where f= $\lambda j. \{j\}$ ], simp, simp add:disjoint-family-on-def,
  simp add: s-ge-0, simp, simp, simp)
  apply (simp add:s-def) using of-nat-ceiling apply blast
  apply simp
  apply (subst  $\Omega_0$ -def)
  apply (subst prob-prod-pmf-slice, simp, simp)
  using b by (simp add: $\Omega_1$ -def)
also have ... =  $\mathcal{P}(\omega \text{ in measure-pmf } \Omega_0.$ 
  |median ( $\lambda i. g \ (f0\text{-sketch } p \ r \ t \ (\omega \ i) \ as)$ ) s - F 0 as|  $\leq \delta * F \ 0 \ as)$ 
  apply (rule arg-cong2[where f=measure], simp)
  apply (rule Collect-cong, simp, subst real-g[symmetric])

```

```

    apply (subst of-rat-mult[symmetric], subst median-rat[OF s-ge-0, symmetric])
    apply (subst of-rat-diff[symmetric], simp)
    using of-rat-less-eq by blast
  finally have a:P( $\omega$  in measure-pmf  $\Omega_0$ .
    |median ( $\lambda i. g (f0\text{-sketch } p \ r \ t \ (\omega \ i) \ as)) \ s - F \ 0 \ as| \leq \delta * F \ 0 \ as) \geq
1 - \text{real-of-rat } \varepsilon
    by blast

show ?thesis
  apply (subst M-def)
  apply (subst f0-alg-sketch[OF assms(1) assms(2) assms(3)], simp)
  apply (simp add:t-def[symmetric] p-def[symmetric] r-def[symmetric] s-def[symmetric]
map-pmf-def)
  apply (subst bind-assoc-pmf)
  apply (subst bind-return-pmf)
  apply (subst f0-result-elim)
  apply (subst map-pmf-def[symmetric])
  using a by (simp add: $\Omega_0$ -def[symmetric])
qed

fun f0-space-usage :: (nat  $\times$  rat  $\times$  rat)  $\Rightarrow$  real where
  f0-space-usage (n,  $\varepsilon$ ,  $\delta$ ) = (
    let s = nat  $\lceil -18 * \ln (\text{real-of-rat } \varepsilon) \rceil$  in
    let r = nat  $(4 * \lceil \log 2 (1 / \text{real-of-rat } \delta) \rceil + 24)$  in
    let t = nat  $\lceil 80 / (\text{real-of-rat } \delta)^2 \rceil$  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 where
  encode-state =
     $N_S \times_D (\lambda s.$ 
     $N_S \times_S ($ 
     $N_S \times_D (\lambda p.$ 
     $N_S \times_S ($ 
     $([0..<s] \rightarrow_S (\text{list}_S (zfact_S \ p)))) \times_S$ 
     $([0..<s] \rightarrow_S (\text{set}_S \ F_S))))))$ 

lemma inj-on encode-state (dom encode-state)
  apply (rule encoding-imp-inj)
  apply (simp add: encode-state-def)
  apply (rule dependent-encoding, metis nat-encoding)
  apply (rule prod-encoding, metis nat-encoding)
  apply (rule dependent-encoding, metis nat-encoding)
  apply (rule prod-encoding, metis nat-encoding)$ 
```

```

apply (rule prod-encoding, metis encode-extensional list-encoding zfact-encoding)
by (rule encode-extensional, rule encode-set, rule encode-float)

lemma f-subset:
  assumes  $g \text{ ' } A \subseteq h \text{ ' } B$ 
  shows  $(\lambda x. f (g x)) \text{ ' } A \subseteq (\lambda x. f (h x)) \text{ ' } B$ 
  using assms by auto

theorem f0-exact-space-usage:
  assumes  $\varepsilon \in \{0 < .. < 1\}$ 
  assumes  $\delta \in \{0 < .. < 1\}$ 
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  defines  $M \equiv \text{fold } (\lambda a \text{ state. state } \gg= \text{f0-update } a) \text{ as } (\text{f0-init } \delta \varepsilon n)$ 
  shows  $AE \ \omega \text{ in } M. \text{bit-count } (\text{encode-state } \omega) \leq \text{f0-space-usage } (n, \varepsilon, \delta)$ 
proof –
  define  $s$  where  $s = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 
  define  $t$  where  $t = \text{nat } \lceil 80 / (\text{real-of-rat } \delta)^2 \rceil$ 
  define  $p$  where  $p = \text{find-prime-above } (\text{max } n \ 19)$ 
  define  $r$  where  $r = \text{nat } (4 * \lceil \log 2 (1 / \text{real-of-rat } \delta) \rceil + 24)$ 

  have  $n\text{-le-}p$ :  $n \leq p$ 
    apply (rule order-trans[where  $y = \text{max } n \ 19$ ], simp)
    apply (subst  $p\text{-def}$ )
    by (rule find-prime-above-lower-bound)

  have  $p\text{-ge-}0$ :  $p > 0$ 
    apply (rule prime-gt-0-nat)
    by (simp add:  $p\text{-def}$  find-prime-above-is-prime)

  have  $p\text{-le-}n$ :  $p \leq 2 * n + 19$ 
    apply (simp add:  $p\text{-def}$ )
    apply (cases  $n \leq 19$ , simp add: find-prime-above.simps)
    apply (rule order-trans[where  $y = 2 * n + 2$ ], simp add: find-prime-above-upper-bound[simplified])
    by simp

  have  $\log\text{-}2\text{-}4$ :  $\log 2 \ 4 = 2$ 
    by (metis log2-of-power-eq mult-2 numeral-Bit0 of-nat-numeral power2-eq-square)

  have  $b\text{-}4\text{-}22$ :  $\bigwedge y. y \in \{0 .. < p\} \implies \text{bit-count } (F_S (\text{float-of } (\text{truncate-down } r \ y)))$ 
   $\leq$ 
     $\text{ereal } (10 + 4 * \text{real } r + 2 * \log 2 (\log 2 (n + 9)))$ 
  proof –
    fix  $y$ 
    assume  $a: y \in \{0 .. < p\}$ 

    show  $\text{bit-count } (F_S (\text{float-of } (\text{truncate-down } r \ y))) \leq \text{ereal } (10 + 4 * \text{real } r$ 
   $+ 2 * \log 2 (\log 2 (n + 9)))$ 
    proof (cases  $y \geq 1$ )
      case True

```

```

have b-4-23: 0 < 2 + log 2 (real p)
  apply (rule order-less-le-trans[where y=2+log 2 1], simp)
  using p-ge-0 by simp

have bit-count (FS (float-of (truncate-down r y))) ≤ ereal (8 + 4 * real r
+ 2 * log 2 (2 + |log 2 |real y|))
  by (rule truncate-float-bit-count)
also have ... ≤ ereal (8 + 4 * real r + 2 * log 2 (2 + log 2 p))
  apply (simp)
  apply (subst log-le-cancel-iff, simp, simp, simp add:b-4-23)
  apply (subst abs-of-nonneg) using True apply simp
  apply (simp, subst log-le-cancel-iff, simp, simp) using True apply simp
  apply (simp add:p-ge-0)
  using a by simp
also have ... ≤ ereal (8 + 4 * real r + 2 * log 2 (log 2 4 + log 2 (2 * n +
19)))
  apply simp
  apply (subst log-le-cancel-iff, simp, simp add:-b-4-23)
  apply (rule add-pos-pos, simp, simp)
  apply (rule add-mono)
  apply (metis dual-order.refl log2-of-power-eq mult-2 numeral-Bit0 of-nat-numeral
power2-eq-square)
  apply (subst log-le-cancel-iff, simp, simp add:p-ge-0, simp)
  using p-le-n by simp
also have ... ≤ ereal (8 + 4 * real r + 2 * log 2 (log 2 ((n+9) powr 2)))
  apply simp
  apply (subst log-le-cancel-iff, simp, rule add-pos-pos, simp, simp, simp)
  apply (subst log-mult[symmetric], simp, simp, simp, simp)
  by (subst log-le-cancel-iff, simp, simp, simp, simp add:power2-eq-square
algebra-simps)
also have ... = ereal (10 + 4 * real r + 2 * log 2 (log 2 (n + 9)))
  apply (subst log-powr, simp)
  apply (simp)
  apply (subst (3) log-2-4[symmetric])
  by (subst log-mult, simp, simp, simp, simp, simp add:log-2-4)
finally show ?thesis by simp
next
case False
hence y = 0 using a by simp
then show ?thesis by (simp add:float-bit-count-zero)
qed
qed

have b:
   $\bigwedge x. x \in (\{0..<s\} \rightarrow_E \text{bounded-degree-polynomials } (Z\text{Fact } (\text{int } p)) \ 2) \implies$ 
     $\text{bit-count } (\text{encode-state } (s, t, p, r, x, \lambda i \in \{0..<s\}. \text{f0-sketch } p \ r \ t \ (x \ i) \ as))$ 
 $\leq$ 
     $\text{f0-space-usage } (n, \varepsilon, \delta)$ 

```

```

proof –
  fix  $x$ 
  assume  $b-1: x \in \{0..<s\} \rightarrow_E \text{bounded-degree-polynomials } (ZFact \text{ (int } p)) \text{ } 2$ 
  have  $b-2: x \in \text{extensional } \{0..<s\}$  using  $b-1$  by (simp add: PiE-def)

  have  $\bigwedge y. y \in \{0..<s\} \implies \text{card } (f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } y) \text{ } as) \leq t$ 
    apply (simp add: f0-sketch-def)
    apply (subst card-least, simp)
    by simp

  hence  $b-3: \bigwedge y. y \in (\lambda z. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } z) \text{ } as) \text{ } ' \{0..<s\} \implies \text{card } y \leq t$ 
    by force

  have  $\bigwedge y. y \in \{0..<s\} \implies f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } y) \text{ } as \subseteq (\lambda k. \text{float-of } (truncate\text{-down } r \text{ } k)) \text{ } ' \{0..<p\}$ 
    apply (simp add: f0-sketch-def)
    apply (rule order-trans[OF least-subset])
    apply (rule f-subset[where f= $\lambda x. \text{float-of } (truncate\text{-down } r \text{ } (real \text{ } x))$ ])
    apply (rule image-subsetI, simp)
    apply (rule hash-range[OF p-ge-0, where n=2])
    using  $b-1$  apply (simp add: PiE-iff)
    using assms(3) n-le-p order-less-le-trans by blast

  hence  $b-4: \bigwedge y. y \in (\lambda z. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } z) \text{ } as) \text{ } ' \{0..<s\} \implies$ 
     $y \subseteq (\lambda k. \text{float-of } (truncate\text{-down } r \text{ } k)) \text{ } ' \{0..<p\}$ 
    by force

  have  $b-4-1: \bigwedge y \text{ } z. y \in (\lambda z. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } z) \text{ } as) \text{ } ' \{0..<s\} \implies z \in y \implies$ 
     $\text{bit-count } (F_S \text{ } z) \leq \text{ereal } (10 + 4 * \text{real } r + 2 * \log 2 (\log 2 (n+9)))$ 
    using  $b-4-22 \text{ } b-4$  by blast

  have  $\bigwedge y. y \in \{0..<s\} \implies \text{finite } (f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } y) \text{ } as)$ 
    apply (simp add: f0-sketch-def)
    by (rule finite-subset[OF least-subset], simp)

  hence  $b-5: \bigwedge y. y \in (\lambda z. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } z) \text{ } as) \text{ } ' \{0..<s\} \implies \text{finite } y$  by force

  have  $\text{bit-count } (\text{encode-state } (s, t, p, r, x, \lambda i \in \{0..<s\}. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } i) \text{ } as))$ 
    =
     $\text{bit-count } (N_S \text{ } s) + \text{bit-count } (N_S \text{ } t) + \text{bit-count } (N_S \text{ } p) + \text{bit-count } (N_S \text{ } r)$ 
    +
     $\text{bit-count } (\text{list}_S (\text{list}_S (\text{zfact}_S \text{ } p)) (\text{map } x \text{ } [0..<s])) +$ 
     $\text{bit-count } (\text{list}_S (\text{sets } F_S) (\text{map } (\lambda i \in \{0..<s\}. f0\text{-sketch } p \text{ } r \text{ } t \text{ } (x \text{ } i) \text{ } as) [0..<s]))$ 
    apply (simp add: b-2 encode-state-def dependent-bit-count prod-bit-count
      s-def[symmetric] t-def[symmetric] p-def[symmetric] r-def[symmetric] en-
code-extensional-def
      del: N_S.simps encode-prod.simps encode-dependent-sum.simps)
    by (simp add: ac-simps del: N_S.simps encode-prod.simps encode-dependent-sum.simps)
    also have  $\dots \leq \text{ereal } (2 * \log 2 (\text{real } s + 1) + 1) + \text{ereal } (2 * \log 2 (\text{real } t +$ 
     $1) + 1)$ 
     $+ \text{ereal } (2 * \log 2 (\text{real } p + 1) + 1) + \text{ereal } (2 * \log 2 (\text{real } r + 1) + 1)$ 

```



```

      + (ereal (real s) * (ereal (real 2 * (2 * log 2 (real p) + 2) + 1) + 1) + 1)
      + (ereal (real s) * ((ereal (real t) * (ereal (10 + 4 * real r + 2 * log 2 (log 2
(real (n + 9))))
      + 1) + 1) + 1) + 1)
    apply (rule add-mono, rule add-mono, rule add-mono, rule add-mono, rule
add-mono)
      apply (metis nat-bit-count)
      apply (metis nat-bit-count)
      apply (metis nat-bit-count)
      apply (metis nat-bit-count)
      apply (rule list-bit-count-est[where xs=map x [0..<s], simplified])
    apply (rule bounded-degree-polynomial-bit-count[OF p-ge-0]) using b-1 apply
blast
      apply (rule list-bit-count-est[where xs=map (λi∈{0..<s}. f0-sketch p r t (x
i) as) [0..<s], simplified])
      apply (rule set-bit-count-est, metis b-5, metis b-3)
      apply simp
      by (metis b-4-1)
    also have ... = ereal ( 6 + 2 * log 2 (real s + 1) + 2 * log 2 (real t + 1) +
      2 * log 2 (real p + 1) + 2 * log 2 (real r + 1) + real s * (8 + 4 * log 2
(real p) +
      real t * (11 + (4 * real r + 2 * log 2 (log 2 (real n + 9))))))
      apply (simp)
      by (subst distrib-left[symmetric], simp)
    also have ... ≤ ereal ( 6 + 2 * log 2 (real s + 1) + 2 * log 2 (real t + 1) +
      2 * log 2 (2 * (10 + real n)) + 2 * log 2 (real r + 1) + real s * (8 + 4 *
log 2 (2 * (10 + real n)) +
      real t * (11 + (4 * real r + 2 * log 2 (log 2 (real n + 9))))))
      apply (simp, rule add-mono, simp) using p-le-n apply simp
      apply (rule mult-left-mono, simp)
      apply (subst log-le-cancel-iff, simp, simp add:p-ge-0, simp)
      using p-le-n apply simp
      by simp
    also have ... ≤ f0-space-usage (n, ε, δ)
      apply (subst log-mult, simp, simp, simp)
      apply (subst log-mult, simp, simp, simp)
      apply (simp add:s-def[symmetric] r-def[symmetric] t-def[symmetric])
      by (simp add:algebra-simps)
    finally show bit-count (encode-state (s, t, p, r, x, λi∈{0..<s}. f0-sketch p r t
(x i) as)) ≤
      f0-space-usage (n, ε, δ) by simp
  qed

  have a: ∧y. y ∈ (λx. (s, t, p, r, x, λi∈{0..<s}. f0-sketch p r t (x i) as)) ‘
    ({0..<s} →E bounded-degree-polynomials (ZFact (int p)) 2) ⇒
    bit-count (encode-state y) ≤ f0-space-usage (n, ε, δ)
    using b apply (simp add:image-def del:f0-space-usage.simps) by blast

  show ?thesis

```

```

apply (subst AE-measure-pmf-iff, rule ballI)
apply (subst (asm) M-def)
apply (subst (asm) f0-alg-sketch[OF assms(1) assms(2) assms(3)], simp)
apply (simp add:s-def[symmetric] t-def[symmetric] p-def[symmetric] r-def[symmetric])
apply (subst (asm) set-prod-pmf, simp)
apply (simp add:comp-def)
apply (subst (asm) set-pmf-of-set)
  apply (metis ne-bounded-degree-polynomials)
  apply (metis fin-bounded-degree-polynomials[OF p-ge-0])
using a
by (simp add:s-def[symmetric] t-def[symmetric] p-def[symmetric] r-def[symmetric])
qed

```

lemma f0-asymptotic-space-complexity:

```

f0-space-usage ∈ O[at-top ×F at-right 0 ×F at-right 0](λ(n, ε, δ). ln (1 / of-rat
ε) *
(ln (real n) + 1 / (of-rat δ)2 * (ln (ln (real n)) + ln (1 / of-rat δ))))
(is - ∈ O[?F](?rhs))

```

proof –

```

define n-of :: nat × rat × rat ⇒ nat where n-of = (λ(n, ε, δ). n)
define ε-of :: nat × rat × rat ⇒ rat where ε-of = (λ(n, ε, δ). ε)
define δ-of :: nat × rat × rat ⇒ rat where δ-of = (λ(n, ε, δ). δ)

```

```

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

```

```

have n-inf: ⋀c. eventually (λx. c ≤ (real (n-of x))) ?F
apply (simp add:n-of-def case-prod-beta')
apply (subst eventually-prod1', simp add:prod-filter-eq-bot)
by (meson eventually-at-top-linorder nat-ceiling-le-eq)

```

```

have delta-inf: ⋀c. eventually (λx. c ≤ 1 / (real-of-rat (δ-of x))) ?F
apply (simp add:δ-of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
by (rule inv-at-right-0-inf)

```

```

have eps-inf: ⋀c. eventually (λx. c ≤ 1 / (real-of-rat (ε-of x))) ?F
apply (simp add:ε-of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod1', simp)
by (rule inv-at-right-0-inf)

```

```

have zero-less-eps: eventually (λx. 0 < (real-of-rat (ε-of x))) ?F
apply (simp add:ε-of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod1', simp)
by (rule eventually-at-rightI[where b=1], simp, simp)

```

```

have zero-less-delta: eventually ( $\lambda x. 0 < (\text{real-of-rat } (\delta\text{-of } x))$ ) ?F
  apply (simp add:  $\delta\text{-of-def}$  case-prod-beta')
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod2', simp)
  by (rule eventually-at-rightI[where  $b=1$ ], simp, simp)

have l1:  $\forall_F x \text{ in } ?F. 0 \leq (\ln (\ln (\text{real } (n\text{-of } x))) + \ln (1 / \text{real-of-rat } (\delta\text{-of } x)))$ 
/  $(\text{real-of-rat } (\delta\text{-of } x))^2$ 
  apply (rule eventually-nonneg-div)
  apply (rule eventually-nonneg-add)
  apply (rule eventually-ln-ge-iff, rule eventually-ln-ge-iff[OF n-inf])
  apply (rule eventually-ln-ge-iff[OF delta-inf])
  by (rule eventually-mono[OF zero-less-delta], simp)

have unit-1:  $(\lambda-. 1) \in O[?F](\lambda x. 1 / (\text{real-of-rat } (\delta\text{-of } x))^2)$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF delta-inf[where  $c=1$ ]
zero-less-delta]])
  by (metis one-le-power power-one-over)

have unit-2:  $(\lambda-. 1) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\delta\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF delta-inf[where  $c=\exp 1$ ]
zero-less-delta]])
  apply (subst abs-of-nonneg)
  apply (rule ln-ge-zero)
  apply (meson dual-order.trans one-le-exp-iff rel-simps(44))
  by (simp add: ln-ge-iff)

have unit-3:  $(\lambda-. 1) \in O[?F](\lambda x. \text{real } (n\text{-of } x))$ 
  by (rule landau-o.big-mono, simp, rule n-inf)

have unit-4:  $(\lambda-. 1) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF eps-inf[where  $c=\exp 1$ ]
zero-less-eps]])
  apply (subst abs-of-nonneg)
  apply (rule ln-ge-zero)
  using one-le-exp-iff order-trans-rules(23) apply blast
  by (simp add: ln-ge-iff)

have unit-5:  $(\lambda-. 1) \in O[?F](\lambda x. 1 / \text{real-of-rat } (\varepsilon\text{-of } x))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF eps-inf[where  $c=1$ ] zero-less-eps]])
  by simp

have unit-6:  $(\lambda-. 1) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)

```

```

apply (rule eventually-mono[OF n-inf[where c=exp 1]])
apply (subst abs-of-nonneg)
apply (rule ln-ge-zero)
apply (metis less-one not-exp-le-zero not-le of-nat-eq-0-iff of-nat-ge-1-iff)
by (metis less-eq-real-def ln-ge-iff not-exp-le-zero of-nat-0-le-iff)

have unit-7:  $(\lambda-. 1) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)) + (\ln (\ln (\text{real } (n\text{-of } x))) + \ln (1 / \text{real-of-rat } (\delta\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$ 
apply (rule landau-sum-1)
apply (rule eventually-ln-ge-iff[OF n-inf])
apply (rule l1)
by (rule unit-6)

have unit-8:  $(\lambda-. 1) \in O[?F](g)$ 
apply (simp add:g-def)
apply (rule landau-o.big-mult-1[OF unit-4])
by (rule unit-7)

have l2:  $(\lambda x. \ln (\text{real } (\text{nat } \lceil - (18 * \ln (\text{real-of-rat } (\varepsilon\text{-of } x))) \rceil) + 1)) \in O[?F](g)$ 

apply (simp add:g-def)
apply (rule landau-o.big-mult-1)
apply (rule landau-ln-2[where a=2], simp, simp)
apply (rule eps-inf)
apply (rule sum-in-bigo)
apply (rule landau-nat-ceil[OF unit-5])
apply (subst minus-mult-right)
apply (subst cmult-in-bigo-iff, rule disjI2)
apply (subst landau-o.big.in-cong[where f= $\lambda x. - \ln (\text{real-of-rat } (\varepsilon\text{-of } x))$ ])
and g= $\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x))$ 
apply (rule eventually-mono[OF zero-less-eps], simp add:ln-div)
apply (rule landau-ln-3[OF eps-inf], simp, rule unit-5)
by (rule unit-7)

have l3:  $(\lambda x. \ln (\text{real } (\text{nat } \lceil 80 / (\text{real-of-rat } (\delta\text{-of } x))^2 \rceil) + 1)) \in O[?F](g)$ 
apply (simp add:g-def)
apply (rule landau-o.big-mult-1'[OF unit-4])
apply (rule landau-sum-2)
apply (rule eventually-ln-ge-iff[OF n-inf])
apply (rule l1)
apply (subst (3) div-commute)
apply (rule landau-o.big-mult-1)
apply (rule landau-ln-3, simp)
apply (rule sum-in-bigo)
apply (rule landau-nat-ceil[OF unit-1])
apply (rule landau-const-inv, simp, simp, rule unit-1)
apply (rule landau-sum-2)
apply (rule eventually-ln-ge-iff[OF eventually-ln-ge-iff[OF n-inf]])
apply (rule eventually-ln-ge-iff[OF delta-inf])

```

```

by (rule unit-2)

have unit-9:  $(\lambda x. 1) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF n-inf[where c=exp 1]])
  by (metis abs-ge-self less-eq-real-def ln-ge-iff not-exp-le-zero of-nat-0-le-iff or-
der.trans)

have l4:  $(\lambda x. \ln (10 + \text{real } (n\text{-of } x))) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)))$ 
  apply (rule landau-ln-2[where a=2], simp, simp, rule n-inf)
  by (rule sum-in-bigo, simp add:unit-3, simp)

have l5:  $(\lambda x. \ln (\text{real } (n\text{-of } x) + 10)) \in O[?F](g)$ 
  apply (simp add:g-def)
  apply (rule landau-o.big-mult-1'[OF unit-4])
  apply (rule landau-sum-1)
  apply (rule eventually-ln-ge-iff[OF n-inf])
  apply (rule l1)
  apply (rule landau-ln-2[where a=2], simp, simp, rule n-inf)
  by (rule sum-in-bigo, simp, simp add:unit-3)

have l6:  $(\lambda x. \log 2 (\text{real } (\text{nat } (4 * \lceil \log 2 (1 / \text{real-of-rat } (\delta\text{-of } x)) \rceil + 24)) + 1)) \in O[?F](g)$ 
  apply (simp add:g-def log-def, rule landau-o.big-mult-1'[OF unit-4], rule lan-
dau-sum-2)
  apply (rule eventually-ln-ge-iff[OF n-inf])
  apply (rule l1)
  apply (subst (4) div-commute)
  apply (rule landau-o.big-mult-1)
  apply (rule landau-ln-3, simp)
  apply (rule sum-in-bigo)
  apply (rule landau-real-nat, simp)
  apply (rule sum-in-bigo)
  apply (simp, rule landau-ceil[OF unit-1], simp, rule landau-ln-3[OF delta-inf])
  apply (rule landau-o.big-mono)
  apply (rule eventually-mono[OF eventually-conj[OF delta-inf[where c=1]
zero-less-delta]])
  apply (simp, metis pos2 power-one-over self-le-power)
  apply (simp add:unit-1)
  apply (simp add:unit-1)
  apply (rule landau-sum-2)
  apply (rule eventually-ln-ge-iff, rule eventually-ln-ge-iff[OF n-inf])
  apply (rule eventually-ln-ge-iff[OF delta-inf])
  by (rule unit-2)

have l7:  $(\lambda x. \text{real } (\text{nat } \lceil -(18 * \ln (\text{real-of-rat } (\varepsilon\text{-of } x))) \rceil)) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)))$ 
  apply (rule landau-nat-ceil, rule unit-4)
  apply (subst minus-mult-right)

```

```

apply (subst cmult-in-bigo-iff, rule disjI2)
apply (rule landau-o.big-mono)
apply (rule eventually-mono[OF zero-less-eps])
by (subst ln-div, simp, simp, simp)

have l8: ( $\lambda x. \text{real} (\text{nat} \lceil 80 / (\text{real-of-rat } (\delta\text{-of } x))^2 \rceil) * (11 + 4 * \text{real} (\text{nat} (4 * \lceil \log 2 (1 / \text{real-of-rat } (\delta\text{-of } x)) \rceil + 24)) + 2 * \log 2 (\log 2 (\text{real} (n\text{-of } x) + 9)))) \in O[?F](\lambda x. (\ln (\ln (\text{real} (n\text{-of } x))) + \ln (1 / \text{real-of-rat } (\delta\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$ )
apply (subst (4) div-commute)
apply (rule landau-o.mult)
apply (rule landau-nat-ceil[OF unit-1], rule landau-const-inv, simp, simp)
apply (subst (3) add-commute)
apply (rule landau-sum)
apply (rule eventually-ln-ge-iff, rule eventually-ln-ge-iff, rule n-inf)
apply (rule eventually-ln-ge-iff, rule delta-inf, simp add:log-def)
apply (rule landau-ln-2[where a=2], simp)
apply (subst pos-le-divide-eq, simp, simp)
apply (rule eventually-mono[OF n-inf[where c=exp 2]])
apply (subst ln-ge-iff, metis less-eq-real-def not-exp-le-zero of-nat-0-le-iff)
apply simp
apply (simp, rule landau-ln-2[where a=2], simp, simp, rule n-inf)
apply (rule sum-in-bigo, simp, simp add:unit-3)
apply (rule sum-in-bigo, simp add:unit-2)
apply (simp, rule landau-real-nat, simp)
apply (rule sum-in-bigo, simp)
by (rule landau-ceil[OF unit-2], simp add:log-def, simp add:unit-2)

have f0-space-usage = ( $\lambda x. \text{f0-space-usage} (n\text{-of } x, \varepsilon\text{-of } x, \delta\text{-of } x)$ )
apply (rule ext)
by (simp add:case-prod-beta' n-of-def  $\varepsilon$ -of-def  $\delta$ -of-def)
also have ...  $\in O[?F](g)$ 
apply (simp add:Let-def)
apply (rule sum-in-bigo-r)
apply (simp add:g-def)
apply (rule landau-o.mult, simp add:l7)
apply (rule landau-sum)
apply (rule eventually-ln-ge-iff[OF n-inf])
apply (rule l1)
apply (rule sum-in-bigo-r, simp add:log-def l4, simp add:unit-9)
apply (simp add:l8)
apply (rule sum-in-bigo-r, simp add:l6)
apply (rule sum-in-bigo-r, simp add:log-def l5)
apply (rule sum-in-bigo-r, simp add:log-def l3)
apply (rule sum-in-bigo-r, simp add:log-def l2)
by (simp add:unit-8)
also have ... =  $O[?F](?rhs)$ 
apply (rule arg-cong2[where f=bigo], simp)

```

```

    apply (rule ext)
    by (simp add: case-prod-beta' g-def n-of-def ε-of-def δ-of-def)
  finally show ?thesis
    by simp
qed

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 ⇒ (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 (*induction xs*)

```

  case Nil
  then show ?case by simp

```

next

```

  case (Cons a xs)
  then show ?case
    apply (cases p a)
    apply (simp, metis append.left-neutral filter.simps(1))
    by (simp, metis append-Cons filter.simps(2))
qed

```

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
  assume a:has-eq-relation (r#rs) (x#xs)
  have ∧ i j. i < length xs ⇒ j < length xs ⇒ (xs ! i = xs ! j) = (rs ! i = rs !
j)
    (is ∧ i j. ?l1 i ⇒ ?l2 j ⇒ ?rhs i j)
  proof -
    fix i j
    assume i < length xs
    hence Suc i < length (x#xs) by auto
    moreover assume j < length xs
    hence Suc j < length (x#xs) by auto
    ultimately show ?rhs i j using a apply (simp only:has-eq-relation-def)
      by (metis nth-Cons-Suc)
  qed
  hence has-eq-relation rs xs using a by (simp add:has-eq-relation-def)
  thus (∀ i < length xs. (r = rs ! i) = (x = xs ! i)) ∧ has-eq-relation rs xs
    apply simp
    using a apply (simp only:has-eq-relation-def)
    by (metis Suc-less-eq length-Cons nth-Cons-0 nth-Cons-Suc zero-less-Suc)
next
  assume a:(∀ i < length xs. (r = rs ! i) = (x = xs ! i)) ∧ has-eq-relation rs xs
  have ∧ i j. i < Suc (length rs) ⇒ j < Suc (length rs) ⇒ ((x # xs) ! i = (x #
xs) ! j) = ((r # rs) ! i = (r # rs) ! j)
    (is ∧ i j. ?l1 i ⇒ ?l2 j ⇒ ?rhs i j)
  proof -
    fix i j
    assume i < Suc (length rs)
    moreover assume j < Suc (length rs)
    ultimately show ?rhs i j using a
      apply (cases i, cases j)
      apply (simp add: has-eq-relation-def)
      apply (cases j)
      apply (simp add: has-eq-relation-def)+
      by (metis less-Suc-eq-0-disj nth-Cons' nth-Cons-Suc)
  qed
  then show has-eq-relation (r # rs) (x # xs)
    using a by (simp add:has-eq-relation-def)
qed

lemma enum-partitions-aux-range:
  x ∈ set (enum-partitions-aux n) ⇒ set (snd x) = {k. k < fst x}
by (induction n arbitrary:x, simp, simp, force)

lemma enum-partitions-aux-len:
  x ∈ set (enum-partitions-aux n) ⇒ length (snd x) = n
by (induction n arbitrary:x, simp, simp, force)

```



```

lemma enum-partitions-complete-aux:  $k < n \implies \text{length } (\text{filter } (\lambda x. x = k) [0..<n])$ 
= Suc 0
  by (induction n, simp, simp)

lemma enum-partitions-complete:
  length (filter ( $\lambda p. \text{has-eq-relation } p \ x$ ) (enum-partitions (length x))) = 1
proof (induction x)
  case Nil
  then show ?case by (simp add:has-eq-relation-def)
next
  case (Cons a y)
  have length (filter ( $\lambda x. \text{has-eq-relation } (\text{snd } x) \ y$ ) (enum-partitions-aux (length
y)))) = 1
    using Cons by (simp add:comp-def)
  then obtain p1 p2 p3 where pi-def: enum-partitions-aux (length y) = p1@p2#p3
and
  p2-t: has-eq-relation (snd p2) y and
  p1-f1: filter ( $\lambda x. \text{has-eq-relation } (\text{snd } x) \ y$ ) p1 = [] and
  p3-f1: filter ( $\lambda x. \text{has-eq-relation } (\text{snd } x) \ y$ ) p3 = []
    using Cons filter-one-elim by (metis (no-types, lifting) length-0-conv)
  have p2-e: p2 ∈ set(enum-partitions-aux (length y))
    using pi-def by auto
  have p1-f:  $\bigwedge x \ p. x \in \text{set } p1 \implies \text{has-eq-relation } (p\#(\text{snd } x)) \ (a\#y) = \text{False}$ 
    by (metis p1-f1 filter-empty-conv has-eq-elim)
  have p3-f:  $\bigwedge x \ p. x \in \text{set } p3 \implies \text{has-eq-relation } (p\#(\text{snd } x)) \ (a\#y) = \text{False}$ 
    by (metis p3-f1 filter-empty-conv has-eq-elim)
  show ?case
proof (cases a ∈ set y)
  case True
  then obtain h where h-def:  $h < \text{length } y \wedge a = y ! h$  by (metis in-set-conv-nth)
  define k where k = snd p2 ! h
  have k-bound:  $k < \text{fst } p2$ 
    using enum-partitions-aux-len enum-partitions-aux-range p2-e k-def h-def
    by (metis mem-Collect-eq nth-mem)
  have k-eq:  $\bigwedge i. \text{has-eq-relation } (i \# \text{snd } p2) \ (a \# y) = (i = k)$ 
    apply (simp add:has-eq-elim p2-t k-def)
    using h-def has-eq-relation-def p2-t by auto
  show ?thesis
    apply (simp add: filter-concat length-concat case-prod-beta' comp-def)
    apply (simp add: pi-def p1-f p3-f cong:map-cong)
    by (simp add: k-eq k-bound enum-partitions-complete-aux)
next
  case False
  hence has-eq-relation (fst p2 # snd p2) (a # y)
    apply (simp add:has-eq-elim p2-t)
    using enum-partitions-aux-range p2-e
    by (metis enum-partitions-aux-len mem-Collect-eq nat-neq-iff nth-mem)
  moreover have  $\bigwedge i. i < \text{fst } p2 \implies \neg(\text{has-eq-relation } (i \# \text{snd } p2) \ (a \# y))$ 

```

```

    apply (simp add:has-eq-elim p2-t)
  by (metis False enum-partitions-aux-range p2-e has-eq-relation-def in-set-conv-nth
mem-Collect-eq p2-t)
ultimately show ?thesis
  apply (simp add: filter-concat length-concat case-prod-beta' comp-def)
  by (simp add: pi-def p1-f p3-f cong:map-cong)
qed
qed

fun verify where
  verify r x 0 = True |
  verify r x (Suc n) 0 = verify r x n |
  verify r x (Suc n) (Suc m) = (((r ! n = r ! m) = (x ! n = x ! m)) ∧ (verify r x
(Suc n) m))

lemma verify-elim-1:
  verify r x (Suc n) m = (verify r x n n ∧ (∀ i < m. (r ! n = r ! i) = (x ! n = x
! i)))
  apply (induction m, simp, simp)
  using less-Suc-eq by auto

lemma verify-elim:
  verify r x m m = (∀ i < m. ∀ j < i. (r ! i = r ! j) = (x ! i = x ! j))
  apply (induction m, simp, simp add:verify-elim-1)
  apply (rule order-antisym, simp, metis less-antisym less-trans)
  apply (simp)
  using less-Suc-eq by presburger

lemma has-eq-relation-elim:
  has-eq-relation r xs = (length r = length xs ∧ verify r xs (length xs) (length xs))
  apply (simp add: has-eq-relation-def verify-elim)
  by (metis (mono-tags, lifting) less-trans nat-neq-iff)

lemma sum-filter: sum-list (map (λp. if f p then (r::real) else 0) y) = r*(length
(filter f y))
  by (induction y, simp, simp add:algebra-simps)

lemma sum-partitions: sum-list (map (λp. if has-eq-relation p x then (r::real) else
0) (enum-partitions (length x))) = r
  by (metis mult.right-neutral of-nat-1 enum-partitions-complete sum-filter)

lemma sum-partitions':
  assumes n = length x
  shows sum-list (map (λp. of-bool (has-eq-relation p x) * (r::real)) (enum-partitions
n)) = r
  apply (simp add:of-bool-def comp-def asms del:enum-partitions.simps)
  apply (subst (2) sum-partitions[where x=x and r=r, symmetric])
  apply (rule arg-cong[where f=sum-list])
  apply (rule map-cong, simp)

```

by *simp*

lemma *eq-rel-obtain-bij*:

assumes *has-eq-relation* *u v*

obtains *f* **where** *bij-betw* *f* (*set u*) (*set v*) $\bigwedge y. y \in \text{set } u \implies \text{count-list } u \ y = \text{count-list } v \ (f \ y)$

proof –

define *A* **where** $A = (\lambda x. \{k. k < \text{length } u \wedge u \ ! \ k = x\})$

define *q* **where** $q = (\lambda x. v \ ! \ (\text{Min } (A \ x)))$

have *A-ne-iff*: $\bigwedge x. x \in \text{set } u \implies A \ x \neq \{\}$ **by** (*simp add:A-def in-set-conv-nth*)

have *f-A*: $\bigwedge x. \text{finite } (A \ x)$ **by** (*simp add:A-def*)

have *a:inj-on* *q* (*set u*)

proof (*rule inj-onI*)

fix *x y*

assume *a-1*: $x \in \text{set } u \ y \in \text{set } u$

have *length u* > 0 **using** *a-1* **by** *force*

define *xi* **where** *xi* = *Min* (*A x*)

have *xi-l*: *xi* < *length u*

using *Min-in*[*OF f-A A-ne-iff*[*OF a-1*(1)]]

by (*simp add:xi-def A-def*)

have *xi-v*: $u \ ! \ xi = x$

using *Min-in*[*OF f-A A-ne-iff*[*OF a-1*(1)]]

by (*simp add:xi-def A-def*)

define *yi* **where** *yi* = *Min* (*A y*)

have *yi-l*: *yi* < *length u*

using *Min-in*[*OF f-A A-ne-iff*[*OF a-1*(2)]]

by (*simp add:yi-def A-def*)

have *yi-v*: $u \ ! \ yi = y$

using *Min-in*[*OF f-A A-ne-iff*[*OF a-1*(2)]]

by (*simp add:yi-def A-def*)

assume *q x* = *q y*

hence $v \ ! \ xi = v \ ! \ yi$

by (*simp add:q-def xi-def yi-def*)

hence $u \ ! \ xi = u \ ! \ yi$

by (*metis* (*no-types*, *lifting*) *has-eq-relation-def* *assms*(1) *xi-l yi-l*)

thus *x* = *y*

using *yi-v xi-v* **by** *blast*

qed

have *b*: $\bigwedge y. y \in \text{set } u \implies \text{count-list } u \ y = \text{count-list } v \ (q \ y)$

proof –

fix *y*

assume *b-1*: $y \in \text{set } u$

define *i* **where** *i* = *Min* (*A y*)

have *i-bound*: *i* < *length u*

```

    using Min-in[OF f-A A-ne-iff[OF b-1]]
    by (simp add:i-def A-def)
  have y-def:  $y = u ! i$ 
    using Min-in[OF f-A A-ne-iff[OF b-1]]
    by (simp add:i-def A-def)

  have count-list u y = card {k.  $k < \text{length } u \wedge u ! k = u ! i$ }
    by (simp add:count-list-card y-def)
  also have ... = card {k.  $k < \text{length } v \wedge v ! k = v ! i$ }
    apply (rule arg-cong[where f=card])
    apply (rule set-eqI, simp)
    by (metis (no-types, lifting) assms(1) has-eq-relation-def i-bound)
  also have ... = card {k.  $k < \text{length } v \wedge v ! k = q y$ }
    by (simp add:q-def i-def)
  also have ... = count-list v (q y)
    by (simp add:count-list-card)
  finally show count-list u y = count-list v (q y)
    by simp
qed

have c:q ' set u  $\subseteq$  set v
  apply (rule image-subsetI)
  by (metis b count-list-gr-1)

have d-1:length v = length u using assms has-eq-relation-def by blast
also have ... = sum (count-list u) (set u)
  by (simp add:sum-count-set)
also have ... = sum ((count-list v)  $\circ$  q) (set u)
  by (rule sum.cong, simp, simp add:comp-def b)
also have ... = sum (count-list v) (q ' set u)
  by (rule sum.reindex[OF a, symmetric])
finally have d-1:sum (count-list v) (q ' set u) = length v
  by simp

have sum (count-list v) (q ' set u) + sum (count-list v) (set v - (q ' set u)) =
sum (count-list v) (set v)
  apply (subst sum.union-disjoint[symmetric], simp, simp, simp)
  apply (rule sum.cong)
  using c apply blast
  by simp
also have ... = length v
  by (simp add:sum-count-set)
finally have d-2:sum (count-list v) (q ' set u) + sum (count-list v) (set v - (q
' set u)) = length v by simp

have sum (count-list v) (set v - (q ' set u)) = 0
  using d-1 d-2 by linarith

hence  $\bigwedge x. x \in (\text{set } v - (q ' \text{set } u)) \implies \text{count-list } v \ x \leq 0$ 

```

```

    using member-le-sum by simp
  hence  $\bigwedge x. x \in (\text{set } v - (q \text{ ' set } u)) \implies \text{False}$ 
    by (metis count-list-gr-1 Diff-iff le-0-eq not-one-le-zero)
  hence  $\text{set } v - (q \text{ ' set } u) = \{\}$ 
    by blast

  hence  $e: q \text{ ' set } u = \text{set } v$ 
    using c by blast

  have  $d: \text{bij-betw } q (\text{set } u) (\text{set } v)$ 
    apply (simp add: bij-betw-def)
    using c e a by blast
  have  $\exists f. \text{bij-betw } f (\text{set } u) (\text{set } v) \wedge (\forall y \in \text{set } u. \text{count-list } u \ y = \text{count-list } v \ (f \ y))$ 
    using b d by blast
  with that show ?thesis by blast
qed

end

```

18 Frequency Moment 2

theory *Frequency-Moment-2*

imports *Main Median Partitions Primes-Ext Encoding List-Ext*

UniversalHashFamilyOfPrime 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 $p \ h \ k = (\text{if } \text{hash } p \ k \ h \in \{k. 2*k < p\} \text{ then } \text{int } p - 1 \text{ else } - \text{int } p - 1)$

type-synonym *f2-state* = $\text{nat} \times \text{nat} \times \text{nat} \times (\text{nat} \times \text{nat} \Rightarrow \text{int set list}) \times (\text{nat} \times \text{nat} \Rightarrow \text{int})$

fun *f2-init* :: $\text{rat} \Rightarrow \text{rat} \Rightarrow \text{nat} \Rightarrow \text{f2-state pmf}$ **where**

```

  f2-init  $\delta \ \varepsilon \ n =$ 
    do {
      let  $s_1 = \text{nat } \lceil 6 / \delta^2 \rceil$ ;
      let  $s_2 = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ ;
      let  $p = \text{find-prime-above } (\max \ n \ 3)$ ;
       $h \leftarrow \text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) (\lambda-. \text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ 4))$ ;
      return-pmf  $(s_1, s_2, p, h, (\lambda-. \in \{0..<s_1\} \times \{0..<s_2\}. (0 :: \text{int})))$ 
    }

```

```

fun f2-update :: nat  $\Rightarrow$  f2-state  $\Rightarrow$  f2-state pmf where
  f2-update x (s1, s2, p, h, sketch) =
    return-pmf (s1, s2, p, h,  $\lambda i \in \{0..<s_1\} \times \{0..<s_2\}. f2\text{-hash } p (h\ i) x + sketch$ 
i)

fun f2-result :: f2-state  $\Rightarrow$  rat pmf where
  f2-result (s1, s2, p, h, sketch) =
    return-pmf (median ( $\lambda i_2 \in \{0..<s_2\}.
      (\sum_{i_1 \in \{0..<s_1\}} . (rat\text{-of-int } (sketch\ (i_1, i_2)))^2) / (((rat\text{-of-nat } p)^2 - 1) *
rat\text{-of-nat } s_1))\ s_2$ 
)

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))
    ( $\lambda \omega. real\text{-of-int } (f2\text{-hash } p\ \omega\ k) \wedge m =
      (((real\ p - 1) \wedge m * (real\ p + 1) + (-\ real\ p - 1) \wedge m * (real\ p - 1)) / (2
* real\ p))$ )
proof -
  have g:p > 0 using assms(1) prime-gt-0-nat by auto

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

  define  $\Omega$  where  $\Omega = pmf\text{-of-set } (bounded\text{-degree-polynomials } (ZFact\ (int\ p))\ 4)$ 

  have b: finite (set-pmf  $\Omega$ )
  apply (simp add: $\Omega$ -def)
  by (metis fin-bounded-degree-polynomials[OF g] ne-bounded-degree-polynomials
set-pmf-of-set)

  have zero-le-4: 0 < (4::nat) by simp

  have card ( $\{k. 2 * k < p\} \cap \{0..<p\}$ ) = card ( $\{0..t\}$ )
  apply (subst Int-absorb2, rule subsetI, simp)
  apply (rule arg-cong[where f=card])
  apply (rule order-antisym, rule subsetI, simp add:t-def)
  by (rule subsetI, simp add:t-def)
  also have ... = t+1
  by simp
  also have ... = (real p + 1)/2
  by (simp add:t-def)
  finally have c-1: card ( $\{k. 2 * k < p\} \cap \{0..<p\}$ ) = (real p+1)/2 by simp

```

```

have card ( $\{k. p \leq 2 * k\} \cap \{0..<p\}$ ) = card  $\{t+1..<p\}$ 
  apply (rule arg-cong[where f=card])
  apply (rule order-antisym, rule subsetI, simp add:t-def)
  by (rule subsetI, simp add:t-def)
also have ... =  $p - (t+1)$  by simp
also have ... =  $(\text{real } p - 1)/2$ 
  by (simp add:t-def)
finally have c-2: card ( $\{k. p \leq 2 * k\} \cap \{0..<p\}$ ) =  $(\text{real } p - 1)/2$  by simp

have integralL  $\Omega (\lambda x. \text{real-of-int } (f2\text{-hash } p \ x \ k) \wedge^m) =$ 
  integralL  $\Omega (\lambda \omega. \text{indicator } \{\omega. 2 * \text{hash } p \ k \ \omega < p\} \ \omega * (\text{real } p - 1) \wedge^m +$ 
    indicator  $\{\omega. 2 * \text{hash } p \ k \ \omega \geq p\} \ \omega * (-\text{real } p - 1) \wedge^m)$ 
  by (rule Bochner-Integration.integral-cong, simp, simp)
also have ... =
   $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. \text{hash } p \ k \ \omega \in \{k. 2 * k < p\}) * (\text{real } p - 1) \wedge^m +$ 
   $\mathcal{P}(\omega \text{ in measure-pmf } \Omega. \text{hash } p \ k \ \omega \in \{k. 2 * k \geq p\}) * (-\text{real } p - 1) \wedge^m$ 
  apply (subst Bochner-Integration.integral-add)
  apply (rule integrable-measure-pmf-finite[OF b])
  apply (rule integrable-measure-pmf-finite[OF b])
  by simp
also have ... =  $(\text{real } p + 1) * (\text{real } p - 1) \wedge^m / (2 * \text{real } p) + (\text{real } p - 1) *$ 
 $(-\text{real } p - 1) \wedge^m / (2 * \text{real } p)$ 
  apply (simp only: $\Omega\text{-def}$  hash-prob-range[OF assms(1) assms(2) zero-le-4] c-1
c-2)
  by simp
also have ... =
   $((\text{real } p - 1) \wedge^m * (\text{real } p + 1) + (-\text{real } p - 1) \wedge^m * (\text{real } p - 1)) / (2 * \text{real } p)$ 
  by (simp add:add-divide-distrib ac-simps)
finally have a:integralL  $\Omega (\lambda x. \text{real-of-int } (f2\text{-hash } p \ x \ k) \wedge^m) =$ 
 $((\text{real } p - 1) \wedge^m * (\text{real } p + 1) + (-\text{real } p - 1) \wedge^m * (\text{real } p - 1)) / (2 * \text{real } p)$ 
  by simp

show ?thesis
  apply (subst  $\Omega\text{-def}$ [symmetric])
  by (metis a)
qed

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

```

```

proof –
  define h where h = ( $\lambda \omega \ x. \text{real-of-int } (\text{f2-hash } p \ \omega \ x)$ )
  define c where c = ( $\lambda x. \text{real } (\text{count-list } as \ x)$ )
  define r where r = ( $\lambda (m::nat). ((\text{real } p - 1) ^ m * (\text{real } p + 1) + (- \text{real } p - 1) ^ m * (\text{real } p - 1)) / (2 * \text{real } p)$ )
  define h-prod where h-prod = ( $\lambda as \ \omega. \text{prod-list } (\text{map } (h \ \omega) \ as)$ )

  define exp-h-prod :: nat list  $\Rightarrow$  real where exp-h-prod = ( $\lambda as. (\prod i \in \text{set } as. r \ (\text{count-list } as \ i))$ )

  interpret prob-space M
    using prob-space-measure-pmf M-def by auto

  have f-eq: f = ( $\lambda \omega. (\sum x \in \text{set } as. c \ x * h \ \omega \ x) ^ 2$ )
    by (simp add:f-def c-def h-def sum-list-eval del:f2-hash.simps)

  have p-ge-0: p > 0 using assms(2) by simp

  have int-M:  $\bigwedge f. \text{integrable } M \ (\lambda \omega. ((f \ \omega)::\text{real}))$ 
    apply (simp add:M-def)
    apply (rule integrable-measure-pmf-finite)
    by (metis p-ge-0 set-pmf-of-set ne-bounded-degree-polynomials fin-bounded-degree-polynomials)

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

  have r-two: r 2 = ( $\text{real } p ^ 2 - 1$ )
    apply (simp add:r-def)
    apply (subst nonzero-divide-eq-eq) using assms apply simp
    by (simp add:algebra-simps power2-eq-square)

  have r-four-est: r 4  $\leq 3 * r \ 2 * r \ 2$ 
    apply (simp add:r-two)
    apply (simp add:r-def)
    apply (subst pos-divide-le-eq) using assms apply simp
    apply (simp add:algebra-simps power2-eq-square power4-eq-xxxx)
    apply (rule order-trans[where y= $\text{real } p * 12 + \text{real } p * (\text{real } p * (\text{real } p * (\text{real } p * 16)))$ ]])
    apply simp
    apply (rule add-mono, simp)
    apply (rule mult-left-mono)
    apply (rule mult-left-mono)
    apply (rule mult-left-mono)
    apply simp
    using assms(2)
    apply (metis assms(1) linorder-not-less num-double numeral-mult of-nat-power power2-eq-square power2-nat-le-eq-le prime-ge-2-nat real-of-nat-less-numeral-iff)
    by simp

```



```

have fold-sym:  $\bigwedge x y. (x \neq y \wedge y \neq x) = (x \neq y)$  by auto

have exp-h-prod-elim:  $\text{exp-h-prod} = (\lambda as. \text{prod-list } (\text{map } (r \circ \text{count-list } as) (\text{remdups } as)))$ 
  apply (simp add:exp-h-prod-def)
  apply (rule ext)
  apply (subst prod.set-conv-list[symmetric])
  by (rule prod.cong, simp, simp add:comp-def)

have exp-h-prod:  $\bigwedge x. \text{set } x \subseteq \text{set } as \implies \text{length } x \leq 4 \implies \text{expectation } (h\text{-prod } x) = \text{exp-h-prod } x$ 
proof -
  fix x
  assume set x  $\subseteq$  set as
  hence x-sub-p:  $\text{set } x \subseteq \{0..<p\}$  using assms(3) atLeastLessThan-iff by blast
  hence x-le-p:  $\bigwedge k. k \in \text{set } x \implies k < p$  by auto
  assume length x  $\leq$  4
  hence card-x:  $\text{card } (\text{set } x) \leq 4$  using card-length dual-order.trans by blast

  have expectation (h-prod x) = expectation ( $\lambda \omega. \prod i \in \text{set } x. h \ \omega \ i \sim (\text{count-list } x \ i)$ )
  apply (rule arg-cong[where f=expectation])
  by (simp add:h-prod-def prod-list-eval)
  also have ... = ( $\prod i \in \text{set } x. \text{expectation } (\lambda \omega. h \ \omega \ i \sim (\text{count-list } x \ i))$ )
  apply (subst indep-vars-lebesgue-integral, simp)
  apply (simp add:h-def)
  apply (rule indep-vars-compose2[where X=hash p and M'=( $\lambda \cdot. \text{pmf-of-set } \{0..<p\}$ )])
  using hash-k-wise-indep[where n=4 and p=p] card-x x-sub-p assms(1)
  apply (simp add:k-wise-indep-vars-def M-def[symmetric])
  apply simp
  apply (rule int-M)
  by simp
  also have ... = ( $\prod i \in \text{set } x. r \ (\text{count-list } x \ i)$ )
  apply (rule prod.cong, simp)
  using f2-hash-exp[OF assms(1) x-le-p assms(2)]
  by (simp add:h-def r-def M-def[symmetric] del:f2-hash.simps)
  also have ... = exp-h-prod x
  by (simp add:exp-h-prod-def)
  finally show expectation (h-prod x) = exp-h-prod x by simp
qed

have exp-h-prod-cong:  $\bigwedge x y. \text{has-eq-relation } x \ y \implies \text{exp-h-prod } x = \text{exp-h-prod } y$ 

proof -
  fix x y :: nat list
  assume a:has-eq-relation x y
  then obtain f where b:bij-betw f (set x) (set y) and c: $\bigwedge z. z \in \text{set } x \implies \text{count-list } x \ z = \text{count-list } y \ (f \ z)$ 

```

```

    using eq-rel-obtain-bij[OF a] by blast
  have exp-h-prod x = prod ( (λi. r(count-list y i)) ∘ f) (set x)
    by (simp add:exp-h-prod-def c)
  also have ... = (∏ i ∈ f ' (set x). r(count-list y i))
    apply (rule prod.reindex[symmetric])
    using b bij-betw-def by blast
  also have ... = exp-h-prod y
    apply (simp add:exp-h-prod-def)
    apply (rule prod.cong)
    apply (metis b bij-betw-def)
    by simp

  finally show exp-h-prod x = exp-h-prod y by simp
qed

  hence exp-h-prod-cong: ∧p x. of-bool (has-eq-relation p x) * exp-h-prod p =
of-bool (has-eq-relation p x) * exp-h-prod x
    by simp

  have expectation f = (∑ i ∈ set as. (∑ j ∈ set as. c i * c j * expectation (h-prod
[i,j])))
    by (simp add:f-eq h-prod-def power2-eq-square sum-distrib-left sum-distrib-right
Bochner-Integration.integral-sum[OF int-M] algebra-simps)
  also have ... = (∑ i ∈ set as. (∑ j ∈ set as. c i * c j * exp-h-prod [i,j]))
    apply (rule sum.cong, simp)
    apply (rule sum.cong, simp)
    apply (subst exp-h-prod, simp, simp)
    by simp
  also have ... = (∑ i ∈ set as. (∑ j ∈ set as.
c i * c j * (sum-list (map (λp. of-bool (has-eq-relation p [i,j]) * exp-h-prod p)
(enum-partitions 2))))))
    apply (subst exp-h-prod-cong)
    apply (subst sum-partitions', simp)
    by simp
  also have ... = (∑ i ∈ set as. c i * c i * r 2)
    apply (simp add:numeral-eq-Suc exp-h-prod-elim r-one)
    by (simp add: has-eq-relation-elim distrib-left sum.distrib sum-collapse fold-sym)
  also have ... = real-of-rat (F 2 as) * ((real p) ^ 2 - 1)
    apply (subst sum-distrib-right[symmetric])
    by (simp add:c-def F-def power2-eq-square of-rat-sum of-rat-mult r-two)
  finally show b: ?B by simp

  have expectation (λx. (f x) ^ 2) = (∑ i1 ∈ set as. (∑ i2 ∈ set as. (∑ i3 ∈ set as.
(∑ i4 ∈ set as.
c i1 * c i2 * c i3 * c i4 * expectation (h-prod [i1, i2, i3, i4])))))
    apply (simp add:f-eq h-prod-def power4-eq-xxxx sum-distrib-left sum-distrib-right
Bochner-Integration.integral-sum[OF int-M])
    by (simp add:algebra-simps)
  also have ... = (∑ i1 ∈ set as. (∑ i2 ∈ set as. (∑ i3 ∈ set as. (∑ i4 ∈ set as.

```

```

  c i1 * c i2 * c i3 * c i4 * exp-h-prod [i1,i2,i3,i4]))))
  apply (rule sum.cong, simp)
  apply (rule sum.cong, simp)
  apply (rule sum.cong, simp)
  apply (rule sum.cong, simp)
  apply (subst exp-h-prod, simp, simp)
  by simp
  also have ... = (∑ i1 ∈ set as. (∑ i2 ∈ set as. (∑ i3 ∈ set as. (∑ i4 ∈ set as.
    c i1 * c i2 * c i3 * c i4 *
    (sum-list (map (λp. of-bool (has-eq-relation p [i1,i2,i3,i4]) * exp-h-prod p)
      (enum-partitions 4)))))))
    apply (subst exp-h-prod-cong)
    apply (subst sum-partitions', simp)
    by simp
  also have ... =
    3 * (∑ i ∈ set as. (∑ j ∈ set as. c i2 * c j2 * r 2 * r 2)) + ((∑ i ∈ set as.
    c i4 * r 4) - 3 * (∑ i ∈ set as. c i4 * r 2 * r 2))
    apply (simp add:numeral-eq-Suc exp-h-prod-elim r-one)
    apply (simp add:has-eq-relation-elim distrib-left sum.distrib sum-collapse fold-sym)
    by (simp add: algebra-simps sum-subtractf sum-collapse)
  also have ... = 3 * (∑ i ∈ set as. c i2 * r 2)2 + (∑ i ∈ set as. c i4 * (r
  4 - 3 * r 2 * r 2))
    apply (rule arg-cong2[where f=(+)])
    apply (simp add:power2-eq-square sum-distrib-left sum-distrib-right algebra-simps)
    apply (simp add:sum-distrib-left sum-subtractf[symmetric])
    apply (rule sum.cong, simp)
    by (simp add:algebra-simps)
  also have ... ≤ 3 * (∑ i ∈ set as. c i2)2 * (r 2)2 + (∑ i ∈ set as. c i4
  * 0)
    apply (rule add-mono)
    apply (simp add:power-mult-distrib sum-distrib-right[symmetric])
    apply (rule sum-mono, rule mult-left-mono)
    using r-four-est by simp+
  also have ... = 3 * (real-of-rat (F 2 as)2) * ((real p)2-1)2
    by (simp add:c-def r-two F-def of-rat-sum of-rat-power)

  finally have v-1: expectation (λx. (f x)2) ≤ 3 * (real-of-rat (F 2 as)2) * ((real
  p)2-1)2
    by simp

  have variance f ≤ 2*(real-of-rat (F 2 as)2) * ((real p)2-1)2
    apply (subst variance-eq[OF int-M int-M], subst b)
    apply (simp add:power-mult-distrib)
    using v-1 by simp

```

thus ?A by simp
qed

lemma f2-alg-sketch:

```

fixes  $n :: \text{nat}$ 
fixes  $as :: \text{nat list}$ 
assumes  $\varepsilon \in \{0 < .. < 1\}$ 
assumes  $\delta > 0$ 
defines  $s_1 \equiv \text{nat } \lceil 6 / \delta^2 \rceil$ 
defines  $s_2 \equiv \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 
defines  $p \equiv \text{find-prime-above } (\text{max } n \ 3)$ 
defines  $\text{sketch} \equiv \text{fold } (\lambda a \text{ state. state } \gg= \text{f2-update } a) \text{ as } (\text{f2-init } \delta \ \varepsilon \ n)$ 
defines  $\Omega \equiv \text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) \ (\lambda-. \text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ 4))$ 
shows  $\text{sketch} = \Omega \gg= (\lambda h. \text{return-pmf } (s_1, s_2, p, h, \lambda i \in \{0..<s_1\} \times \{0..<s_2\}. \text{sum-list } (\text{map } (\text{f2-hash } p \ (h \ i)) \ as)))$ 
proof –
  define  $ys$  where  $ys = \text{rev } as$ 
  have  $b:\text{sketch} = \text{foldr } (\lambda x \text{ state. state } \gg= \text{f2-update } x) \ ys \ (\text{f2-init } \delta \ \varepsilon \ n)$ 
  by ( $\text{simp add: foldr-conv-fold } ys\text{-def sketch-def}$ )
  also have  $\dots = \Omega \gg= (\lambda h. \text{return-pmf } (s_1, s_2, p, h, \lambda i \in \{0..<s_1\} \times \{0..<s_2\}. \text{sum-list } (\text{map } (\text{f2-hash } p \ (h \ i)) \ ys)))$ 
  proof ( $\text{induction } ys$ )
    case  $\text{Nil}$ 
    then show  $?case$ 
    by ( $\text{simp add:s}_1\text{-def [symmetric] } s_2\text{-def[symmetric] } p\text{-def[symmetric] } \Omega\text{-def restrict-def}$ )
  next
    case ( $\text{Cons } a \ as$ )
    have  $a:\text{f2-update } a = (\lambda x. \text{f2-update } a \ (\text{fst } x, \text{fst } (\text{snd } x), \text{fst } (\text{snd } (\text{snd } x))), \text{fst } (\text{snd } (\text{snd } (\text{snd } x))), \text{snd } (\text{snd } (\text{snd } (\text{snd } x))))$  by  $\text{simp}$ 
    show  $?case$ 
    using  $\text{Cons apply (simp del:f2-hash.simps f2-init.simps)}$ 
    apply ( $\text{subst } a$ )
    apply ( $\text{subst bind-assoc-pmf}$ )
    apply ( $\text{subst bind-return-pmf}$ )
    by ( $\text{simp add:restrict-def del:f2-hash.simps f2-init.simps cong:restrict-cong}$ )
  qed
  also have  $\dots = \Omega \gg= (\lambda h. \text{return-pmf } (s_1, s_2, p, h, \lambda i \in \{0..<s_1\} \times \{0..<s_2\}. \text{sum-list } (\text{map } (\text{f2-hash } p \ (h \ i)) \ as)))$ 
  by ( $\text{simp add: } ys\text{-def rev-map[symmetric]}$ )
  finally show  $?thesis$  by  $\text{auto}$ 
qed

theorem  $\text{f2-alg-correct}$ :
  assumes  $\varepsilon \in \{0 < .. < 1\}$ 
  assumes  $\delta > 0$ 
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  defines  $M \equiv \text{fold } (\lambda a \text{ state. state } \gg= \text{f2-update } a) \text{ as } (\text{f2-init } \delta \ \varepsilon \ n) \gg= \text{f2-result}$ 
  shows  $\mathcal{P}(\omega \text{ in measure-pmf } M. |\omega - F \ 2 \ as| \leq \delta * F \ 2 \ as) \geq 1 - \text{of-rat } \varepsilon$ 
proof –
  define  $s_1$  where  $s_1 = \text{nat } \lceil 6 / \delta^2 \rceil$ 

```

```

define  $s_2$  where  $s_2 = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 
define  $p$  where  $p = \text{find-prime-above } (\text{max } n \ 3)$ 
define  $\Omega_0$  where  $\Omega_0 =$ 
   $\text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) \ (\lambda-. \text{pmf-of-set } (\text{bounded-degree-polynomials}$ 
   $(\text{ZFact } (\text{int } p)) \ 4))$ 

define  $s_1\text{-from} :: f2\text{-state} \Rightarrow \text{nat}$  where  $s_1\text{-from} = \text{fst}$ 
define  $s_2\text{-from} :: f2\text{-state} \Rightarrow \text{nat}$  where  $s_2\text{-from} = \text{fst} \circ \text{snd}$ 
define  $p\text{-from} :: f2\text{-state} \Rightarrow \text{nat}$  where  $p\text{-from} = \text{fst} \circ \text{snd} \circ \text{snd}$ 
define  $h\text{-from} :: f2\text{-state} \Rightarrow (\text{nat} \times \text{nat} \Rightarrow \text{int set list})$  where  $h\text{-from} = \text{fst} \circ \text{snd}$ 
 $\circ \text{snd} \circ \text{snd}$ 
define  $\text{sketch-from} :: f2\text{-state} \Rightarrow (\text{nat} \times \text{nat} \Rightarrow \text{int})$  where  $\text{sketch-from} = \text{snd} \circ$ 
 $\text{snd} \circ \text{snd} \circ \text{snd}$ 

have  $p\text{-prime} : \text{Factorial-Ring.prime } p$ 
apply ( $\text{simp add:p-def}$ )
using  $\text{find-prime-above-is-prime}$  by  $\text{blast}$ 

have  $p\text{-ge-3} : p \geq 3$ 
apply ( $\text{simp add:p-def}$ )
by ( $\text{meson find-prime-above-lower-bound dual-order.trans max.cobounded2}$ )

hence  $p\text{-ge-2} : p > 2$  by  $\text{simp}$ 

hence  $p\text{-sq-ne-1} : (\text{real } p)^2 \neq 1$ 
by ( $\text{metis Num.of-nat-simps(2) nat-1 nat-one-as-int nat-power-eq-Suc-0-iff}$ 
 $\text{not-numeral-less-one of-nat-eq-iff of-nat-power zero-neq-numeral}$ )

have  $p\text{-ge-0} : p > 0$  using  $p\text{-ge-2}$  by  $\text{simp}$ 

have  $\text{fin-omega-2} : \text{finite } (\text{set-pmf } (\text{pmf-of-set } (\text{bounded-degree-polynomials } (\text{ZFact}$ 
 $(\text{int } p)) \ 4)))$ 
by ( $\text{metis fin-bounded-degree-polynomials[OF p-ge-0] ne-bounded-degree-polynomials}$ 
 $\text{set-pmf-of-set}$ )

have  $\text{fin-omega-1} : \text{finite } (\text{set-pmf } \Omega_0)$ 
apply ( $\text{simp add:\Omega_0-def set-prod-pmf}$ )
apply ( $\text{rule finite-PiE, simp}$ )
by ( $\text{metis fin-omega-2}$ )

have  $\text{as-le-p} : \bigwedge x. x \in \text{set as} \implies x < p$ 
apply ( $\text{rule order-less-le-trans[where y=n], metis assms(3)}$ )
apply ( $\text{simp add:p-def}$ )
by ( $\text{meson find-prime-above-lower-bound max.boundedE}$ )

have  $\text{fin-poly'} : \text{finite } (\text{bounded-degree-polynomials } (\text{ZFact } (\text{int } p)) \ 4)$ 
apply ( $\text{rule fin-bounded-degree-polynomials}$ )
using  $p\text{-ge-3}$  by  $\text{auto}$ 

```

```

have s2-nonzero:  $s_2 > 0$ 
using assms by (simp add:s2-def)

have s1-nonzero:  $s_1 > 0$ 
using assms by (simp add:s1-def)

have split-f2-space:  $\bigwedge x. x = (s_1\text{-from } x, s_2\text{-from } x, p\text{-from } x, h\text{-from } x, \text{sketch-from } x)$ 
by (simp add:prod-eq-iff s1-from-def s2-from-def p-from-def h-from-def sketch-from-def)

have f2-result-conv:  $f2\text{-result} = (\lambda x. f2\text{-result } (s_1\text{-from } x, s_2\text{-from } x, p\text{-from } x, h\text{-from } x, \text{sketch-from } x))$ 
by (simp add:split-f2-space[symmetric] del:f2-result.simps)

define f where  $f = (\lambda x. \text{median } (\lambda i \in \{0..<s_2\}. (\sum i_1 = 0..<s_1. (\text{rat-of-int } (\text{sum-list } (\text{map } (f2\text{-hash } p \ (x \ (i_1, i))) \ as))))^2 / (((\text{rat-of-nat } p)^2 - 1) * \text{rat-of-nat } s_1))$ 

define f3 where  $f3 = (\lambda x \ (i_1::\text{nat}) \ (i_2::\text{nat}). (\text{real-of-int } (\text{sum-list } (\text{map } (f2\text{-hash } p \ (x \ (i_1, i_2))) \ as))))^2$ 

define f2 where  $f2 = (\lambda x. \lambda i \in \{0..<s_2\}. (\sum i_1 = 0..<s_1. f3 \ x \ i_1 \ i) / (((\text{real } p)^2 - 1) * \text{real } s_1))$ 

have f2-var'':  $\bigwedge i. i < s_2 \implies \text{prob-space.variance } \Omega_0 \ (\lambda \omega. f2 \ \omega \ i) \leq (\text{real-of-rat } (\delta * F \ 2 \ as))^2 / 3$ 
proof -
fix i
assume a:  $i < s_2$ 
have b:  $\text{prob-space.indep-vars } (\text{measure-pmf } \Omega_0) \ (\lambda -. \text{borel}) \ (\lambda i_1 \ x. f3 \ x \ i_1 \ i) \ \{0..<s_1\}$ 
apply (simp add:Ω0-def, rule indep-vars-restrict-intro [where f=λj. {(j,i)}])
using a f3-def disjoint-family-on-def s1-nonzero s2-nonzero by auto

have  $\text{prob-space.variance } \Omega_0 \ (\lambda \omega. f2 \ \omega \ i) = (\sum j = 0..<s_1. \text{prob-space.variance } \Omega_0 \ (\lambda \omega. f3 \ \omega \ j \ i)) / (((\text{real } p)^2 - 1) * \text{real } s_1)^2$ 
apply (simp add: a f2-def del:Bochner-Integration.integral-divide-zero)
apply (subst prob-space.variance-divide[OF prob-space-measure-pmf])
apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
apply (subst prob-space.var-sum-all-indep[OF prob-space-measure-pmf])
apply (simp)
apply (simp)
apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
apply (metis b)
by simp

```

```

    also have ... ≤ (∑ j = 0..<s1. 2*(real-of-rat (F 2 as)2) * ((real p)2-1)2) /
    (((real p)2 - 1) * real s1)2
      apply (rule divide-right-mono)
      apply (rule sum-mono)
      apply (simp add:f3-def Ω0-def)
      apply (subst variance-prod-pmf-slice, simp add:a, simp)
      apply (rule integrable-measure-pmf-finite[OF fin-omega-2])
      apply (rule var-f2[OF p-prime p-ge-2 as-le-p], simp)
    by simp
    also have ... = 2 * (real-of-rat (F 2 as)2) / real s1
      apply (simp)
    apply (subst frac-eq-eq, simp add:s1-nonzero, metis p-sq-ne-1, simp add:s1-nonzero)
    by (simp add:power2-eq-square)
    also have ... ≤ 2 * (real-of-rat (F 2 as)2) / (6 / (real-of-rat δ)2)
      apply (rule divide-left-mono)
      apply (simp add:s1-def)
      apply (metis (mono-tags, opaque-lifting) of-rat-ceiling of-rat-divide of-rat-numeral-eq
of-rat-power real-nat-ceiling-ge)
      apply simp
      apply (rule mult-pos-pos)
      using s1-nonzero apply simp
      using assms(2) by simp
    also have ... = (real-of-rat (δ * F 2 as))2 / 3
      by (simp add:of-rat-mult algebra-simps)
    finally show prob-space.variance Ω0 (λω. f2 ω i) ≤ (real-of-rat (δ * F 2 as))2
    / 3
      by simp
  qed

```

```

  have f2-exp'': ∧i. i < s2 ⇒ prob-space.expectation Ω0 (λω. f2 ω i) = real-of-rat
  (F 2 as)
  proof -
    fix i
    assume a:i < s2
    have prob-space.expectation Ω0 (λω. f2 ω i) = (∑ j = 0..<s1. prob-space.expectation
    Ω0 (λω. f3 ω j i)) / (((real p)2 - 1) * real s1)
      apply (simp add: a f2-def)
      apply (subst Bochner-Integration.integral-sum)
      apply (rule integrable-measure-pmf-finite[OF fin-omega-1])
    by simp
    also have ... = (∑ j = 0..<s1. real-of-rat (F 2 as) * ((real p)2-1)) / (((real
    p)2 - 1) * real s1)
      apply (rule arg-cong2[where f=(/)])
      apply (rule sum.cong, simp)
      apply (simp add:f3-def Ω0-def)
      apply (subst integral-prod-pmf-slice, simp, simp add:a)
      apply (rule integrable-measure-pmf-finite[OF fin-omega-2])
      apply (subst exp-f2[OF p-prime p-ge-2 as-le-p], simp, simp)
    by simp

```

```

also have ... = real-of-rat (F 2 as)
  by (simp add:s1-nonzero p-sq-ne-1)
finally show prob-space.expectation  $\Omega_0$  ( $\lambda\omega. f2 \ \omega \ i$ ) = real-of-rat (F 2 as)
  by simp
qed

define f' where f' = ( $\lambda x. \text{median } (f2 \ x) \ s_2$ )
have real-f:  $\bigwedge x. \text{real-of-rat } (f \ x) = f' \ x$ 
  using s2-nonzero apply (simp add:f'-def f2-def f3-def f-def median-rat median-restrict cong:restrict-cong)
  by (simp add:of-rat-divide of-rat-sum of-rat-power of-rat-mult of-rat-diff)

have distr':  $M = \text{map-pmf } f \ (\text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) \ (\lambda-. \text{pmf-of-set } (\text{bounded-degree-polynomials } (ZFact \ (\text{int } p)) \ 4))))$ 
  using f2-arg-sketch[OF assms(1) assms(2), where as=as and n=n]
  apply (simp add:M-def Let-def s1-def [symmetric] s2-def[symmetric] p-def[symmetric])
  apply (subst bind-assoc-pmf)
  apply (subst bind-return-pmf)
  apply (subst f2-result-conv, simp)
  apply (simp add:s2-from-def s1-from-def p-from-def h-from-def sketch-from-def cong:restrict-cong)
  by (simp add:map-pmf-def[symmetric] f-def)

define g where  $g = (\lambda\omega. \text{real-of-rat } (\delta * F \ 2 \ as) \geq |\omega - \text{real-of-rat } (F \ 2 \ as)|)$ 
have e:  $\{\omega. \delta * F \ 2 \ as \geq |\omega - F \ 2 \ as|\} = \{\omega. (g \circ \text{real-of-rat}) \ \omega\}$ 
  apply (simp add:g-def)
  apply (rule order-antisym, rule subsetI, simp)
  apply (metis abs-of-rat of-rat-diff of-rat-less-eq)
  apply (rule subsetI, simp)
  by (metis abs-of-rat of-rat-diff of-rat-less-eq)

have median-bound-2':  $\text{prob-space.indep-vars } \Omega_0 \ (\lambda-. \text{borel}) \ (\lambda i \ \omega. f2 \ \omega \ i) \ \{0..<s_2\}$ 
  apply (subst  $\Omega_0$ -def)
  apply (rule indep-vars-restrict-intro [where f= $\lambda j. \{0..<s_1\} \times \{j\}$ ])
    apply (simp add:f2-def f3-def)
    apply (simp add:disjoint-family-on-def, fastforce)
    apply (simp add:s2-nonzero)
    apply (rule subsetI, simp add:mem-Times-iff)
  apply simp
  by simp

have median-bound-3:  $-(18 * \ln (\text{real-of-rat } \varepsilon)) \leq \text{real } s_2$ 
  apply (simp add:s2-def)
  using of-nat-ceiling by blast

have median-bound-4:  $\bigwedge i. i < s_2 \implies \mathcal{P}(\omega \text{ in } \Omega_0. \text{real-of-rat } (\delta * F \ 2 \ as) < |f2 \ \omega \ i - \text{real-of-rat } (F \ 2 \ as)|) \leq 1/3$ 
proof -
  fix i

```



```

assume  $a:i < s_2$ 
show  $\mathcal{P}(\omega \text{ in } \Omega_0. \text{real-of-rat } (\delta * F \ 2 \ as) < |f2 \ \omega \ i - \text{real-of-rat } (F \ 2 \ as)|) \leq$ 
 $1/3$ 
proof (cases as = [])
  case True
    then show ?thesis using a by (simp add:f2-def F-def f3-def)
  next
    case False
      have F-2-nonzero:  $F \ 2 \ as > 0$  using F-gr-0[OF False] by simp

      define var where  $var = \text{prob-space.variance } \Omega_0 \ (\lambda\omega. f2 \ \omega \ i)$ 
      have b-1:  $\text{real-of-rat } (F \ 2 \ as) = \text{prob-space.expectation } \Omega_0 \ (\lambda\omega. f2 \ \omega \ i)$ 
        using f2-exp'' a by metis
      have b-2:  $0 < \text{real-of-rat } (\delta * F \ 2 \ as)$ 
        using assms(2) F-2-nonzero by simp
      have b-3:  $\text{integrable } \Omega_0 \ (\lambda\omega. f2 \ \omega \ i^2)$ 
        by (rule integrable-measure-pmf-finite[OF fin-omega-1])
      have b-4:  $(\lambda\omega. f2 \ \omega \ i) \in \text{borel-measurable } \Omega_0$ 
        by (simp add:Ω0-def)
      have  $\mathcal{P}(\omega \text{ in } \Omega_0. \text{real-of-rat } (\delta * F \ 2 \ as) < |f2 \ \omega \ i - \text{real-of-rat } (F \ 2 \ as)|) \leq$ 
 $\mathcal{P}(\omega \text{ in } \Omega_0. \text{real-of-rat } (\delta * F \ 2 \ as) \leq |f2 \ \omega \ i - \text{real-of-rat } (F \ 2 \ as)|)$ 
        apply (simp add:Ω0-def)
        apply (rule pmf-mono-1)
        by simp
      also have  $\dots \leq var / (\text{real-of-rat } (\delta * F \ 2 \ as))^2$ 
        using prob-space.Chebyshev-inequality[where M=Ω0 and a=real-of-rat (δ
 $* F \ 2 \ as)$ 
        and  $f=\lambda\omega. f2 \ \omega \ i, \text{simplified}] \text{ assms(2) prob-space-measure-pmf[where}$ 
 $p=\Omega_0] \text{ F-2-nonzero}$ 
        b-1 b-2 b-3 b-4 by (simp add:var-def)
      also have  $\dots \leq 1/3$  (is ?ths)
        apply (subst pos-divide-le-eq)
        using F-2-nonzero assms(2) apply simp
        apply (simp add:var-def)
        using f2-var'' a by fastforce
      finally show ?thesis
        by blast
    qed
  qed

show ?thesis
  apply (simp add: distr' e real-f f'-def g-def Ω0-def[symmetric])
  apply (rule prob-space.median-bound-2[where M=Ω0 and ε=real-of-rat ε and
 $X=(\lambda i \ \omega. f2 \ \omega \ i), \text{simplified}]$ )
    apply (metis prob-space-measure-pmf)
    using assms apply simp
    apply (metis median-bound-2')
    apply (metis median-bound-3)
    using median-bound-4 by simp

```

qed

fun *f2-space-usage* :: (*nat* × *nat* × *rat* × *rat*) ⇒ *real* **where**
f2-space-usage (*n*, *m*, *ε*, *δ*) = (
 let *s*₁ = *nat* ⌈6 / *δ*²⌋ *in*
 let *s*₂ = *nat* ⌈−(18 * *ln* (*real-of-rat* *ε*))⌋ *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*} ×_{*D*} (λ*s*₁.
 *N*_{*S*} ×_{*D*} (λ*s*₂.
 *N*_{*S*} ×_{*D*} (λ*p*.
 (*List.product* [0..*s*₁] [0..*s*₂] →_{*S*} (*list*_{*S*} (*zfact*_{*S*} *p*))) ×_{*S*}
 (*List.product* [0..*s*₁] [0..*s*₂] →_{*S*} *I*_{*S*}))))

lemma *inj-on encode-state* (*dom encode-state*)

apply (*rule encoding-imp-inj*)
apply (*simp add:encode-state-def*)
apply (*rule dependent-encoding, metis nat-encoding*)
apply (*rule dependent-encoding, metis nat-encoding*)
apply (*rule dependent-encoding, metis nat-encoding*)
apply (*rule prod-encoding, metis encode-extensional list-encoding zfact-encoding*)
by (*metis encode-extensional int-encoding*)

theorem *f2-exact-space-usage*:

assumes *ε* ∈ {0 <..*1*}
assumes *δ* > 0
assumes ∧*a. a* ∈ *set as* ⇒ *a* < *n*
defines *M* ≡ *fold* (λ*a state. state* ≫= *f2-update a*) *as* (*f2-init δ ε n*)
shows *AE ω in M. bit-count* (*encode-state ω*) ≤ *f2-space-usage* (*n, length as, ε,*
δ)

proof –

define *s*₁ **where** *s*₁ = *nat* ⌈6 / *δ*²⌋
define *s*₂ **where** *s*₂ = *nat* ⌈−(18 * *ln* (*real-of-rat* *ε*))⌋
define *p* **where** *p* = *find-prime-above* (*max n 3*)

have *find-prime-above-3*: *find-prime-above 3* = 3
by (*simp add:find-prime-above.simps*)

have *p-ge-0*: *p* > 0
by (*metis find-prime-above-min p-def gr0I not-numeral-le-zero*)
have *p-le-n*: *p* ≤ 2 * *n* + 3
apply (*cases n ≤ 3*)

```

apply (simp add: p-def find-prime-above-3)
apply (simp add: p-def)
by (metis One-nat-def find-prime-above-upper-bound Suc-1 add-Suc-right linear
not-less-eq-eq numeral-3-eq-3)

have a:  $\bigwedge y. y \in \{0..<s_1\} \times \{0..<s_2\} \rightarrow_E \text{bounded-degree-polynomials} (ZFact (int$ 
p))  $4 \implies$ 
  bit-count (encode-state (s1, s2, p, y,  $\lambda i \in \{0..<s_1\} \times \{0..<s_2\}.$ 
sum-list (map (f2-hash p (y i)) as)))
   $\leq \text{ereal} (f2\text{-space-usage} (n, \text{length as}, \epsilon, \delta))$ 
proof –
  fix y
  assume a-1:  $y \in \{0..<s_1\} \times \{0..<s_2\} \rightarrow_E \text{bounded-degree-polynomials} (ZFact$ 
(int p))  $4$ 

  have a-2:  $y \in \text{extensional} (\{0..<s_1\} \times \{0..<s_2\})$  using a-1 PiE-iff by blast

  have a-3:  $\bigwedge x. x \in y \text{ ‘ } (\{0..<s_1\} \times \{0..<s_2\}) \implies \text{bit-count} (\text{list}_S (zfact_S p)$ 
x)
   $\leq \text{ereal} (9 + 8 * \log 2 (4 + 2 * \text{real } n))$ 
proof –
  fix x
  assume a-5:  $x \in y \text{ ‘ } (\{0..<s_1\} \times \{0..<s_2\})$ 
  have bit-count (listS (zfactS p) x)  $\leq \text{ereal} ( \text{real } 4 * (2 * \log 2 (\text{real } p) + 2)$ 
+ 1)
  apply (rule bounded-degree-polynomial-bit-count[OF p-ge-0])
  using a-1 a-5 by blast
  also have ...  $\leq \text{ereal} (\text{real } 4 * (2 * \log 2 (3 + 2 * \text{real } n) + 2) + 1)$ 
  apply simp
  apply (subst log-le-cancel-iff, simp, simp add:p-ge-0, simp)
  using p-le-n by simp
  also have ...  $\leq \text{ereal} (9 + 8 * \log 2 (4 + 2 * \text{real } n))$ 
  by simp
  finally show bit-count (listS (zfactS p) x)  $\leq \text{ereal} (9 + 8 * \log 2 (4 + 2 *$ 
real n))
  by blast
qed

have a-7:  $\bigwedge x.$ 
   $x \in (\lambda x. \text{sum-list} (\text{map} (f2\text{-hash } p (y x)) as)) \text{ ‘ } (\{0..<s_1\} \times \{0..<s_2\}) \implies$ 
 $|x| \leq (4 + 2 * \text{int } n) * \text{int} (\text{length as})$ 
proof –
  fix x
  assume x  $\in (\lambda x. \text{sum-list} (\text{map} (f2\text{-hash } p (y x)) as)) \text{ ‘ } (\{0..<s_1\} \times \{0..<s_2\})$ 
  then obtain i where i  $\in \{0..<s_1\} \times \{0..<s_2\}$  and x-def:  $x = \text{sum-list} (\text{map}$ 
(f2-hash p (y i)) as)
  by blast
  have abs x  $\leq \text{sum-list} (\text{map abs} (\text{map} (f2\text{-hash } p (y i)) as))$ 
  by (subst x-def, rule sum-list-abs)

```

```

also have ...  $\leq$  sum-list (map ( $\lambda$ -. (int  $p+1$ )) as)
  apply (simp add:comp-def del:f2-hash.simps)
  apply (rule sum-list-mono)
  using p-ge-0 by simp
also have ... = int (length as) * (int  $p+1$ )
  by (simp add: sum-list-triv)
also have ...  $\leq$  int (length as) * ( $4+2*(\text{int } n)$ )
  apply (rule mult-mono, simp)
  using p-le-n apply linarith
  by simp+
finally show  $abs\ x \leq (4 + 2 * \text{int } n) * \text{int } (\text{length } as)$ 
  by (simp add: mult.commute)
qed

have bit-count (encode-state ( $s_1, s_2, p, y, \lambda i \in \{0..<s_1\} \times \{0..<s_2\}$ .
  sum-list (map (f2-hash p (y i)) as)))
   $\leq$  ereal ( $2 * (\log 2 (\text{real } s_1 + 1)) + 1$ )
  + (ereal ( $2 * (\log 2 (\text{real } s_2 + 1)) + 1$ )
  + (ereal ( $2 * (\log 2 (1 + \text{real } (2*n+3))) + 1$ )
  + ((ereal ( $\text{real } s_1 * \text{real } s_2$ ) * ( $10 + 8 * \log 2 (4 + 2 * \text{real } n)$ )) + 1)
  + (ereal ( $\text{real } s_1 * \text{real } s_2$ ) * ( $3 + 2 * \log 2 (\text{real } (\text{length } as) * (4 + 2 * \text{real } n) + 1$ )) + 1))))))
  using a-2
apply (simp add: encode-state-def s1-def[symmetric] s2-def[symmetric] p-def[symmetric]

  dependent-bit-count prod-bit-count encode-extensional-def
  del:encode-dependent-sum.simps encode-prod.simps N_S.simps plus-ereal.simps
of-nat-add)
  apply (rule add-mono, rule nat-bit-count)
  apply (rule add-mono, rule nat-bit-count)
  apply (rule add-mono, rule nat-bit-count-est, metis p-le-n)
  apply (rule add-mono)
  apply (rule list-bit-count-estI[where  $a=9 + 8 * \log 2 (4 + 2 * \text{real } n)$ ],
rule a-3, simp, simp)
  apply (rule list-bit-count-estI[where  $a=2 * \log 2 (\text{real-of-int } (\text{int } ((4+2*n) * \text{length } as)+1))+2$ ]])
  apply (rule int-bit-count-est)
  apply (simp add:a-7)
  by (simp add:algebra-simps)
also have ... = ereal (f2-space-usage ( $n, \text{length } as, \varepsilon, \delta$ ))
  by (simp add:distrib-left[symmetric] s1-def[symmetric] s2-def[symmetric]
p-def[symmetric])
finally show bit-count (encode-state ( $s_1, s_2, p, y, \lambda i \in \{0..<s_1\} \times \{0..<s_2\}$ .
  sum-list (map (f2-hash p (y i)) as)))
   $\leq$  ereal (f2-space-usage ( $n, \text{length } as, \varepsilon, \delta$ )) by blast
qed

show ?thesis
  apply (subst AE-measure-pmf-iff)

```

```

apply (subst M-def)
apply (subst f2-alg-sketch[OF assms(1) assms(2), where n=n and as=as])
apply (simp add: s1-def[symmetric] s2-def[symmetric] p-def[symmetric] del:f2-space-usage.simps)
apply (subst set-prod-pmf, simp)
apply (simp add: PiE-iff del:f2-space-usage.simps)
apply (subst set-pmf-of-set, metis ne-bounded-degree-polynomials, metis fin-bounded-degree-polynomials[OF
p-ge-0])
  by (metis a)
qed

```

theorem *f2-asymptotic-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 –

```

define n-of :: nat  $\times$  nat  $\times$  rat  $\times$  rat  $\Rightarrow$  nat where n-of =  $(\lambda(n, m, \varepsilon, \delta). n)$ 
define m-of :: nat  $\times$  nat  $\times$  rat  $\times$  rat  $\Rightarrow$  nat where m-of =  $(\lambda(n, m, \varepsilon, \delta). m)$ 
define  $\varepsilon$ -of :: nat  $\times$  nat  $\times$  rat  $\times$  rat  $\Rightarrow$  rat where  $\varepsilon$ -of =  $(\lambda(n, m, \varepsilon, \delta). \varepsilon)$ 
define  $\delta$ -of :: nat  $\times$  nat  $\times$  rat  $\times$  rat  $\Rightarrow$  rat where  $\delta$ -of =  $(\lambda(n, m, \varepsilon, \delta). \delta)$ 

```

```

define g where g =  $(\lambda x. (\ln\ (1 / of\text{-}rat\ (\varepsilon\text{-}of\ x))) / (of\text{-}rat\ (\delta\text{-}of\ x))^2 * (\ln\ (real\ (n\text{-}of\ x)) + \ln\ (real\ (m\text{-}of\ x))))$ 

```

```

have n-inf:  $\bigwedge c. eventually\ (\lambda x. c \leq (real\ (n\text{-}of\ x)))\ ?F$ 
apply (simp add: n-of-def case-prod-beta')
apply (subst eventually-prod1', simp add: prod-filter-eq-bot)
by (meson eventually-at-top-linorder nat-ceiling-le-eq)

```

```

have m-inf:  $\bigwedge c. eventually\ (\lambda x. c \leq (real\ (m\text{-}of\ x)))\ ?F$ 
apply (simp add: m-of-def case-prod-beta')
apply (subst eventually-prod2', simp add: prod-filter-eq-bot)
apply (subst eventually-prod1', simp add: prod-filter-eq-bot)
by (meson eventually-at-top-linorder nat-ceiling-le-eq)

```

```

have eps-inf:  $\bigwedge c. eventually\ (\lambda x. c \leq 1 / (real\text{-}of\text{-}rat\ (\varepsilon\text{-}of\ x)))\ ?F$ 
apply (simp add:  $\varepsilon$ -of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod1', simp)
by (rule inv-at-right-0-inf)

```

```

have delta-inf:  $\bigwedge c. eventually\ (\lambda x. c \leq 1 / (real\text{-}of\text{-}rat\ (\delta\text{-}of\ x)))\ ?F$ 
apply (simp add:  $\delta$ -of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
by (rule inv-at-right-0-inf)

```

```

have zero-less-eps: eventually ( $\lambda x. 0 < (\text{real-of-rat } (\varepsilon\text{-of } x))$ ) ?F
  apply (simp add:  $\varepsilon\text{-of-def case-prod-beta}$ )
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod1', simp)
  by (rule eventually-at-rightI[where  $b=1$ ], simp, simp)

have zero-less-delta: eventually ( $\lambda x. 0 < (\text{real-of-rat } (\delta\text{-of } x))$ ) ?F
  apply (simp add:  $\delta\text{-of-def case-prod-beta}$ )
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod2', simp)
  by (rule eventually-at-rightI[where  $b=1$ ], simp, simp)

have unit-1: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. 1 / (\text{real-of-rat } (\delta\text{-of } x))^2)$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF zero-less-delta delta-inf[where
 $c=1$ ]]]])
  by (metis one-le-power power-one-over)

have unit-2: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF zero-less-eps eps-inf[where
 $c=\exp 1$ ]]]])
  by (meson abs-ge-self dual-order.trans exp-gt-zero ln-ge-iff order-trans-rules(22))

have unit-3: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. \text{real } (n\text{-of } x))$ 
  by (rule landau-o.big-mono, simp, rule n-inf)

have unit-4: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. \text{real } (m\text{-of } x))$ 
  by (rule landau-o.big-mono, simp, rule m-inf)

have unit-5: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono [OF n-inf[where  $c=\exp 1$ ]])
  by (metis abs-ge-self linorder-not-le ln-ge-iff not-exp-le-zero order.trans)

have unit-6: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x)))$ 
  apply (rule landau-sum-1)
  apply (rule eventually-ln-ge-iff[OF n-inf])
  apply (rule eventually-ln-ge-iff[OF m-inf])
  by (rule unit-5)

have unit-7: ( $\lambda-. 1$ )  $\in O[?F](\lambda x. 1 / \text{real-of-rat } (\varepsilon\text{-of } x))$ 
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono [OF eventually-conj[OF zero-less-eps eps-inf[where
 $c=1$ ]]]])
  by simp

```

```

have unit-8:  $(\lambda x. 1) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)) * (\ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$ 
apply (subst (2) div-commute)
apply (rule landau-o.big-mult-1[OF unit-1])
by (rule landau-o.big-mult-1[OF unit-2 unit-6])

have unit-9:  $(\lambda x. 1) \in O[?F](\lambda x. \text{real } (n\text{-of } x) * \text{real } (m\text{-of } x))$ 
by (rule landau-o.big-mult-1[OF unit-3 unit-4])

have zero-less-eps: eventually  $(\lambda x. 0 < (\text{real-of-rat } (\varepsilon\text{-of } x)))$  ?F
apply (simp add:ε-of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod1', simp)
by (rule eventually-at-rightI[where b=1], simp, simp)

have l1:  $(\lambda x. \text{real } (\text{nat } \lceil 6 / (\delta\text{-of } x)^2 \rceil)) \in O[?F](\lambda x. 1 / (\text{real-of-rat } (\delta\text{-of } x))^2)$ 
apply (rule landau-real-nat)
apply (subst landau-o.big.in-cong[where g= $\lambda x. \text{real-of-int } \lceil 6 / (\text{real-of-rat } (\delta\text{-of } x))^2 \rceil$ ])
apply (rule always-eventually, rule allI, rule arg-cong[where f=real-of-int])
apply (metis (no-types, opaque-lifting) of-rat-ceiling of-rat-divide of-rat-numeral-eq of-rat-power)
apply (rule landau-ceil[OF unit-1])
by (rule landau-const-inv, simp, simp)

have l2:  $(\lambda x. \text{real } (\text{nat } \lceil -(18 * \ln (\text{real-of-rat } (\varepsilon\text{-of } x))) \rceil)) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)))$ 
apply (rule landau-real-nat, rule landau-ceil, simp add:unit-2)
apply (subst minus-mult-right)
apply (subst cmult-in-bigo-iff, rule disjI2)
apply (rule landau-o.big-mono)
apply (rule eventually-mono[OF zero-less-eps])
by (subst ln-div, simp+)

have l3:  $(\lambda x. \log 2 (\text{real } (m\text{-of } x) * (4 + 2 * \text{real } (n\text{-of } x)) + 1)) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x)))$ 
apply (simp add:log-def)
apply (rule landau-o.big-trans[where g= $\lambda x. \ln (\text{real } (n\text{-of } x) * \text{real } (m\text{-of } x))$ ])
apply (rule landau-ln-2[where a=2], simp, simp)
apply (rule eventually-mono[OF eventually-conj[OF m-inf[where c=2] n-inf[where c=1]]])
apply (metis dual-order.trans mult-left-mono mult-of-nat-commute of-nat-0-le-iff verit-prod-simplify(1))
apply (rule sum-in-bigo)
apply (subst mult.commute)
apply (rule landau-o.mult)
apply (rule sum-in-bigo, simp add:unit-3, simp)
apply simp

```

apply (*simp add:unit-9*)
apply (*subst landau-o.big.in-cong*[**where** $g=\lambda x. \ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x))$])
apply (*rule eventually-mono*[*OF eventually-conj*[*OF m-inf*[**where** $c=1$] *n-inf*[**where** $c=1$]]])
by (*subst ln-mult, simp+*)

have l_4 : $(\lambda x. \log 2 (4 + 2 * \text{real } (n\text{-of } x))) \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x)))$
apply (*rule landau-sum-1*)
apply (*rule eventually-ln-ge-iff*[*OF n-inf*])
apply (*rule eventually-ln-ge-iff*[*OF m-inf*])
apply (*simp add:log-def*)
apply (*rule landau-ln-2*[**where** $a=2$], *simp, simp, rule n-inf*)
apply (*rule sum-in-bigo, simp, simp add:unit-3*)
by *simp*

have l_5 : $(\lambda x. \ln (\text{real } (\text{nat } \lceil 6 / (\delta\text{-of } x)^2 \rceil) + 1)) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)) * (\ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$
apply (*subst (2) div-commute*)
apply (*rule landau-o.big-mult-1*)
apply (*rule landau-ln-3, simp*)
apply (*rule sum-in-bigo, rule l1, rule unit-1*)
by (*rule landau-o.big-mult-1*[*OF unit-2 unit-6*])

have l_6 : $(\lambda x. \ln (4 + 2 * \text{real } (n\text{-of } x))) \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)) * (\ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$
apply (*subst (2) div-commute*)
apply (*rule landau-o.big-mult-1'*[*OF unit-1*])
apply (*rule landau-o.big-mult-1'*[*OF unit-2*])
using l_4 **by** (*simp add:log-def*)

have l_7 : $(\lambda x. \ln (\text{real } (\text{nat } \lceil - (18 * \ln (\text{real-of-rat } (\varepsilon\text{-of } x))) \rceil) + 1)) \in O[?F](\lambda x.$

$\ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)) * (\ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x))) / (\text{real-of-rat } (\delta\text{-of } x))^2)$
apply (*subst (2) div-commute*)
apply (*rule landau-o.big-mult-1'*[*OF unit-1*])
apply (*rule landau-o.big-mult-1*)
apply (*rule landau-ln-2*[**where** $a=2$], *simp, simp, simp add:eps-inf*)
apply (*rule sum-in-bigo*)
apply (*rule landau-nat-ceil*[*OF unit-7*])
apply (*subst minus-mult-right*)
apply (*subst cmult-in-bigo-iff, rule disjI2*)
apply (*subst landau-o.big.in-cong*[**where** $g=\lambda x. \ln (1 / (\text{real-of-rat } (\varepsilon\text{-of } x)))$])
apply (*rule eventually-mono*[*OF zero-less-eps*])
apply (*subst ln-div, simp, simp, simp*)


```

    apply (rule landau-ln-3[OF eps-inf], simp)
  apply (rule unit-7)
  by (rule unit-6)

have f2-space-usage = (λx. f2-space-usage (n-of x, m-of x, ε-of x, δ-of x))
  apply (rule ext)
  by (simp add:case-prod-beta' n-of-def ε-of-def δ-of-def m-of-def)
also have ... ∈ O[?F](g)
  apply (simp add:g-def Let-def)
  apply (rule sum-in-bigo-r)
  apply (subst (2) div-commute, subst mult.assoc)
  apply (rule landau-o.mult, simp add:l1)
  apply (rule landau-o.mult, simp add:l2)
  apply (rule sum-in-bigo-r, simp add:l3)
  apply (rule sum-in-bigo-r, simp add:l4, simp add:unit-6)
  apply (rule sum-in-bigo-r, simp add:log-def l6)
  apply (rule sum-in-bigo-r, simp add:log-def l7)
  apply (rule sum-in-bigo-r, simp add:log-def l5)
  by (simp add:unit-8)
also have ... = O[?F](?rhs)
  apply (rule arg-cong2[where f=bigo], simp)
  apply (rule ext)
  by (simp add:case-prod-beta' g-def n-of-def ε-of-def δ-of-def m-of-def)
finally show ?thesis by simp
qed

end

```

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

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  $s_1 = nat\ \lceil 3 * real\ k * (real\ n)\ powr\ (1 - 1 / real\ k) / (real\ of\ rat\ \delta)^2 \rceil$ ;
    let  $s_2 = nat\ \lceil -18 * \ln\ (real\ of\ rat\ \varepsilon) \rceil$ ;
    return-pmf ( $s_1, s_2, k, 0, (\lambda\cdot. undefined)$ )
  }

```

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.. $s_1$ } × {0.. $s_2$ }) (λ-. bernoulli-pmf (1/(real m+1)));
  return-pmf (s1, s2, k, m+1, λi ∈ {0.. $s_1$ } × {0.. $s_2$ }.
    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.. $s_2$ }.
      (∑ i1 ∈ {0.. $s_1$ } . rat-of-nat (let t = snd (r (i1, i2)) + 1 in m * (tk - (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.. $s_1$ } × {0.. $s_2$ }) (λ-. bernoulli-pmf (1/(real m+1)));
      return-pmf (λi ∈ {0.. $s_1$ } × {0.. $s_2$ }.
        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
by (rule pmf-eqI, simp add:indicator-def)

lemma split-space:

$(\sum a \in \{(u, v). v < \text{count-list as } u\}. (f (\text{snd } a))) =$
 $(\sum u \in \text{set as}. (\sum v \in \{0..<\text{count-list as } u\}. (f v)))$ (is ?lhs = ?rhs)

proof –

define A **where** $A = (\lambda u. \{u\} \times \{v. v < \text{count-list as } u\})$

have $a : \bigwedge u v. u < \text{count-list as } v \implies v \in \text{set as}$
by (*subst count-list-gr-1, force*)

have ?lhs = $\text{sum } (f \circ \text{snd}) \ (\bigcup (A \text{ ' set as}))$
apply (*rule sum.cong, rule order-antisym*)
apply (*rule subsetI, simp add:A-def case-prod-beta' mem-Times-iff a*)
apply (*rule subsetI, simp add:A-def case-prod-beta' mem-Times-iff a*)
by *simp*

also have ... = $\text{sum } (\lambda x. \text{sum } (f \circ \text{snd}) (A x)) (\text{set as})$
by (*rule sum.UNION-disjoint, simp, simp add:A-def, simp add:A-def, blast*)

also have ... = ?rhs
apply (*rule sum.cong, simp*)
apply (*subst sum.reindex[symmetric]*)
apply (*simp add:A-def inj-on-def*)
apply (*simp add:A-def*)
apply (*rule sum.cong*)
using *lessThan-atLeast0* **apply** *blast*
by *simp*

finally show ?thesis **by** *blast*

qed

lemma

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

proof –

have $\{(u, v). v < \text{count-list as } u\} \subseteq \text{set as} \times \{k. k < \text{length as}\}$
apply (*rule subsetI, simp add:case-prod-beta mem-Times-iff count-list-gr-1*)
by (*metis count-le-length order-less-le-trans*)

thus *fin-space*: *finite* $\{(u, v). v < \text{count-list as } u\}$
using *finite-subset* **by** *blast*

have $(as ! 0, 0) \in \{(u, v). v < \text{count-list as } u\}$
apply (*simp*)
using *assms(1)*
by (*metis count-list-gr-1 gr0I length-greater-0-conv not-one-le-zero nth-mem*)

thus $\{(u, v). v < \text{count-list as } u\} \neq \{\}$ **by** *blast*

show $\text{card } \{(u, v). v < \text{count-list as } u\} = \text{length as}$
using *fin-space split-space* [**where** $f = \lambda \cdot. (1 :: \text{nat})$, **where** $as = as$]
by (*simp add:sum-count-set* [**where** $X = \text{set as}$ **and** $xs = as$, *simplified*])

qed

```

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

  have  $a3: \bigwedge x y. y < \text{length } as \implies x < y \implies as ! x = as ! y \implies$ 
     $\text{count-list } (\text{drop } (Suc x) as) (as ! x) \neq \text{count-list } (\text{drop } (Suc y) as) (as !$ 
 $y)$ 
     $(\text{is } \bigwedge x y. - \implies - \implies - \implies ?ths \ x \ y)$ 
  proof -
    fix  $x \ y$ 
    assume  $a3-1: y < \text{length } as$ 
    assume  $a3-2: x < y$ 
    assume  $a3-3: as ! x = as ! y$ 
    have  $a3-4: \text{drop } (Suc x) as = \text{take } (y-x) (\text{drop } (Suc x) as) @ \text{drop } (Suc y) as$ 
    apply ( $\text{subst append-take-drop-id}[\text{where } xs = \text{drop } (Suc x) as \text{ and } n = y - x,$ 
 $\text{symmetric}]$ )
    using  $a3-2$  by simp
    have  $\text{count-list } (\text{drop } (Suc x) as) (as ! x) = \text{count-list } (\text{take } (y-x) (\text{drop } (Suc$ 
 $x) as)) (as ! y) +$ 
     $\text{count-list } (\text{drop } (Suc y) as) (as ! y)$ 
    using  $a3-3$  by ( $\text{subst } a3-4, \text{simp add:count-list-append}$ )
    moreover have  $\text{count-list } (\text{take } (y-x) (\text{drop } (Suc x) as)) (as ! y) \geq 1$ 
    apply ( $\text{subst count-list-gr-1}[\text{symmetric}]$ )
    apply ( $\text{simp add:set-conv-nth}$ )
    apply ( $\text{rule exI}[\text{where } x = y - x - 1]$ )
    apply ( $\text{subst nth-take, meson diff-less } a3-2 \text{ zero-less-diff zero-less-one}$ )
    apply ( $\text{subst nth-drop}$ ) using  $a3-1 \ a3-2$  apply simp
    apply ( $\text{rule conjI, rule arg-cong2}[\text{where } f = (!), \text{simp}]$ )
    using  $a3-2$  apply simp
    apply ( $\text{rule conjI}$ )
    using  $a3-1 \ a3-2$  apply simp
    by ( $\text{meson diff-less } a3-2 \text{ zero-less-diff zero-less-one}$ )
    ultimately show  $?ths \ x \ y$  by presburger
qed

  have  $a1: \text{inj-on } f \ \{k. k < \text{length } as\}$ 
proof ( $\text{rule inj-onI}$ )
  fix  $x \ y$ 
  assume  $x \in \{k. k < \text{length } as\}$ 
  moreover assume  $y \in \{k. k < \text{length } as\}$ 
  moreover assume  $f \ x = f \ y$ 
  ultimately show  $x = y$ 
    apply ( $\text{cases } x < y, \text{simp add:f-def,metis } a3$ )
    apply ( $\text{cases } y < x, \text{simp add:f-def,metis } a3$ )

```

```

    by simp
  qed
  have a2-1:  $\bigwedge x. x < \text{length } as \implies \text{count-list } (\text{drop } (\text{Suc } x) \text{ as}) (as ! x) < \text{count-list } as (as ! x)$ 
  proof -
    fix x
    assume a:x < length as
    have 1  $\leq \text{count-list } (\text{take } (\text{Suc } x) \text{ as}) (as ! x)$ 
    apply (subst count-list-gr-1[symmetric])
    using a by (simp add: take-Suc-conv-app-nth)
    hence count-list (drop (Suc x) as) (as ! x) < count-list (take (Suc x) as) (as ! x) + count-list (drop (Suc x) as) (as ! x)
    by (simp)
    also have ... = count-list as (as ! x)
    by (simp add: count-list-append[symmetric])
    finally show count-list (drop (Suc x) as) (as ! x) < count-list as (as ! x)
    by blast
  qed
  have a2:  $f \text{ ` } \{k. k < \text{length } as\} = \{(u, v). v < \text{count-list } as \ u\}$ 
  apply (rule card-seteq)
  apply (metis fin-space[OF assms(1)])
  apply (rule image-subsetI, simp add:f-def)
  apply (metis a2-1)
  apply (subst card-image[OF a1])
  by (subst card-space[OF assms(1)], simp)

  have bij-betw f  $\{k. k < \text{length } as\} \{(u, v). v < \text{count-list } as \ u\}$ 
  using a1 a2 by (simp add:bij-betw-def)
  thus ?thesis
  using assms apply (subst map-pmf-def[symmetric])
  by (rule map-pmf-of-set-bij-betw, simp add:f-def, blast, simp)
qed

lemma fk-alg-aux-4:
  assumes as  $\neq []$ 
  shows fold  $(\lambda x (c, state). (c+1, state \ggg \text{fk-update'' } x \ c)) \ as \ (0, \text{return-pmf undefined}) =$ 
 $(\text{length } as, \text{pmf-of-set } \{k. k < \text{length } as\} \ggg (\lambda k. \text{return-pmf } (as ! k, \text{count-list } (\text{drop } (k+1) \ as) (as ! k))))$ 
  using assms
  proof (induction as rule:rev-nonempty-induct)
    case (single x)
    have c: $\bigwedge c. \text{fk-update'' } x \ c = (\lambda a. \text{fk-update'' } x \ c \ (\text{fst } a, \text{snd } a))$ 
    by auto
    have b: $\{(u, v). v < (\text{if } x = u \text{ then count-list } [] \ u + 1 \text{ else count-list } [] \ u)\} = \{(x, 0)\}$ 
    apply (rule order-antisym, rule subsetI, simp add:case-prod-beta)
    apply (metis (full-types) add-cancel-left-left count-list.simps(1) less-nat-zero-code less-one prod.collapse)
  qed

```

```

    by (rule subsetI, simp)
  have a:bernoulli-pmf 1 = return-pmf True
    by (rule pmf-eqI, simp add:indicator-def)
  show ?case using single
    apply (simp add:bind-return-pmf pmf-of-set-singleton)
    apply (subst c, subst fk-update''.simps)
    by (simp add:a bind-return-pmf)
next
  case (snoc x xs)
  have c:  $\bigwedge c. \text{fk-update}'' x c = (\lambda a. \text{fk-update}'' x c (\text{fst } a, \text{snd } a))$ 
    by auto
  have a:  $\bigwedge y. \text{pmf-of-set } \{k. k < \text{length } xs\} \gg (\lambda k. \text{return-pmf } (xs ! k, \text{count-list } (\text{drop } (\text{Suc } k) xs) (xs ! k))) \gg$ 
    ( $\lambda xa. \text{return-pmf } (\text{if } y \text{ then } (x, 0) \text{ else } (\text{fst } xa, \text{snd } xa + (\text{of-bool } (\text{fst } xa = x))))$ )
    =  $\text{pmf-of-set } \{k. k < \text{length } xs\} \gg (\lambda k. \text{return-pmf } (\text{if } y \text{ then } (\text{length } xs) \text{ else } k)) \gg (\lambda k. \text{return-pmf } ((xs@[x]) ! k, \text{count-list } (\text{drop } (\text{Suc } k) (xs@[x])) ((xs@[x]) ! k)))$ 
    apply (simp add:bind-return-pmf)
    apply (rule bind-pmf-cong, simp)
    apply (subst (asm) set-pmf-of-set)
    using snoc apply blast apply simp
    by (simp add:nth-append count-list-append)

  show ?case using snoc
    apply (simp del:drop-append, subst c, subst fk-update''.simps)
    apply (subst bind-commute-pmf)
    apply (subst bind-assoc-pmf)
    apply (simp add:a del:drop-append)
    apply (subst bind-assoc-pmf[symmetric])
    apply (subst bind-assoc-pmf[symmetric])
    apply (rule arg-cong2[where f=bind-pmf])
    apply (rule pmf-eqI)
    apply (subst pmf-bind)
    apply (subst pmf-of-set, blast, simp)
    apply (subst pmf-bind)
    apply (simp)
    apply (subst measure-pmf-of-set, blast, simp)
    apply (simp add:indicator-def)
    apply (subst frac-eq-eq, simp, linarith)
    apply (simp add:algebra-simps)
    by simp
qed

```

definition *if-then-else* **where** *if-then-else* $p \ q \ r = (\text{if } p \text{ then } q \text{ else } r)$

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 (λx (c, state). (c+1, state ≫≡ fk-update' x s1 s2 c)) as (0, return-pmf (λ-.
  undefined))
  = (length as, prod-pmf ({0..<s1} × {0..<s2}) (λ-. (snd (fold (λx (c, state).
  (c+1, state ≫≡ fk-update'' x c)) as (0, return-pmf undefined))))))
  (is ?lhs = ?rhs)
proof (induction as rule:rev-induct)
  case Nil
  thus ?case
    apply (simp, rule pmf-eqI)
    apply (simp add:pmf-prod-pmf)
    apply (rule conjI, rule impI)
    apply (simp add:indicator-def, rule conjI, rule impI)
    apply force
    using extensional-arb apply fastforce
    apply (simp add:extensional-def indicator-def)
    by blast
next
  case (snoc x xs)
  obtain t1 t2 where t-def:
    (t1, t2) = fold (λx (c, state). (Suc c, state ≫≡ fk-update'' x c)) xs (0, return-pmf
  undefined)
    using surj-pair
    by (smt (z3))
  have a:fk-update' x s1 s2 (length xs) = (λa. fk-update' x s1 s2 (length xs) a)
    by auto
  have c:∧c. fk-update'' x c = (λa. fk-update'' x c (fst a, snd a))
    by auto
  have fst (fold (λx (c, state). (Suc c, state ≫≡ fk-update'' x c)) xs (0, return-pmf
  undefined)) = length xs
    by (induction xs rule:rev-induct, simp, simp add:case-prod-beta)
  hence d:t1 = length xs
    by (metis t-def fst-conv)

show ?case using snoc
  apply (simp del:fk-update''.simps fk-update'.simps)
  apply (simp add:t-def[symmetric])
  apply (subst a[simplified])
  apply (subst pair-pmfI)
  apply (subst pair-pmf-ptw, simp)
  apply (subst bind-assoc-pmf)
  apply (subst bind-return-pmf)
  apply (subst if-then-else-def[symmetric])
  apply (simp add:comp-def cong:restrict-cong)
  apply (subst map-ptw, simp)
  apply (subst if-then-else-def)
  apply (rule arg-cong2[where f=prod-pmf], simp)
  apply (rule ext)
  apply (subst c, subst fk-update''.simps, simp)

```

```

    apply (simp add:d)
    apply (subst pair-pmfI)
    apply (rule arg-cong2[where f=bind-pmf], simp)
    by force
qed

lemma fk-alg-aux-1:
  fixes k :: nat
  fixes ε :: rat
  assumes δ > 0
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  assumes  $as \neq []$ 
  defines sketch  $\equiv \text{fold } (\lambda a \text{ state}. \text{state} \gg= \text{fk-update } a) \text{ as } (\text{fk-init } k \delta \varepsilon n)$ 
  defines s1  $\equiv \text{nat } \lceil 3 * \text{real } k * (\text{real } n) \text{ powr } (1 - 1 / \text{real } k) / (\text{real-of-rat } \delta)^2 \rceil$ 
  defines s2  $\equiv \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 
  shows sketch =
    map-pmf ( $\lambda x. (s_1, s_2, k, \text{length } as, x)$ )
    (snd (fold ( $\lambda x (c, \text{state}). (c+1, \text{state} \gg= \text{fk-update}' x s_1 s_2 c)$ ) as (0, return-pmf
    ( $\lambda \cdot. \text{undefined}$ ))))
  using assms(3)
proof (subst sketch-def, induction as rule:rev-nonempty-induct)
  case (single x)
  then show ?case
    by (simp add: map-bind-pmf bind-return-pmf s1-def[symmetric] s2-def[symmetric])
next
  case (snoc x xs)
  obtain t1 t2 where t:
    fold ( $\lambda x (c, \text{state}). (\text{Suc } c, \text{state} \gg= \text{fk-update}' x s_1 s_2 c)$ ) xs (0, return-pmf
    ( $\lambda \cdot. \text{undefined}$ ))
    = (t1, t2)
  by fastforce

  have fst (fold ( $\lambda x (c, \text{state}). (\text{Suc } c, \text{state} \gg= \text{fk-update}' x s_1 s_2 c)$ ) xs (0,
  return-pmf ( $\lambda \cdot. \text{undefined}$ )))
    = length xs
  by (induction xs rule:rev-induct, simp, simp add:split-beta)
  hence t1: t1 = length xs using t fst-conv by auto

  show ?case using snoc
    apply (simp add: s1-def[symmetric] s2-def[symmetric] t del:fk-update'.simps
    fk-update.simps)
    apply (subst bind-map-pmf)
    apply (subst map-bind-pmf)
    apply simp
    by (subst map-bind-pmf, simp add:t1)
qed

lemma power-diff-sum:

```


assumes $k > 0$
shows $(a :: 'a :: \{comm-ring-1, power\})^{\wedge} k - b^{\wedge} k = (a-b) * \sum (\lambda i. a^{\wedge} i * b^{\wedge} (k-1-i)) \{0..<k\}$ (**is** $?lhs = ?rhs$)
proof –
have $?rhs = \sum (\lambda i. a * (a^{\wedge} i * b^{\wedge} (k-1-i))) \{0..<k\} - \sum (\lambda i. b * (a^{\wedge} i * b^{\wedge} (k-1-i))) \{0..<k\}$
by (*simp add: sum-distrib-left[symmetric] algebra-simps*)
also have $\dots = \sum ((\lambda i. (a^{\wedge} i * b^{\wedge} (k-i))) \circ (\lambda i. i+1)) \{0..<k\} - \sum (\lambda i. (a^{\wedge} i * b^{\wedge} (k-i))) \{0..<k\}$
apply (*rule arg-cong2[where f=(-)]*)
apply (*rule sum.cong, simp, simp add: algebra-simps*)
apply (*rule sum.cong, simp*)
apply (*subst mult.assoc[symmetric], subst mult.commute, subst mult.assoc*)
by (*rule arg-cong2[where f=(*)], simp, simp add: power-eq-if*)
also have $\dots = \sum (\lambda i. (a^{\wedge} i * b^{\wedge} (k-i))) (\text{insert } k \{1..<k\}) - \sum (\lambda i. (a^{\wedge} i * b^{\wedge} (k-i))) (\text{insert } 0 \{1..<k\})$
apply (*rule arg-cong2[where f=(-)]*)
apply (*subst sum.reindex[symmetric], simp*)
apply (*rule sum.cong*) **using** *assms* **apply** (*simp add: atLeastLessThanSuc, simp*)
apply (*rule sum.cong*) **using** *assms Icc-eq-insert-lb-nat*
apply (*metis One-nat-def Suc-pred atLeastLessThanSuc-atLeastAtMost le-add1 le-add-same-cancel1*)
by *simp*
also have $\dots = ?lhs$
by *simp*
finally show $?thesis$ **by** *presburger*
qed

lemma *power-diff-est*:

assumes $k > 0$
assumes $(a :: real) \geq b$
assumes $b \geq 0$
shows $a^{\wedge} k - b^{\wedge} k \leq (a-b) * k * a^{\wedge} (k-1)$

proof –
have $\bigwedge i. i < k \implies a^{\wedge} i * b^{\wedge} (k-1-i) \leq a^{\wedge} i * a^{\wedge} (k-1-i)$
apply (*rule mult-left-mono, rule power-mono, metis assms(2), metis assms(3)*)
using *assms* **by** *simp*
also have $\bigwedge i. i < k \implies a^{\wedge} i * a^{\wedge} (k-1-i) = a^{\wedge} (k - \text{Suc } 0)$
apply (*subst power-add[symmetric]*)
apply (*rule arg-cong2[where f=power], simp*)
using *assms(1)* **by** *simp*
finally have $t: \bigwedge i. i < k \implies a^{\wedge} i * b^{\wedge} (k-1-i) \leq a^{\wedge} (k - \text{Suc } 0)$
by *blast*
have $a^{\wedge} k - b^{\wedge} k = (a-b) * \sum (\lambda i. a^{\wedge} i * b^{\wedge} (k-1-i)) \{0..<k\}$
by (*rule power-diff-sum[OF assms(1)]*)
also have $\dots \leq (a-b) * k * a^{\wedge} (k - \text{Suc } 0)$
apply (*subst mult.assoc*)
apply (*rule mult-left-mono*)

```

    apply (rule sum-mono[where g= $\lambda x. a^{k-1}$  and  $K=\{0..k\}$ , simplified])
    apply (metis t)
    using assms(2) by auto
    finally show ?thesis by simp
qed

```

Specialization of the Hoelder inequality for sums.

```

lemma Holder-inequality-sum:
  assumes  $p > (0::real)$   $q > 0$   $1/p + 1/q = 1$ 
  assumes finite A
  shows  $|\text{sum } (\lambda x. f x * g x) A| \leq (\text{sum } (\lambda x. |f x|^p) A)^{1/p} * (\text{sum } (\lambda x. |g x|^q) A)^{1/q}$ 
  using assms apply (simp add: lebesgue-integral-count-space-finite[symmetric])
  apply (rule Lp.Holder-inequality)
  by (simp add: integrable-count-space)+

```

```

lemma fk-estimate:
  assumes  $as \neq []$ 
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  assumes  $k \geq 1$ 
  shows  $\text{real } (\text{length } as) * \text{real-of-rat } (F (2*k-1) as) \leq \text{real } n^{\text{real } (1 - 1 / \text{real } k)} * (\text{real-of-rat } (F k as))^2$ 
  (is ?lhs  $\leq$  ?rhs)
proof (cases  $k \geq 2$ )
  case True
  define M where  $M = \text{Max } (\text{count-list } as \text{ 'set } as)$ 
  then obtain m where m-in:  $m \in \text{set } as$  and m-def:  $M = \text{count-list } as m$ 
  by (metis (mono-tags, lifting) List.finite-set Max-in finite-imageI image-iff image-is-empty set-empty assms(1))

```

```

  have a2:  $\text{real } M > 0$  apply (simp add: M-def)
  by (metis (mono-tags, opaque-lifting) List.finite-set assms(1) Max-in bot-nat-0.not-eq-extremum count-list-gr-1 finite-imageI imageE image-is-empty linorder-not-less set-empty zero-less-one)
  have a1:  $2*k-1 = (k-1) + k$  by simp
  have a4:  $(k-1) = k * ((k-1)/k)$  by simp

```

```

  have a3:  $M^k \leq \text{real-of-rat } (F k as)$ 
  apply (simp add: m-def F-def of-rat-sum of-rat-power)
  apply (subst powr-realpow, simp)
  using m-in count-list-gr-1 apply force
  by (rule member-le-sum, metis m-in, simp, simp)

```

```

  have a5:  $0 \leq \text{real-of-rat } (F k as)$ 
  using F-gr-0[OF assms(1)]
  by (simp add: order-le-less)
  hence a6:  $\text{real-of-rat } (F k as) = \text{real-of-rat } (F k as)^{\text{real } 1}$  by simp

```

```

  have  $\text{real } (k-1) / \text{real } k + 1 = \text{real } (k-1) / \text{real } k + \text{real } k / \text{real } k$ 
  using assms True by simp

```

```

also have ... = real (2 * k - 1) / real k
  apply (subst add-divide-distrib[symmetric])
  apply (rule arg-cong2[where f=(/)])
  apply (subst of-nat-diff) using True apply linarith
  apply (subst of-nat-diff) using True apply linarith
  by simp+
finally have a7: real (k - 1) / real k + 1 = real (2 * k - 1) / real k
  by blast

have a: real-of-rat (F (2*k-1) as) ≤ M powr (k-1) * (real-of-rat (F k as))
  using a1 apply (simp add:F-def of-rat-sum sum-distrib-left of-rat-mult power-add
of-rat-power)
  apply (rule sum-mono)
  apply (rule mult-right-mono)
  apply (subst powr-realpow)
  apply (metis a2)
  apply (subst power-mono)
  by (simp add:M-def)+
also have ... ≤ (real-of-rat (F k as)) powr ((k-1)/k) * (real-of-rat (F k as))
  apply (rule mult-right-mono)
  apply (subst a4)
  apply (subst powr-powr[symmetric])
  by (subst powr-mono2, simp, simp, metis a3, simp, metis a5)
also have ... = (real-of-rat (F k as)) powr ((2*k-1) / k)
  apply (subst (2) a6)
  apply (subst powr-add[symmetric])
  by (rule arg-cong2[where f=(powr)], simp, metis a7)
finally have a: real-of-rat (F (2*k-1) as) ≤ (real-of-rat (F k as)) powr ((2*k-1)
/ k)
  by blast

have b1: card (set as) ≤ n
  apply (rule card-mono[where B={k. k < n}, simplified])
  by (rule subsetI, simp add: assms(2))

have real (length as) = abs (sum (λx. real (count-list as x)) (set as))
  apply (subst of-nat-sum[symmetric])
  by (simp add: sum-count-set)
also have ... ≤ (real (card (set as))) powr ((k-Suc 0)/k) * (sum (λx. abs (real
(count-list as x)) powr k) (set as)) powr (1/k)
  apply (rule Holder-inequality-sum[where p=k/(k-1) and q=k and A=set as
and f=λ-.1, simplified])
  using assms True apply (simp)
  using assms True apply (simp)
  apply (subst add-divide-distrib[symmetric])
  using assms True by simp
also have ... ≤ real n powr (1 - 1 / real k) * real-of-rat (F k as) powr (1/real
k)
  apply (rule mult-mono)

```

```

    apply (subst of-nat-diff) using assms True apply linarith
    apply (subst diff-divide-distrib) using assms True apply simp
    apply (rule powr-mono2, force, simp)
  using b1 of-nat-le-iff apply blast
  apply (rule powr-mono2, force)
  apply (rule sum-mono[where f=λ-. 0, simplified])
  apply simp
  apply (simp add:F-def of-rat-sum of-rat-power)
  apply (rule sum-mono)
  apply (subst powr-realpow, simp)
  using count-list-gr-1
  by (metis gr0I not-one-le-zero, simp, simp, simp)
  finally have b: real (length as) ≤ real n powr (1 - 1 / real k) * real-of-rat (F
k as) powr (1 / real k)
  by blast

  have c: 1 / real k + real (2 * k - 1) / real k = real 2
  apply (subst add-divide-distrib[symmetric])
  apply (subst of-nat-diff) using True apply linarith
  using assms(2) True by simp

  have ?lhs ≤ real n powr (1 - 1 / real k) * real-of-rat (F k as) powr (1 / real k)
* (real-of-rat (F k as)) powr ((2*k-1) / k)
  apply (rule mult-mono, metis b, metis a, simp, simp add:F-def)
  apply (rule sum-mono[where f=λ-. (0::rat), simplified])
  by auto
  also have ... ≤ ?rhs
  apply (subst mult.assoc, subst powr-add[symmetric], subst mult-left-mono)
  apply (subst c, subst powr-realpow)
  using F-gr-0[OF assms(1)] by simp+
  finally show ?thesis
  by blast
next
case False
have n > 0
  apply (cases n=0)
  using assms(1) assms(2) equals0I apply (simp, blast)
  by simp
moreover have k = 1 using assms False by linarith
ultimately show ?thesis
  apply (simp add:power2-eq-square)
  apply (rule mult-right-mono)
  apply (simp add:F-def sum-count-set of-nat-sum[symmetric] del:of-nat-sum)
  using F-gr-0[OF assms(1)] order-le-less by auto
qed

lemma fk-alg-core-exp:
  assumes as ≠ []
  assumes k ≥ 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 -
  show ?thesis
    apply (subst has-bochner-integral-iff)
    apply (rule conjI)
    apply (rule integrable-measure-pmf-finite)
    apply (subst set-pmf-of-set, metis non-empty-space assms(1), metis fin-space
assms(1))
    apply (subst integral-measure-pmf-real[OF fin-space[OF assms(1)]])
    apply (subst (asm) set-pmf-of-set[OF non-empty-space[OF assms(1)] fin-space[OF
assms(1)]]], simp)
    apply (subst pmf-of-set[OF non-empty-space[OF assms(1)] fin-space[OF assms(1)]])
    using assms(1) apply (simp add: card-space F-def of-rat-sum of-rat-power)
    apply (subst split-space)
    apply (rule sum.cong, simp)
    apply (subst of-nat-diff)
    apply (simp add: power-mono)
    apply (subst sum-Suc-diff', simp, simp)
    using assms by linarith
qed

```

lemma *fk-alg-core-var*:

```

assumes as ≠ []
assumes k ≥ 1
assumes ∧a. a ∈ set as ⇒ a < n
shows prob-space.variance (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))2 * real k * real n powr (1 - 1 / real k)
proof -
  define f :: nat × nat ⇒ real
    where f = (λx. (real (length as) * real (Suc (snd x) ^ k - snd x ^ k)))
  define Ω where Ω = pmf-of-set {(u, v). v < count-list as u}

```

```

have integrable: ∧k f. integrable (measure-pmf Ω) (λω. (f ω)::real)
  apply (simp add: Ω-def)
  apply (rule integrable-measure-pmf-finite)
  apply (subst set-pmf-of-set)
  using assms(1) fin-space non-empty-space by auto

```

```

have k-g-0: k > 0 using assms by linarith

```

```

have c: ∧a v. v < count-list as a ⇒ real (Suc v ^ k) - real (v ^ k) ≤ real k *
real (count-list as a) ^ (k - Suc 0)
proof -
  fix a v

```

```

assume c-1:  $v < \text{count-list as } a$ 
have  $\text{real } (\text{Suc } v \wedge k) - \text{real } (v \wedge k) \leq (\text{real } (v+1) - \text{real } v) * \text{real } k * (1 + \text{real } v) \wedge (k - \text{Suc } 0)$ 
using k-g-0 power-diff-est[where  $a = \text{Suc } v$  and  $b = v$  and  $k = k$ ]
by simp
moreover have  $(\text{real } (v+1) - \text{real } v) = 1$  by auto
ultimately have  $\text{real } (\text{Suc } v \wedge k) - \text{real } (v \wedge k) \leq \text{real } k * (1 + \text{real } v) \wedge (k - \text{Suc } 0)$ 
by auto
also have  $\dots \leq \text{real } k * \text{real } (\text{count-list as } a) \wedge (k - \text{Suc } 0)$ 
apply (rule mult-left-mono, rule power-mono)
using c-1 apply linarith
by simp+
finally show  $\text{real } (\text{Suc } v \wedge k) - \text{real } (v \wedge k) \leq \text{real } k * \text{real } (\text{count-list as } a) \wedge (k - \text{Suc } 0)$ 
by blast
qed

```

```

have  $\text{real } (\text{length as}) * (\sum a \in \text{set as. } (\sum v \in \{0..< \text{count-list as } a\}. (\text{real } (\text{Suc } v \wedge k - v \wedge k))^2))$ 
 $\leq \text{real } (\text{length as}) * (\sum a \in \text{set as. } (\sum v \in \{0..< \text{count-list as } a\}. (\text{real } (k * \text{count-list as } a \wedge (k-1) * (\text{Suc } v \wedge k - v \wedge k))))))$ 
apply (rule mult-left-mono)
apply (rule sum-mono, rule sum-mono)
apply (simp add: power2-eq-square)
apply (rule mult-right-mono)
apply (subst of-nat-diff, simp add: power-mono)
by (metis c, simp, simp)
also have  $\dots = \text{real } (\text{length as}) * (\sum a \in \text{set as. } \text{real } (k * \text{count-list as } a \wedge (2*k-1)))$ 
apply (rule arg-cong2[where  $f = (*)$ ], simp)
apply (rule sum.cong, simp)
apply (simp add: sum-distrib-left[symmetric])
apply (subst of-nat-diff, rule power-mono, simp, simp)
apply (subst sum-Suc-diff', simp, simp add: zero-power[OF k-g-0] sum-distrib-left)
apply (subst power-add[symmetric])
using assms by (simp add: mult-2)
also have  $\dots = \text{real } (\text{length as}) * \text{real } k * \text{real-of-rat } (F (2*k-1) \text{ as})$ 
apply (subst mult.assoc)
apply (rule arg-cong2[where  $f = (*)$ ], simp)
by (simp add: sum-distrib-left[symmetric] F-def of-rat-sum of-rat-power)
also have  $\dots \leq \text{real } k * ((\text{real-of-rat } (F k \text{ as}))^2 * \text{real } n \text{ powr } (1 - 1 / \text{real } k))$ 
apply (subst mult.commute)
apply (subst mult.assoc)
apply (rule mult-left-mono)
using fk-estimate[OF assms(1) assms(3) assms(2)]
by (simp add: mult.commute, simp)
finally have  $b: \text{real } (\text{length as}) * (\sum a \in \text{set as. } (\sum v \in \{0..< \text{count-list as } a\}. (\text{real } (\text{Suc } v \wedge k - v \wedge k))^2))$ 

```

```

    ≤ real k * ((real-of-rat (F k as))2 * real n powr (1 - 1 / real k))
  by blast

  have measure-pmf.expectation Ω (λω. f ω2) - (measure-pmf.expectation Ω
f)2 ≤
    measure-pmf.expectation Ω (λω. f ω2)
  by simp
  also have measure-pmf.expectation Ω (λω. f ω2) ≤ (
    real-of-rat (F k as))2 * real k * real n powr (1 - 1 / real k)
  apply (simp add:Ω-def f-def)
  apply (subst integral-measure-pmf-real[OF fin-space[OF assms(1)]])
  apply (subst (asm) set-pmf-of-set[OF non-empty-space fin-space],metis assms(1),
simp)
  apply (subst pmf-of-set[OF non-empty-space fin-space],metis assms(1))
  apply (simp add:card-space[OF assms(1)] power-mult-distrib)
  apply (subst mult.commute, subst (2) power2-eq-square, subst split-space)
  using assms(1) by (simp add:algebra-simps sum-distrib-left[symmetric] b)
  finally have a:measure-pmf.expectation Ω (λω. f ω2) - (measure-pmf.expectation
Ω f)2 ≤
    (real-of-rat (F k as))2 * real k * real n powr (1 - 1 / real k)
  by blast

  show ?thesis
  apply (subst measure-pmf.variance-eq)
  apply (subst Ω-def[symmetric],metis integrable)
  apply (subst Ω-def[symmetric],metis integrable)
  apply (simp add:Ω-def[symmetric])
  using a f-def by simp
qed

theorem fk-alg-sketch:
  fixes ε :: rat
  assumes k ≥ 1
  assumes δ > 0
  assumes ⋀x. x ∈ set xs ⟹ x < n
  assumes xs ≠ []
  defines sketch ≡ fold (λx state. state ≫= fk-update x) xs (fk-init k δ ε n)
  defines s1 ≡ nat ⌈3*real k*(real n) powr (1-1/ real k)/ (real-of-rat δ)2⌋
  defines s2 ≡ nat ⌈-(18 * ln (real-of-rat ε))⌋
  shows sketch = map-pmf (λx. (s1,s2,k,length xs, x))
    (prod-pmf ({0..s1} × {0..s2}) (λ-. pmf-of-set {(u,v). v < count-list xs u}))
  apply (simp add:s sketch-def)
  using fk-alg-aux-1[OF assms(2) assms(3) assms(4), where k=k and ε=ε]
  apply (simp add:s1-def[symmetric] s2-def[symmetric])
  apply (rule arg-cong2[where f=map-pmf], simp)
  apply (subst fk-alg-aux-2[simplified], simp)
  apply (subst fk-alg-aux-4[OF assms(4), simplified], simp)
  by (subst fk-alg-aux-5[OF assms(4), simplified], simp)

```

```

lemma fk-alg-correct:
  assumes  $k \geq 1$ 
  assumes  $\varepsilon \in \{0 < \dots < 1\}$ 
  assumes  $\delta > 0$ 
  assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
  defines  $M \equiv \text{fold } (\lambda a \text{ state}. \text{state} \gg= \text{fk-update } a) \text{ as } (\text{fk-init } k \delta \varepsilon n) \gg= \text{fk-result}$ 
  shows  $\mathcal{P}(\omega \text{ in measure-pmf } M. |\omega - F \ k \ as| \leq \delta * F \ k \ as) \geq 1 - \text{of-rat } \varepsilon$ 
proof (cases as = [])
  case True
    have  $a: \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil > 0$  using assms by simp
    show ?thesis using True apply (simp add: F-def M-def bind-return-pmf median-const[OF a])
      using assms(2) by simp
  next
    case False
      define  $s_1$  where  $s_1 = \text{nat } \lceil 3 * \text{real } k * (\text{real } n) \text{ powr } (1 - 1 / \text{real } k) / (\text{real-of-rat } \delta)^2 \rceil$ 
      define  $s_2$  where  $s_2 = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 

      define  $f :: (\text{nat} \times \text{nat} \Rightarrow (\text{nat} \times \text{nat})) \Rightarrow \text{rat}$ 
      where  $f = (\lambda x. \text{median } (\lambda i_2 \in \{0..<s_2\}. (\sum i_1 = 0..<s_1. \text{rat-of-nat } (\text{length } as * (\text{Suc } (\text{snd } (x \ (i_1, i_2)))) \wedge k - \text{snd } (x \ (i_1, i_2)) \wedge k))) / \text{rat-of-nat } s_1)$ 

      define  $f2 :: (\text{nat} \times \text{nat} \Rightarrow (\text{nat} \times \text{nat})) \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{real})$ 
      where  $f2 = (\lambda x \ i_1 \ i_2. \text{real } (\text{length } as * (\text{Suc } (\text{snd } (x \ (i_1, i_2)))) \wedge k - \text{snd } (x \ (i_1, i_2)) \wedge k)))$ 
      define  $f1 :: (\text{nat} \times \text{nat} \Rightarrow (\text{nat} \times \text{nat})) \Rightarrow (\text{nat} \Rightarrow \text{real})$ 
      where  $f1 = (\lambda x \ i_2. (\sum i_1 = 0..<s_1. f2 \ x \ i_1 \ i_2) / \text{real } s_1)$ 
      define  $f' :: (\text{nat} \times \text{nat} \Rightarrow (\text{nat} \times \text{nat})) \Rightarrow \text{real}$ 
      where  $f' = (\lambda x. \text{median } (f1 \ x) \ s_2)$ 

      have  $\text{set } as \neq \{\}$  using assms False by blast
      hence  $n\text{-nonzero}: n > 0$  using assms(4) by fastforce

      have  $\text{fk-nonzero}: F \ k \ as > 0$  using F-gr-0 assms False by simp

      have  $s1\text{-nonzero}: s_1 > 0$ 
      apply (simp add: s1-def)
      apply (rule divide-pos-pos)
      apply (rule mult-pos-pos)
      using assms apply linarith
      apply (simp add: n-nonzero)
      by (meson assms zero-less-of-rat-iff zero-less-power)
      have  $s2\text{-nonzero}: s_2 > 0$  using assms by (simp add: s2-def)
      have  $\text{real-of-rat-f}: \bigwedge x. f' \ x = \text{real-of-rat } (f \ x)$ 

```



```

using s2-nonzero apply (simp add:f-def f'-def f1-def f2-def median-rat median-restrict)
apply (rule arg-cong2[where f=median])
by (simp add:of-rat-divide of-rat-sum of-rat-mult, simp)

define  $\Omega$  where  $\Omega = \text{pmf-of-set } \{(u, v). v < \text{count-list as } u\}$ 
have fin-omega: finite (set-pmf  $\Omega$ )
apply (subst  $\Omega$ -def, subst set-pmf-of-set)
using assms(5) fin-space non-empty-space False by auto
have fin-omega-2: finite (set-pmf ((prod-pmf ( $\{0..<s_1\} \times \{0..<s_2\}$ ) ( $\lambda-. \Omega$ ))))
apply (subst set-prod-pmf, simp)
apply (rule finite-PiE, simp)
by (simp add:fin-omega)

have a:fold ( $\lambda x \text{ state}. \text{state} \gg \text{fk-update } x$ ) as (fk-init k  $\delta \varepsilon n$ ) = map-pmf ( $\lambda x.$ 
( $s_1, s_2, k, \text{length as}, x$ )
(prod-pmf ( $\{0..<s_1\} \times \{0..<s_2\}$ ) ( $\lambda-. \text{pmf-of-set } \{(u, v). v < \text{count-list as } u\}$ )))
apply (subst fk-arg-sketch[OF assms(1) assms(3) assms(4) False], simp)
by (simp add:s1-def[symmetric] s2-def[symmetric])

have fk-result-exp: fk-result = ( $\lambda(x, y, z, u, v). \text{fk-result } (x, y, z, u, v)$ )
by (rule ext, fastforce)

have b:M = prod-pmf ( $\{0..<s_1\} \times \{0..<s_2\}$ ) ( $\lambda-. \Omega$ )  $\gg$  return-pmf  $\circ f$ 
apply (subst M-def)
apply (subst a)
apply (subst fk-result-exp, simp)
apply (simp add:map-pmf-def)
apply (subst bind-assoc-pmf)
apply (subst bind-return-pmf)
by (simp add:f-def comp-def  $\Omega$ -def)

have c:  $\{y. \text{real-of-rat } (\delta * F k \text{ as}) \geq |f' y - \text{real-of-rat } (F k \text{ as})|\} =$ 
 $\{y. (\delta * F k \text{ as}) \geq |f y - (F k \text{ as})|\}$ 
apply (simp add:real-of-rat-f)
by (metis abs-of-rat of-rat-diff of-rat-less-eq)

have f2-exp:  $\bigwedge i_1 i_2. i_1 < s_1 \implies i_2 < s_2 \implies$ 
has-bochner-integral (measure-pmf (prod-pmf ( $\{0..<s_1\} \times \{0..<s_2\}$ ) ( $\lambda-. \Omega$ )))
( $\lambda x. f2 \ x \ i_1 \ i_2$ )
(real-of-rat (F k as))
apply (simp add:f2-def  $\Omega$ -def of-rat-mult of-rat-sum of-rat-power)
apply (rule has-bochner-integral-prod-pmf-sliceI, simp, simp)
by (rule fk-arg-core-exp, metis False, metis assms(1))

have  $3 * \text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k) = (\text{real-of-rat } \delta)^2 * (3 * \text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k) / (\text{real-of-rat } \delta)^2)$ 
using assms by simp
also have  $\dots \leq (\text{real-of-rat } \delta)^2 * (\text{real } s_1)$ 

```

```

    apply (rule mult-mono, simp)
    apply (simp add:s1-def)
    apply (meson of-nat-ceiling)
    using assms apply simp
  by simp
  finally have f2-var-2:  $3 * \text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k) \leq (\text{real-of-rat } \delta)^2 * (\text{real } s_1)$ 
  by blast
  have  $(\text{real-of-rat } (F \ k \ as))^2 * \text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k) = (\text{real-of-rat } (F \ k \ as))^2 * (\text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k))$ 
  by (simp add:ac-simps)
  also have  $\dots \leq (\text{real-of-rat } (F \ k \ as * \delta))^2 * (\text{real } s_1 / 3)$ 
  apply (subst of-rat-mult, subst power-mult-distrib)
  apply (subst mult.assoc[where c=real s1 / 3])
  apply (rule mult-mono, simp) using f2-var-2
  by (simp+)
  finally have f2-var-1:  $(\text{real-of-rat } (F \ k \ as))^2 * \text{real } k * \text{real } n \text{ powr } (1 - 1 / \text{real } k) \leq (\text{real-of-rat } (\delta * F \ k \ as))^2 * \text{real } s_1 / 3$ 
  by (simp add: mult.commute)

  have f2-var:  $\bigwedge i_1 \ i_2. i_1 < s_1 \implies i_2 < s_2 \implies \text{prob-space.variance } (\text{measure-pmf } (\text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) (\lambda-. \Omega))) (\lambda\omega. f2 \ \omega \ i_1 \ i_2) \leq (\text{real-of-rat } (\delta * F \ k \ as))^2 * \text{real } s_1 / 3$ 
  apply (simp only: f2-def)
  apply (subst variance-prod-pmf-slice, simp, simp, rule integrable-measure-pmf-finite[OF fin-omega])
  apply (rule order-trans [where y=(real-of-rat (F k as))^2 * real k * real n powr (1 - 1 / real k)])
  apply (simp add:  $\Omega$ -def)
  using assms False fk-alg-core-var[where k=k] apply simp
  using f2-var-1 by blast

  have f1-exp-1:  $(\text{real-of-rat } (F \ k \ as)) = (\sum i \in \{0..<s_1\}. (\text{real-of-rat } (F \ k \ as)) / \text{real } s_1)$ 
  by (simp add:s1-nonzero)

  have f1-exp:  $\bigwedge i. i < s_2 \implies \text{has-bochner-integral } (\text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) (\lambda-. \Omega)) (\lambda\omega. f1 \ \omega \ i) (\text{real-of-rat } (F \ k \ as))$ 
  apply (simp add:f1-def sum-divide-distrib)
  apply (subst f1-exp-1)
  apply (rule has-bochner-integral-sum)
  apply (rule has-bochner-integral-divide-zero)
  by (simp add: f2-exp)

  have f1-var:  $\bigwedge i. i < s_2 \implies \text{prob-space.variance } (\text{prod-pmf } (\{0..<s_1\} \times \{0..<s_2\}) (\lambda-. \Omega)) (\lambda\omega. f1 \ \omega \ i) \leq \text{real-of-rat } (\delta * F \ k \ as)^2 / 3$  (is  $\bigwedge i. - \implies ?rhs \ i$ )

```

```

proof -
  fix i
  assume f1-var-1:i < s2
  have prob-space.variance (prod-pmf ({0..s1} × {0..s2}) (λ-. Ω)) (λω. f1 ω
i) =
    (∑ j = 0..s1. prob-space.variance (prod-pmf ({0..s1} × {0..s2}) (λ-.
Ω)) (λω. f2 ω j i / real s1))
  apply (simp add:f1-def sum-divide-distrib)
  apply (subst measure-pmf.var-sum-all-indep, simp, simp)
  apply (rule integrable-measure-pmf-finite[OF fin-omega-2])
  apply (rule indep-vars-restrict-intro[where f=λj. {j} × {i}])
    apply (simp add:f2-def)
    apply (simp add:disjoint-family-on-def)
    apply (simp add:s1-nonzero)
    apply (simp add:f1-var-1)
  apply simp
  apply simp
  by simp
  also have ... = (∑ j = 0..s1. prob-space.variance (prod-pmf ({0..s1} ×
{0..s2}) (λ-. Ω)) (λω. f2 ω j i) / real s1 ^2)
  apply (rule sum.cong, simp)
  apply (rule measure-pmf.variance-divide)
  by (rule integrable-measure-pmf-finite[OF fin-omega-2])
  also have ... ≤ (∑ j = 0..s1. ((real-of-rat (δ * F k as))^2 * real s1 / 3) / (real
s1 ^2))
  apply (rule sum-mono)
  apply (rule divide-right-mono)
  apply (rule f2-var[OF f1-var-1], simp)
  by simp
  also have ... = real-of-rat (δ * F k as) ^2 / 3
  apply simp
  apply (subst nonzero-divide-eq-eq, simp add:s1-nonzero)
  by (simp add:power2-eq-square)
  finally show ?rhs i by simp
qed

have d: ∧i. i < s2 ⇒ measure-pmf.prob (prod-pmf ({0..s1} × {0..s2}) (λ-.
Ω))
  {y. real-of-rat (δ * F k as) < |f1 y i - real-of-rat (F k as)|} ≤ 1/3 (is ∧i. - ⇒
?lhs i ≤ -)
proof -
  fix i
  assume d-1:i < s2
  define a where a = real-of-rat (δ * F k as)
  have d-2: 0 < a apply (simp add:a-def)
  using assms fk-nonzero mult-pos-pos by blast
  have d-3: integrable (measure-pmf (prod-pmf ({0..s1} × {0..s2}) (λ-. Ω)))
(λx. (f1 x i)^2)
  by (rule integrable-measure-pmf-finite[OF fin-omega-2])

```

```

have ?lhs i ≤ measure-pmf.prob (prod-pmf ({0..<s1} × {0..<s2})) (λ-. Ω))
  {y. real-of-rat (δ * F k as) ≤ |f1 y i - real-of-rat (F k as)|}
  by (rule pmf-mono-1, simp)
also have ... ≤ prob-space.variance (prod-pmf ({0..<s1} × {0..<s2})) (λ-. Ω))
(λω. f1 ω i)/a^2
  using f1-exp[OF d-1]
  using prob-space.Chebyshev-inequality[OF prob-space-measure-pmf - d-3 d-2,
simplified]
  by (simp add:a-def[symmetric] has-bochner-integral-iff)
also have ... ≤ 1/3 using d-2
  using f1-var[OF d-1]
  by (simp add:algebra-simps, simp add:a-def)
finally show ?lhs i ≤ 1/3
  by blast
qed

show ?thesis
  apply (simp add: b comp-def map-pmf-def[symmetric])
  apply (subst c[symmetric])
  apply (simp add:f'-def)
  apply (rule prob-space.median-bound-2[where X=λi ω. f1 ω i and M=(prod-pmf
({0..<s1} × {0..<s2})) (λ-. Ω)), simplified])
    apply (simp add:prob-space-measure-pmf)
    using assms(2) apply simp
  using assms(2) apply simp
  apply (simp add:f1-def f2-def)
  apply (rule indep-vars-restrict-intro[where f=λi. ({0..<s1} × {i})])
    apply (simp)
    apply (simp add:disjoint-family-on-def, blast)
    apply (simp add:s2-nonzero)
    apply (rule subsetI, simp, force)
    apply (simp)
    apply (simp)
  apply (simp add: s2-def)
    using of-nat-ceiling apply blast
    using d by simp
qed

fun fk-space-usage :: (nat × nat × nat × rat × rat) ⇒ real where
fk-space-usage (k, n, m, ε, δ) = (
  let s1 = nat ⌈3*real k*(real n) powr (1-1/ real k) / (real-of-rat δ)2 ⌋ in
  let s2 = nat ⌈-(18 * ln (real-of-rat ε))⌋ in
  5 +
  2 * log 2 (s1 + 1) +
  2 * log 2 (s2 + 1) +
  2 * log 2 (real k + 1) +
  2 * log 2 (real m + 1) +
  s1 * s2 * (3 + 2 * log 2 (real n) + 2 * log 2 (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*)

```

apply (rule encoding-imp-inj)
apply (simp add:encode-state-def)
apply (rule dependent-encoding, metis nat-encoding)
apply (rule dependent-encoding, metis nat-encoding)
apply (rule prod-encoding, metis nat-encoding)
apply (rule prod-encoding, metis nat-encoding)
by (metis encode-extensional prod-encoding nat-encoding)

```

theorem *fk-exact-space-usage*:

```

assumes  $k \geq 1$ 
assumes  $\varepsilon \in \{0..<1\}$ 
assumes  $\delta > 0$ 
assumes  $\bigwedge a. a \in \text{set } as \implies a < n$ 
assumes  $as \neq []$ 
defines  $M \equiv \text{fold } (\lambda a \text{ state. state } \gg= \text{fk-update } a) \text{ as } (\text{fk-init } k \delta \varepsilon n)$ 
shows  $AE \omega \text{ in } M. \text{bit-count } (\text{encode-state } \omega) \leq \text{fk-space-usage } (k, n, \text{length } as,$ 
 $\varepsilon, \delta) \text{ (is } AE \omega \text{ in } M. (- \leq ?rhs))$ 
proof –
  define  $s_1$  where  $s_1 = \text{nat } \lceil 3 * \text{real } k * (\text{real } n) \text{ powr } (1 - 1 / \text{real } k) / (\text{real-of-rat } \delta)^2 \rceil$ 
  define  $s_2$  where  $s_2 = \text{nat } \lceil -(18 * \ln (\text{real-of-rat } \varepsilon)) \rceil$ 

```

```

have  $a:M = \text{map-pmf } (\lambda x. (s_1, s_2, k, \text{length } as, x))$ 
  (prod-pmf ( $\{0..<s_1\} \times \{0..<s_2\}$ ) ( $\lambda-. \text{pmf-of-set } \{(u, v). v < \text{count-list } as \text{ at } u\}$ ))
apply (subst M-def)
apply (subst fk-alg-sketch[OF assms(1) assms(3) assms(4) assms(5)], simp)
by (simp add:s1-def[symmetric] s2-def[symmetric])

```

```

have  $\text{set } as \neq \{\}$  using assms by blast
hence  $n\text{-nonzero}: n > 0$  using assms(4) by fastforce
have  $\text{length-}xs\text{-gr-}0: \text{length } as > 0$  using assms(5) by blast

```

```

have  $b:\bigwedge y. y \in \{0..<s_1\} \times \{0..<s_2\} \rightarrow_E \{(u, v). v < \text{count-list } as \text{ at } u\} \implies$ 
   $\text{bit-count } (\text{encode-state } (s_1, s_2, k, \text{length } as, y)) \leq ?rhs$ 

```

proof –

```

  fix  $y$ 
  assume  $b0:y \in \{0..<s_1\} \times \{0..<s_2\} \rightarrow_E \{(u, v). v < \text{count-list } as \text{ at } u\}$ 
  have  $\bigwedge x. x \in y ' (\{0..<s_1\} \times \{0..<s_2\}) \implies 1 \leq \text{count-list } as \text{ (fst } x)$ 
    using  $b0$  by (simp add:PiE-iff case-prod-beta, fastforce)
  hence  $b1:\bigwedge x. x \in y ' (\{0..<s_1\} \times \{0..<s_2\}) \implies \text{fst } x \leq n - \text{Suc } 0$  using

```

```

assms(4)
  apply (simp add: count-list-gr-1[simplified, symmetric])
  by (metis Suc-pred less-Suc-eq-le n-nonzero)
have b2:  $\bigwedge x. x \in y \rightarrow (\{0..<s_1\} \times \{0..<s_2\}) \implies \text{snd } x \leq \text{length } as - \text{Suc } 0$ 
  using count-le-length b0 apply (simp add: PiE-iff case-prod-beta)
  using dual-order.strict-trans1 by fastforce
have b3:  $y \in \text{extensional } (\{0..<s_1\} \times \{0..<s_2\})$  using b0 PiE-iff by blast
hence bit-count (encode-state (s1, s2, k, length as, y)) ≤
  ereal (2 * log 2 (real s1 + 1) + 1) + (
  ereal (2 * log 2 (real s2 + 1) + 1) + (
  ereal (2 * log 2 (real k + 1) + 1) + (
  ereal (2 * log 2 (real (length as) + 1) + 1) + (
  (ereal (real s1 * real s2) * ((ereal (2 * log 2 ((n-1)+1) + 1) + 1) + ereal (2 *
log 2 ((length as-1)+1) + 1) + 1))) + 1)))
  apply (simp add: encode-state-def dependent-bit-count prod-bit-count PiE-iff
comp-def encode-extensional-def
del: NS.simps encode-prod.simps encode-dependent-sum.simps plus-ereal.simps
sum-list-ereal times-ereal.simps)
  apply (rule add-mono, simp add: nat-bit-count[simplified])
  apply (rule add-mono, simp add: nat-bit-count[simplified])
  apply (rule add-mono, simp add: nat-bit-count[simplified])
  apply (rule add-mono, simp add: nat-bit-count[simplified])
  apply (rule list-bit-count-est[where xs=map y (List.product [0..<s1] [0..<s2]),
simplified])
  apply (subst prod-bit-count-2)
  apply (rule add-mono)
  apply (rule nat-bit-count-est, metis b1)
  by (rule nat-bit-count-est, metis b2)
also have ... ≤ ?rhs
using n-nonzero length-xs-gr-0 apply (simp add: s1-def[symmetric] s2-def[symmetric,simplified])
  by (rule mult-left-mono, simp, simp)
finally show bit-count (encode-state (s1, s2, k, length as, y)) ≤ ?rhs
  by blast
qed

show ?thesis
  apply (simp add: a AE-measure-pmf-iff del:fk-space-usage.simps)
  apply (subst set-prod-pmf, simp, simp add: PiE-def del:fk-space-usage.simps)
  apply (subst set-pmf-of-set [OF non-empty-space[OF assms(5)] fin-space[OF
assms(5)]]])
  apply (subst PiE-def[symmetric])
  by (metis b)
qed

lemma fk-asymptotic-space-complexity:
  fk-space-usage ∈
  O[at-top ×F at-top ×F at-top ×F at-right (0::rat) ×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 - ∈ O[?F](?rhs))
proof -
  define k-of :: nat × nat × nat × rat × rat ⇒ nat where k-of = (λ(k, n, m, ε,
δ). k)
  define n-of :: nat × nat × nat × rat × rat ⇒ nat where n-of = (λ(k, n, m, ε,
δ). n)
  define m-of :: nat × nat × nat × rat × rat ⇒ nat where m-of = (λ(k, n, m,
ε, δ). m)
  define ε-of :: nat × nat × nat × rat × rat ⇒ rat where ε-of = (λ(k, n, m, ε,
δ). ε)
  define δ-of :: nat × nat × nat × rat × rat ⇒ rat where δ-of = (λ(k, n, m, ε,
δ). δ)

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

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

have k-inf: ∧c. eventually (λx. c ≤ (real (k-of x))) ?F
  apply (simp add:k-of-def case-prod-beta')
  apply (subst eventually-prod1', simp add:prod-filter-eq-bot)
  by (meson eventually-at-top-linorder nat-ceiling-le-eq)

have n-inf: ∧c. eventually (λx. c ≤ (real (n-of x))) ?F
  apply (simp add:n-of-def case-prod-beta')
  apply (subst eventually-prod2', simp add:prod-filter-eq-bot)
  apply (subst eventually-prod1', simp add:prod-filter-eq-bot)
  by (meson eventually-at-top-linorder nat-ceiling-le-eq)

have m-inf: ∧c. eventually (λx. c ≤ (real (m-of x))) ?F
  apply (simp add:m-of-def case-prod-beta')
  apply (subst eventually-prod2', simp add:prod-filter-eq-bot)
  apply (subst eventually-prod2', simp add:prod-filter-eq-bot)
  apply (subst eventually-prod1', simp add:prod-filter-eq-bot)
  by (meson eventually-at-top-linorder nat-ceiling-le-eq)

have eps-inf: ∧c. eventually (λx. c ≤ 1 / (real-of-rat (ε-of x))) ?F
  apply (simp add:ε-of-def case-prod-beta')
  apply (subst eventually-prod2', simp add:prod-filter-eq-bot)
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod2', simp)
  apply (subst eventually-prod1', simp)
  by (rule inv-at-right-0-inf)

have delta-inf: ∧c. eventually (λx. c ≤ 1 / (real-of-rat (δ-of x))) ?F
  apply (simp add:δ-of-def case-prod-beta')

```

```

apply (subst eventually-prod2', simp add:prod-filter-eq-bot)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
by (rule inv-at-right-0-inf)

have zero-less-eps: eventually ( $\lambda x. 0 < (\text{real-of-rat } (\varepsilon\text{-of } x))$ ) ?F
apply (simp add: $\varepsilon$ -of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod1', simp)
by (rule eventually-at-rightI[where  $b=1$ ], simp, simp)

have zero-less-delta: eventually ( $\lambda x. 0 < (\text{real-of-rat } (\delta\text{-of } x))$ ) ?F
apply (simp add: $\delta$ -of-def case-prod-beta')
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
apply (subst eventually-prod2', simp)
by (rule eventually-at-rightI[where  $b=1$ ], simp, simp)

have unit-9: ( $\lambda-. 1 \in O[?F](\lambda x. \text{real } (n\text{-of } x) \text{ powr } (1 - 1 / \text{real } (k\text{-of } x)))$ )
apply (rule landau-o.big-mono, simp)
apply (rule eventually-mono[OF eventually-conj[OF  $n\text{-inf}$ [where  $c=1$ ]  $k\text{-inf}$ [where
 $c=1$ ]]])
by (simp add: ge-one-powr-ge-zero)

have unit-8: ( $\lambda-. 1 \in O[?F](\lambda x. \text{real } (k\text{-of } x))$ )
by (rule landau-o.big-mono, simp, rule  $k\text{-inf}$ )
have unit-6: ( $\lambda-. 1 \in O[?F](\lambda x. \text{real } (m\text{-of } x))$ )
by (rule landau-o.big-mono, simp, rule  $m\text{-inf}$ )

have unit-2: ( $\lambda-. 1 \in O[?F](\lambda x. \ln (1 / \text{real-of-rat } (\varepsilon\text{-of } x)))$ )
apply (rule landau-o.big-mono, simp)
apply (rule eventually-mono[OF eventually-conj[OF zero-less-eps  $\text{eps-inf}$ [where
 $c=\exp 1$ ]]])
by (meson abs-ge-self dual-order.trans exp-gt-zero  $\ln\text{-ge-iff}$  order-trans-rules(22))

have unit-10: ( $\lambda-. 1 \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)))$ )
apply (rule landau-o.big-mono, simp)
apply (rule eventually-mono [OF  $n\text{-inf}$ [where  $c=\exp 1$ ]])
by (metis abs-ge-self linorder-not-le  $\ln\text{-ge-iff}$  not-exp-le-zero order.trans)

have unit-3: ( $\lambda x. 1 \in O[?F](\lambda x. \ln (\text{real } (n\text{-of } x)) + \ln (\text{real } (m\text{-of } x)))$ )
apply (rule landau-sum-1)
apply (rule eventually- $\ln\text{-ge-iff}$ [OF  $n\text{-inf}$ ])
apply (rule eventually- $\ln\text{-ge-iff}$ [OF  $m\text{-inf}$ ])
by (rule unit-10)

```



```

have unit-7: (λ-. 1) ∈ O[?F](λx. 1 / (real-of-rat (δ-of x))2)
  apply (rule landau-o.big-mono, simp)
  apply (rule eventually-mono[OF eventually-conj[OF zero-less-delta delta-inf[where
c=1]]]])
  by (metis one-le-power power-one-over)

have unit-4: (λ-. 1) ∈ O[?F](g1)
  apply (simp add:g1-def)
  apply (subst (2) div-commute)
  apply (rule landau-o.big-mult-1[OF unit-7])
  by (rule landau-o.big-mult-1[OF unit-8 unit-9])

have unit-5: (λ-. 1) ∈ O[?F](λx. g1 x * ln (1 / real-of-rat (ε-of x)))
  by (rule landau-o.big-mult-1[OF unit-4 unit-2])

have unit-1: (λ-. 1) ∈ O[?F](g)
  apply (simp add:g-def)
  by (rule landau-o.big-mult-1[OF unit-5 unit-3])

have l6: (λx. real (nat ⌈3 * real (k-of x) * real (n-of x) powr (1 - 1 / real (k-of
x)) / (real-of-rat (δ-of x))2⌋))
  ∈ O[?F](g1)
  apply (rule landau-nat-ceil[OF unit-4])
  apply (simp add:g1-def)
  apply (subst (2) div-commute, subst (4) div-commute)
  apply (rule landau-o.mult, simp)
  by simp

have l9: (λx. real (nat ⌈- (18 * ln (real-of-rat (ε-of x)))⌋))
  ∈ O[?F](λx. ln (1 / real-of-rat (ε-of x)))
  apply (rule landau-nat-ceil[OF unit-2])
  apply (subst minus-mult-right)
  apply (subst cmult-in-bigo-iff, rule disjI2)
  apply (subst landau-o.big.in-cong[where g=λx. ln (1 / (real-of-rat (ε-of x)))])
  apply (rule eventually-mono[OF zero-less-eps])
  by (subst ln-div, simp, simp, simp, simp)

have l1: (λx. real (nat ⌈3 * real (k-of x) * real (n-of x) powr (1 - 1 / real (k-of
x)) / (real-of-rat (δ-of x))2⌋) *
  real (nat ⌈- (18 * ln (real-of-rat (ε-of x)))⌋) *
  (3 + 2 * log 2 (real (n-of x)) + 2 * log 2 (real (m-of x)))) ∈ O[?F](g)
  apply (simp add:g-def)
  apply (rule landau-o.mult)
  apply (rule landau-o.mult, simp add:l6, simp add:l9)
  apply (rule sum-in-bigo)
  apply (rule sum-in-bigo, simp add:unit-3)
  apply (simp add:log-def)
  apply (rule landau-sum-1 [OF eventually-ln-ge-iff[OF n-inf] eventually-ln-ge-iff[OF

```

```

m-inf]], simp)
  apply (simp add:log-def)
  by (rule landau-sum-2 [OF eventually-ln-ge-iff[OF n-inf] eventually-ln-ge-iff[OF
m-inf]], simp)

have l2: ( $\lambda x. \ln (\text{real } (m\text{-of } x) + 1) \in O[?F](g)$ )
  apply (simp add:g-def)
  apply (rule landau-o.big-mult-1 '[OF unit-5])
  apply (rule landau-sum-2 [OF eventually-ln-ge-iff[OF n-inf] eventually-ln-ge-iff[OF
m-inf]])
  apply (rule landau-ln-2[where a=2], simp, simp, rule m-inf)
  by (rule sum-in-bigo, simp, rule unit-6)

have l7: ( $\lambda x. \ln (\text{real } (k\text{-of } x) + 1) \in O[?F](g1)$ )
  apply (simp add:g1-def)
  apply (subst (2) div-commute)
  apply (rule landau-o.big-mult-1 '[OF unit-7])
  apply (rule landau-o.big-mult-1)
  apply (rule landau-ln-3, simp)
  by (rule sum-in-bigo, simp, simp add:unit-8, simp add: unit-9)

have l3: ( $\lambda x. \ln (\text{real } (k\text{-of } x) + 1) \in O[?F](g)$ )
  apply (simp add:g-def)
  apply (rule landau-o.big-mult-1)
  apply (rule landau-o.big-mult-1)
  apply (simp add:l7)
  by (rule unit-2, rule unit-3)

have l4: ( $\lambda x. \ln (\text{real } (\text{nat } \lceil - (18 * \ln (\text{real-of-rat } (\varepsilon\text{-of } x))) \rceil) + 1) \in O[?F](g)$ )
  apply (simp add:g-def)
  apply (rule landau-o.big-mult-1)
  apply (rule landau-o.big-mult-1 '[OF unit-4])
  apply (rule landau-ln-3, simp)
  by (rule sum-in-bigo, simp add:l9, rule unit-2, rule unit-3)

have l5: ( $\lambda x. \ln (\text{real } (\text{nat } \lceil 3 * \text{real } (k\text{-of } x) * \text{real } (n\text{-of } x) \text{ powr } (1 - 1 / \text{real } (k\text{-of } x)) / (\text{real-of-rat } (\delta\text{-of } x))^2 \rceil) + 1) \in O[?F](g)$ )
  apply (rule landau-ln-3, simp)
  apply (rule sum-in-bigo)
  apply (simp add:g-def)
  apply (rule landau-o.big-mult-1)
  apply (rule landau-o.big-mult-1)
  apply (simp add:l6)
  by (rule unit-2, rule unit-3, rule unit-1)

have fk-space-usage = ( $\lambda x. \text{fk-space-usage } (k\text{-of } x, n\text{-of } x, m\text{-of } x, \varepsilon\text{-of } x, \delta\text{-of } x)$ )
  apply (rule ext)
  by (simp add:case-prod-beta' k-of-def n-of-def  $\varepsilon$ -of-def  $\delta$ -of-def m-of-def)

```

```

also have ...  $\in O[?F](g)$ 
  apply (simp add: Let-def)
  apply (rule sum-in-bigo-r, simp add:l1)
  apply (rule sum-in-bigo-r, simp add:l2 log-def)
  apply (rule sum-in-bigo-r, simp add:l3 log-def)
  apply (rule sum-in-bigo-r, simp add:l4 log-def)
  apply (rule sum-in-bigo-r, simp add:l4 log-def)
  by (simp add:l5, simp add:unit-1)
also have ...  $= O[?F](?rhs)$ 
  apply (rule arg-cong2[where  $f=bigo$ ], simp)
  apply (rule ext)
  by (simp add:case-prod-beta' g1-def g-def n-of-def  $\varepsilon$ -of-def  $\delta$ -of-def m-of-def
k-of-def)
  finally show ?thesis by simp
qed

end

```

A Informal proof of correctness for the F_0 algorithm

This 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, \dots, a_m be the stream elements, where we assume that the elements are a subset of $\{0, \dots, n-1\}$ 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 algorithm 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:

$$\begin{aligned}
 A &:= \{a_1, \dots, a_m\} & H &:= \{\lfloor h(a) \rfloor_r \mid a \in A\} \\
 R &:= \begin{cases} tp(\min_t(H))^{-1} & \text{if } |H| \geq t \\ |H| & \text{otherwise,} \end{cases}
 \end{aligned}$$

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:

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

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:

$$\delta' := \frac{3}{4}\delta \quad (1)$$

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

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

I.e. the probability that R^* has not the relative accuracy of $\frac{3}{4}\delta$ is less than $\frac{2}{9}$ and the probability that assuming R^* has the relative accuracy of $\frac{3}{4}\delta$ but that R deviates by more than $\frac{1}{4}\delta F_0$ 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:

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

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 linearity of expectation and Bienaymé's identity, it follows that $\text{Var } Q(u) \leq \mathbb{E} Q(u) = |A|up^{-1} = F_0up^{-1}$ for $u \in \{0, \dots, p\}$.

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

$$\begin{aligned} t - 1 &\leq^3 \frac{t}{(1-\delta')} - 3\sqrt{\frac{t}{(1-\delta')}} - 1 \\ &\leq \frac{F_0v}{p} - 3\sqrt{\frac{F_0v}{p}} \leq \mathbb{E}Q(v) - 3\sqrt{\text{Var}Q(v)} \end{aligned}$$

and thus using Tchebyshev's inequality:

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

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

$$\begin{aligned} t &\geq \frac{t}{(1+\delta')} + 3\sqrt{\frac{t}{(1+\delta')}} + 1 + 1 \\ &\geq \frac{F_0u}{p} + 3\sqrt{\frac{F_0u}{p}} \geq \mathbb{E}Q(u) + 3\sqrt{\text{Var}Q(u)} \end{aligned}$$

and thus using Tchebyshev's inequality:

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

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

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

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

To verify Equation 3, note that

$$\min_t(H) = \lfloor \min_t^\#(H^*) \rfloor_r \quad (7)$$

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 chosen uniformly from the polynomials of degree less than 2. If it is a degree 1 polynomial, it is a linear function on $GF(p)$ and thus injective. Because $p \geq 18$ the probability that h is not injective can be bounded by $1/18$. 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.

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

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

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

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:

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

Which concludes the proof. \square

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.