Universal Hash Families

Emin Karayel

March 17, 2025

Abstract

A k-universal hash family is a probability space of functions, which have uniform distribution and form k-wise independent random variables.

They can often be used in place of classic (or cryptographic) hash functions and allow the rigorous analysis of the performance of randomized algorithms and data structures that rely on hash functions.

In 1981 Wegman and Carter [4] introduced a generic construction for such families with arbitrary k using polynomials over a finite field. This entry contains a formalization of them and establishes the property of k-universality.

To be useful the formalization also provides an explicit construction of finite fields using the factor ring of integers modulo a prime. Additionally, some generic results about independent families are shown that might be of independent interest.

1 Introduction and Definition

theory *Universal-Hash-Families*
  imports *HOL−Probability.Independent-Family*
begin

Universal hash families are commonly used in randomized algorithms and data structures to randomize the input of algorithms, such that probabilistic methods can be employed without requiring any assumptions about the input distribution.

If we regard a family of hash functions from a domain D to a finite range R as a uniform probability space, then the family is k-universal if:

- For each $x \in D$ the evaluation of the functions at x forms a uniformly distributed random variable on R.
- The evaluation random variables for k or fewer distinct domain elements form an independent family of random variables.

This definition closely follows the definition from Vadhan [3, §3.5.5], with the minor modification that independence is required not only for exactly k, but also for *fewer* than k distinct domain elements. The correction is due to the fact that in the corner case where D has fewer than k elements, the second part of their definition becomes void. In the formalization this helps avoid an unnecessary assumption in the theorems.

The following definition introduces the notion of k-wise independent random variables:

definition (**in** *prob-space*) *k-wise-indep-vars* **where**
k-wise-indep-vars k M′ X I =
($\forall$ $J \subseteq I$. *card* $J \leq k \longrightarrow$ *finite* $J \longrightarrow$ *indep-vars* M' X J)

lemma (**in** *prob-space*) *k-wise-indep-vars-subset*:
assumes *k-wise-indep-vars k M′ X I*
assumes $J \subseteq I$
assumes *finite J*
assumes *card* $J \leq k$
shows *indep-vars M′ X J*
⟨*proof*⟩

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

Similarly for a finite non-empty set A the predicate *uniform-on X A* indicates that the random variable is uniformly distributed on A:

definition (**in** *prob-space*) *uniform-on X A* = (
distr M (*count-space UNIV*) *X* = *uniform-measure* (*count-space UNIV*) *A* $\wedge$
$A \neq \{\}$ $\wedge$ *finite A* $\wedge$ *random-variable* (*count-space UNIV*) *X*)

lemma (**in** *prob-space*) *uniform-onD*:
assumes *uniform-on X A*
shows *prob* $\{\omega \in$ *space M*. $X\ \omega \in B\}$ = *card* $(A \cap B)$ / *card A*
⟨*proof*⟩

With the two previous definitions it is possible to define the k-universality condition for a family of hash functions from D to R:

definition (**in** *prob-space*) *k-universal k X D R* = (
k-wise-indep-vars k (λ-. *count-space UNIV*) *X D* $\wedge$
($\forall$ $i \in D$. *uniform-on* (*X i*) *R*))

Note: The definition is slightly more generic then the informal specification from above. This is because usually a family is formed by a single function with a variable seed parameter. Instead of choosing a random function from a probability space, a random seed is chosen from the probability space which parameterizes the hash function.

The following section contains some preliminary results about independent families of random variables. Section 3 introduces the Carter-Wegman hash family, which is an explicit construction of k-universal families for arbitrary k using polynomials over finite fields. The last section contains a proof that the factor ring of the integers modulo a prime ideal is a finite field, followed by an isomorphic construction of prime fields over an initial segment of the natural numbers.

end

2 Preliminary Results

theory *Universal-Hash-Families-More-Independent-Families*
imports
Universal-Hash-Families
HOL−Probability.Stream-Space
HOL−Probability.Probability-Mass-Function
begin

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

lemma (**in** *prob-space*) *k-wise-indep-vars-compose*:
assumes *k-wise-indep-vars* $k\ M'\ X\ I$
assumes $\bigwedge i.\ i \in I \Longrightarrow Y\ i \in$ *measurable* $(M'\ i)\ (N\ i)$
shows *k-wise-indep-vars* $k\ N\ (\lambda i\ x.\ Y\ i\ (X\ i\ x))\ I$
⟨*proof*⟩

lemma (**in** *prob-space*) *k-wise-indep-vars-triv*:
assumes *indep-vars* $N\ T\ I$
shows *k-wise-indep-vars* $k\ N\ T\ I$
⟨*proof*⟩

The following two lemmas are of independent interest, they help infer independence of events and random variables on distributions. (Candidates for *HOL−Probability.Independent-Family*).

lemma (**in** *prob-space*) *indep-sets-distr*:
fixes A
assumes *random-variable* $N\ f$
defines $F \equiv (\lambda i.\ (\lambda a.\ f\ -`\ a \cap \textit{space}\ M)\ `\ A\ i)$
assumes *indep-F*: *indep-sets* $F\ I$
assumes *sets-A*: $\bigwedge i.\ i \in I \Longrightarrow A\ i \subseteq$ *sets* N
shows *prob-space.indep-sets* (*distr* $M\ N\ f$) $A\ I$
⟨*proof*⟩

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

lemma *range-inter*: *range* $((\cap)\ F)$ = *Pow* F
  ⟨*proof*⟩

The singletons and the empty set form an intersection stable generator of a countable discrete σ-algebra:

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

In some of the following theorems, the premise M = *measure-pmf* p is used. This allows stating theorems that hold for pmfs more concisely, for example, instead of *measure-pmf.prob* p $A \leq$ *measure-pmf.prob* p B we can just write M = *measure-pmf* $p \Longrightarrow$ *prob* $A \leq$ *prob* B in the locale *prob-space*.

lemma *prob-space-restrict-space*:
  assumes [*simp*]:M = *measure-pmf* p
  shows *prob-space* (*restrict-space* M (*set-pmf* p))
  ⟨*proof*⟩

The abbreviation below is used to specify the discrete σ-algebra on *UNIV* as a measure space. It is used in places where the existing definitions, such as *indep-vars*, expect a measure space even though only a *measurable* space is really needed, i.e., in cases where the property is invariant with respect to the actual measure.

hide-const (**open**) *discrete*

abbreviation *discrete* $\equiv$ *count-space UNIV*

lemma (**in** *prob-space*) *indep-vars-restrict-space*:
  assumes [*simp*]:M = *measure-pmf* p
  assumes
    prob-space.indep-vars (*restrict-space* M (*set-pmf* p)) (λ-. *discrete*) X I
  shows *indep-vars* (λ-. *discrete*) X I
⟨*proof*⟩

lemma (**in** *prob-space*) *measure-pmf-eq*:
  assumes M = *measure-pmf* p
  assumes $\bigwedge x.\ x \in$ *set-pmf* $p \Longrightarrow (x \in P) = (x \in Q)$
  shows *prob* P = *prob* Q
  ⟨*proof*⟩

The following lemma is an intro rule for the independence of random variables defined on pmfs. In that case it is possible, to check the independence of random variables point-wise.

The proof relies on the fact that the support of a pmf is countable and the σ-algebra of such a set can be generated by singletons.

lemma (**in** *prob-space*) *indep-vars-pmf*:
  assumes [*simp*]:$M = \textit{measure-pmf}\ p$
  assumes $\bigwedge a\ J.\ J \subseteq I \Longrightarrow \textit{finite}\ J \Longrightarrow$
    $\textit{prob}\ \{\omega.\ \forall\, i \in J.\ X\ i\ \omega = a\ i\} = (\prod i \in J.\ \textit{prob}\ \{\omega.\ X\ i\ \omega = a\ i\})$
  shows *indep-vars* (λ-. *discrete*) X I
⟨*proof*⟩

lemma (**in** *prob-space*) *split-indep-events*:
  assumes $M = \textit{measure-pmf}\ p$
  assumes *indep-vars* (λi. *discrete*) X' I
  assumes $K \subseteq I$ *finite* K
  shows $\textit{prob}\ \{\omega.\ \forall\, x \in K.\ P\ x\ (X'\ x\ \omega)\} = (\prod x \in K.\ \textit{prob}\ \{\omega.\ P\ x\ (X'\ x\ \omega)\})$
⟨*proof*⟩

lemma *pmf-of-set-eq-uniform*:
  assumes *finite* A $A \neq \{\}$
  shows *measure-pmf* (*pmf-of-set* A) = *uniform-measure discrete* A
⟨*proof*⟩

lemma (**in** *prob-space*) *uniform-onI*:
  assumes $M = \textit{measure-pmf}\ p$
  assumes *finite* A $A \neq \{\}$
  assumes $\bigwedge a.\ \textit{prob}\ \{\omega.\ X\ \omega = a\} = \textit{indicator}\ A\ a\ /\ \textit{card}\ A$
  shows *uniform-on* X A
⟨*proof*⟩

end

3 Carter-Wegman Hash Family

theory *Carter-Wegman-Hash-Family*
  imports
    Interpolation-Polynomials-HOL-Algebra.Interpolation-Polynomial-Cardinalities
    Universal-Hash-Families-More-Independent-Families
begin

The Carter-Wegman hash family is a generic method to obtain k-universal hash families for arbitrary k. (There are faster solutions, such as tabulation hashing, which are limited to a specific k. See for example [2].)

The construction was described by Wegman and Carter [4], it is a hash family between the elements of a finite field and works by choosing randomly a polynomial over the field with degree less than k. The hash function is the evaluation of a such a polynomial.

Using the property that the fraction of polynomials interpolating a given set of $s \leq k$ points is $1\ /\ real\ (card\ (carrier\ R))^s$, which is shown in [1], it is possible to obtain both that the hash functions are k-wise independent and uniformly distributed.

In the following two locales are introduced, the main reason for both is to make the statements of the theorems and proofs more concise. The first locale *poly-hash-family* fixes a finite ring R and the probability space of the polynomials of degree less than k. Because the ring is not a field, the family is not yet k-universal, but it is still possible to state a few results such as the fact that the range of the hash function is a subset of the carrier of the ring.

The second locale *carter-wegman-hash-family* is an extension of the former with the assumption that R is a field with which the k-universality follows.

The reason for using two separate locales is to support use cases, where the ring is only probably a field. For example if it is the set of integers modulo an approximate prime, in such a situation a subset of the properties of an algorithm using approximate primes would need to be verified even if R is only a ring.

definition (**in** *ring*) *hash x ω = eval ω x*

locale *poly-hash-family = ring +*
fixes *k :: nat*
assumes *finite-carrier*[*simp*]: *finite* (*carrier R*)
assumes *k-ge-0*: *k > 0*
begin

definition *space* **where** *space = bounded-degree-polynomials R k*
definition *M* **where** *M = measure-pmf* (*pmf-of-set space*)

lemma *finite-space*[*simp*]:*finite space*
⟨*proof*⟩

lemma *non-empty-bounded-degree-polynomials*[*simp*]:*space ≠ {}*
⟨*proof*⟩

This is to add *carrier-not-empty* to the simp set in the context of *poly-hash-family*:

lemma *non-empty-carrier*[*simp*]: *carrier R ≠ {}*
⟨*proof*⟩

sublocale *prob-space M*
⟨*proof*⟩

lemma *hash-range*[*simp*]:
assumes *ω ∈ space*
assumes *x ∈ carrier R*

shows *hash* x $\omega \in$ *carrier R*
⟨*proof*⟩

lemma *hash-range-2*:
assumes $\omega \in$ *space*
shows $(\lambda x.$ *hash* x $\omega)$ ‘ *carrier R* $\subseteq$ *carrier R*
⟨*proof*⟩

lemma *integrable-M*[*simp*]:
fixes f :: $'a$ *list* $\Rightarrow$ $'c$::{*banach*, *second-countable-topology*}
shows *integrable M f*
⟨*proof*⟩

end

locale *carter-wegman-hash-family* = *poly-hash-family* +
assumes *field-R*: *field R*
begin
sublocale *field*
⟨*proof*⟩

abbreviation *field-size* $\equiv$ *card* (*carrier R*)

lemma *poly-cards*:
assumes $K \subseteq$ *carrier R*
assumes *card* $K \leq k$
assumes y ‘ $K \subseteq$ (*carrier R*)
shows
card $\{\omega \in$ *space*. $(\forall k \in K.$ *eval* ω $k = y\ k)\}$ = *field-size*^($k-$*card* K)
⟨*proof*⟩

lemma *poly-cards-single*:
assumes $x \in$ *carrier R*
assumes $y \in$ *carrier R*
shows *card* $\{\omega \in$ *space*. *eval* ω $x = y\}$ = *field-size*^($k-1$)
⟨*proof*⟩

lemma *hash-prob*:
assumes $K \subseteq$ *carrier R*
assumes *card* $K \leq k$
assumes y ‘ $K \subseteq$ *carrier R*
shows
prob $\{\omega.$ $(\forall x \in K.$ *hash* x $\omega = y\ x)\}$ = $1/$(*real field-size*)^*card* K
⟨*proof*⟩

lemma *prob-single*:
assumes $x \in$ *carrier R* $y \in$ *carrier R*
shows *prob* $\{\omega.$ *hash* x $\omega = y\}$ = $1/$(*real field-size*)
⟨*proof*⟩

lemma *prob-range*:
  assumes [*simp*]:$x \in$ *carrier R*
  shows *prob* $\{\omega.$ *hash* $x\ \omega \in A\}$ = *card* ($A \cap$ *carrier R*) / *field-size*
⟨*proof*⟩

lemma *indep*:
  assumes $J \subseteq$ *carrier R*
  assumes *card* $J \leq k$
  shows *indep-vars* (λ-. *discrete*) *hash J*
⟨*proof*⟩

lemma *k-wise-indep*:
  k-wise-indep-vars k (λ-. *discrete*) *hash* (*carrier R*)
  ⟨*proof*⟩

lemma *inj-if-degree-1*:
  assumes $\omega \in$ *space*
  assumes *degree* ω = *1*
  shows *inj-on* ($\lambda x.$ *hash* $x\ \omega$) (*carrier R*)
  ⟨*proof*⟩

lemma *uniform*:
  assumes $i \in$ *carrier R*
  shows *uniform-on* (*hash i*) (*carrier R*)
⟨*proof*⟩

This the main result of this section - the Carter-Wegman hash family is *k*-universal.

theorem *k-universal*:
  k-universal k hash (*carrier R*) (*carrier R*)
  ⟨*proof*⟩

end

lemma *poly-hash-familyI*:
  assumes *ring R*
  assumes *finite* (*carrier R*)
  assumes $0 < k$
  shows *poly-hash-family R k*
  ⟨*proof*⟩

lemma *carter-wegman-hash-familyI*:
  assumes *field F*
  assumes *finite* (*carrier F*)
  assumes $0 < k$
  shows *carter-wegman-hash-family F k*
  ⟨*proof*⟩

lemma *hash-k-wise-indep*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes $1 \leq n$
shows
prob-space.k-wise-indep-vars (*pmf-of-set* (*bounded-degree-polynomials F n*)) *n*
(λ-. *pmf-of-set* (*carrier F*)) (*ring.hash F*) (*carrier F*)
$\langle$*proof*$\rangle$

lemma *hash-prob-single*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes $x \in$ *carrier F*
assumes $1 \leq n$
assumes $y \in$ *carrier F*
shows
$\mathcal{P}$(ω *in pmf-of-set* (*bounded-degree-polynomials F n*). *ring.hash F x* $\omega = y$)
= *1*/(*real* (*card* (*carrier F*)))
$\langle$*proof*$\rangle$

end

4 Indexed Products of Probability Mass Functions

theory *Universal-Hash-Families-More-Product-PMF*
imports
Concentration-Inequalities.Concentration-Inequalities-Preliminary
Finite-Fields.Finite-Fields-More-Bijections
Universal-Hash-Families-More-Independent-Families
begin

hide-const (**open**) *Isolated.discrete*

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

abbreviation *prod-pmf* **where** *prod-pmf I M* $\equiv$ *Pi-pmf I undefined M*

lemma *measure-pmf-cong*:
assumes $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow x \in P \longleftrightarrow x \in Q$
shows *measure* (*measure-pmf p*) $P =$ *measure* (*measure-pmf p*) Q
$\langle$*proof*$\rangle$

lemma *pmf-mono*:
assumes $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow x \in P \Longrightarrow x \in Q$
shows *measure* (*measure-pmf p*) $P \leq$ *measure* (*measure-pmf p*) Q
$\langle$*proof*$\rangle$

lemma *pmf-add*:
assumes $\bigwedge x.\ x \in P \Longrightarrow x \in$ *set-pmf p* $\Longrightarrow x \in Q \vee x \in R$

shows *measure p P* $\leq$ *measure p Q* + *measure p R*
⟨*proof*⟩

lemma *pmf-prod-pmf*:
assumes *finite I*
shows *pmf* (*prod-pmf I M*) *x* = (*if x* $\in$ *extensional I then* $\prod$ *i* $\in$ *I*. (*pmf* (*M i*)) (*x i*) *else 0*)
⟨*proof*⟩

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

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

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

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

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

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

lemma *pi-pmf-decompose*:
assumes *finite I*
shows *Pi-pmf I d M* = *map-pmf* ($\lambda\omega$. *restrict-dfl* (λ*i*. ω (*f i*) *i*) *I d*) (*Pi-pmf* (*f* ‘ *I*) (λ-. *d*) (λ*j*. *Pi-pmf* (*f* −‘ {*j*} $\cap$ *I*) *d M*))
⟨*proof*⟩

lemma *restrict-dfl-iter*: *restrict-dfl* (*restrict-dfl f I d*) *J d* = *restrict-dfl f* (*I* $\cap$ *J*) *d*
⟨*proof*⟩

lemma *indep-vars-restrict'*:
assumes *finite I*
shows *prob-space.indep-vars* (*Pi-pmf I d M*) (λ-. *discrete*) (λ*i* ω. *restrict-dfl* ω (*f* −‘ {*i*} $\cap$ *I*) *d*) (*f* ‘ *I*)
⟨*proof*⟩

lemma *indep-vars-restrict-intro'*:
assumes *finite* I
assumes $\bigwedge i\ \omega.\ i \in J \Longrightarrow X'\ i\ \omega = X'\ i\ (\mathit{restrict\text{-}dfl}\ \omega\ (f\ -`\ \{i\} \cap I)\ d)$
assumes $J = f\ `\ I$
assumes $\bigwedge \omega\ i.\ i \in J \Longrightarrow\ X'\ i\ \omega \in \mathit{space}\ (M'\ i)$
shows $\mathit{prob\text{-}space.indep\text{-}vars}\ (\mathit{measure\text{-}pmf}\ (\mathit{Pi\text{-}pmf}\ I\ d\ p))\ M'\ (\lambda i\ \omega.\ X'\ i\ \omega)\ J$
⟨*proof*⟩

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

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

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

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

lemma *Pi-pmf-bind-return*:
assumes *finite* I
shows $\mathit{Pi\text{-}pmf}\ I\ d\ (\lambda i.\ M\ i \gg= (\lambda x.\ \mathit{return\text{-}pmf}\ (f\ i\ x))) = \mathit{Pi\text{-}pmf}\ I\ d'\ M \gg= (\lambda x.\ \mathit{return\text{-}pmf}\ (\lambda i.\ \mathit{if}\ i \in I\ \mathit{then}\ f\ i\ (x\ i)\ \mathit{else}\ d))$
⟨*proof*⟩

lemma *pmf-of-set-prod-eq*:
assumes $A \neq \{\}$ *finite* A
assumes $B \neq \{\}$ *finite* B
shows $\mathit{pmf\text{-}of\text{-}set}\ (A \times B) = \mathit{pair\text{-}pmf}\ (\mathit{pmf\text{-}of\text{-}set}\ A)\ (\mathit{pmf\text{-}of\text{-}set}\ B)$
⟨*proof*⟩

lemma *split-pmf-mod-div'*:
  assumes $a > (0::nat)$
  assumes $b > 0$
  shows $map\text{-}pmf\ (\lambda x.\ (x\ mod\ a,\ x\ div\ a))\ (pmf\text{-}of\text{-}set\ \{..<a * b\}) = pmf\text{-}of\text{-}set\ (\{..<a\} \times \{..<b\})$
  ⟨*proof*⟩

lemma *split-pmf-mod-div*:
  assumes $a > (0::nat)$
  assumes $b > 0$
  shows $map\text{-}pmf\ (\lambda x.\ (x\ mod\ a,\ x\ div\ a))\ (pmf\text{-}of\text{-}set\ \{..<a * b\}) =$
    $pair\text{-}pmf\ (pmf\text{-}of\text{-}set\ \{..<a\})\ (pmf\text{-}of\text{-}set\ \{..<b\})$
  ⟨*proof*⟩

end

5 Pseudorandom Objects

theory *Pseudorandom-Objects*
  imports *Universal-Hash-Families-More-Product-PMF*
begin

This section introduces a combinator library for pseudorandom objects [3]. These can be thought of as PRNGs but with rigorous mathematical properties, which can be used to in algorithms to reduce their randomness usage.

Such an object represents a non-empty multiset, with an efficient mechanism to sample from it. They have a natural interpretation as a probability space (each element is selected with a probability proportional to its occurrence count in the multiset).

The following section will introduce a construction of k-independent hash families as a pseudorandom object. The AFP entry `Expander_Graphs` then follows up with expander walks as pseudorandom objects.

record *'a pseudorandom-object* =
  pro-last :: *nat*
  pro-select :: $nat \Rightarrow 'a$

definition *pro-size* **where** $pro\text{-}size\ S = pro\text{-}last\ S + 1$
definition *sample-pro* **where** $sample\text{-}pro\ S = map\text{-}pmf\ (pro\text{-}select\ S)\ (pmf\text{-}of\text{-}set\ \{0..pro\text{-}last\ S\})$

declare [[*coercion sample-pro*]]

abbreviation *pro-set* **where** $pro\text{-}set\ S \equiv set\text{-}pmf\ (sample\text{-}pro\ S)$

lemma *sample-pro-alt*: $sample\text{-}pro\ S = map\text{-}pmf\ (pro\text{-}select\ S)\ (pmf\text{-}of\text{-}set\ \{..<pro\text{-}size\ S\})$
  ⟨*proof*⟩

lemma *pro-size-gt-0*: *pro-size S* > *0*
⟨*proof*⟩

lemma *set-sample-pro*: *pro-set S* = *pro-select S* ‘ {..<*pro-size S*}
⟨*proof*⟩

lemma *set-pmf-of-set-sample-size*[*simp*]:
set-pmf (*pmf-of-set* {..<*pro-size S*}) = {..<*pro-size S*}
⟨*proof*⟩

lemma *pro-select-in-set*: *pro-select S* (*x mod pro-size S*) ∈ *pro-set S*
⟨*proof*⟩

lemma *finite-pro-set*: *finite* (*pro-set S*)
⟨*proof*⟩

lemma *integrable-sample-pro*[*simp*]:
fixes *f* :: ′*a* ⇒ ′*c*::{*banach*, *second-countable-topology*}
shows *integrable* (*measure-pmf* (*sample-pro S*)) *f*
⟨*proof*⟩

definition *list-pro* :: ′*a list* ⇒ ′*a pseudorandom-object* **where**
list-pro ls = (| *pro-last* = *length ls* − *1*, *pro-select* = (!) *ls* |)

lemma *list-pro*:
assumes *xs* ≠ []
shows *sample-pro* (*list-pro xs*) = *pmf-of-multiset* (*mset xs*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *list-pro-2*:
assumes *xs* ≠ [] *distinct xs*
shows *sample-pro* (*list-pro xs*) = *pmf-of-set* (*set xs*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *list-pro-size*:
assumes *xs* ≠ []
shows *pro-size* (*list-pro xs*) = *length xs*
⟨*proof*⟩

lemma *list-pro-set*:
assumes *xs* ≠ []
shows *pro-set* (*list-pro xs*) = *set xs*
⟨*proof*⟩

definition *nat-pro* :: *nat* ⇒ *nat pseudorandom-object* **where**
nat-pro n = (| *pro-last* = *n*−*1*, *pro-select* = *id* |)

lemma *nat-pro-size*:
assumes $n > 0$
shows*pro-size* (*nat-pro n*) = *n*
⟨*proof*⟩

lemma *nat-pro*:
assumes $n > 0$
shows *sample-pro* (*nat-pro n*) = *pmf-of-set* {..<*n*}
⟨*proof*⟩

lemma *nat-pro-set*:
assumes $n > 0$
shows *pro-set* (*nat-pro n*) = {..<*n*}
⟨*proof*⟩

fun *count-zeros* :: *nat* ⇒ *nat* ⇒ *nat* **where**
count-zeros 0 k = *0* |
count-zeros (*Suc n*) *k* = (*if odd k then 0 else 1* + *count-zeros n* (*k div 2*))

lemma *count-zeros-iff*: $j \le n \Longrightarrow$ *count-zeros n k* $\ge j \longleftrightarrow$ *2^j dvd k*
⟨*proof*⟩

lemma *count-zeros-max*:
count-zeros n k $\le n$
⟨*proof*⟩

definition *geom-pro* :: *nat* ⇒ *nat pseudorandom-object* **where**
geom-pro n = (| *pro-last* = *2^n* − *1*, *pro-select* = *count-zeros n* |)

lemma *geom-pro-size*: *pro-size* (*geom-pro n*) = *2^n*
⟨*proof*⟩

lemma *geom-pro-range*: *pro-set* (*geom-pro n*) ⊆ {..*n*}
⟨*proof*⟩

lemma *geom-pro-prob*:
measure (*sample-pro* (*geom-pro n*)) {ω. ω ≥ *j*} = *of-bool* (*j* ≤ *n*) / *2^j* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *geom-pro-prob-single*:
measure (*sample-pro* (*geom-pro n*)) {*j*} ≤ *1* / *2^j* (**is** *?L* ≤ *?R*)
⟨*proof*⟩

definition *prod-pro* ::
$'a$ *pseudorandom-object* $\Rightarrow$ $'b$ *pseudorandom-object* $\Rightarrow$ $('a \times 'b)$ *pseudorandom-object*
where
prod-pro P Q =
⦇ *pro-last* = *pro-size P* ∗ *pro-size Q* − *1*,
pro-select = (λ*k*. (*pro-select P* (*k mod pro-size P*), *pro-select Q* (*k div pro-size P*))) ⦈

lemma *prod-pro-size*:
pro-size (*prod-pro P Q*) = *pro-size P* ∗ *pro-size Q*
⟨*proof*⟩

lemma *prod-pro*:
sample-pro (*prod-pro P Q*) = *pair-pmf* (*sample-pro P*) (*sample-pro Q*) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *prod-pro-set*:
pro-set (*prod-pro P Q*) = *pro-set P* × *pro-set Q*
⟨*proof*⟩

end

6 K-Independent Hash Families as Pseudorandom Objects

theory *Pseudorandom-Objects-Hash-Families*
imports
Pseudorandom-Objects
Finite-Fields.Find-Irreducible-Poly
Carter-Wegman-Hash-Family
Universal-Hash-Families-More-Product-PMF
begin

hide-const (**open**) *Numeral-Type.mod-ring*
hide-const (**open**) *Divisibility.prime*
hide-const (**open**) *Isolated.discrete*

definition *hash-space′* ::
$('a,'b)$ *idx-ring-enum-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $('c,'d)$ *pseudorandom-object-scheme*
$\Rightarrow$ (*nat* $\Rightarrow$ $'c$) *pseudorandom-object*
where *hash-space′ R k S* = (
⦇
pro-last = *idx-size R* ^*k*−*1*,
pro-select = (λ*x i*.
pro-select S

$(idx\text{-}enum\text{-}inv\ R\ (poly\text{-}eval\ R\ (poly\text{-}enum\ R\ k\ x)\ (idx\text{-}enum\ R\ i))\ mod\ pro\text{-}size\ S))$
$\|))$

lemma *hash-prob-single'*:
assumes *field F finite* (*carrier F*)
assumes $x \in carrier\ F$
assumes $1 \le n$
shows *measure* (*pmf-of-set* (*bounded-degree-polynomials F n*)) $\{\omega.\ ring.hash\ F\ x\ \omega = y\} =$
of-bool ($y\in$ *carrier F*)/(*real* (*card* (*carrier F*))) (**is** *?L = ?R*)
⟨*proof*⟩

lemma *hash-k-wise-indep'*:
assumes *field F* $\wedge$ *finite* (*carrier F*)
assumes $1 \le n$
shows *prob-space.k-wise-indep-vars* (*pmf-of-set* (*bounded-degree-polynomials F n*)) *n*
(λ-. *discrete*) (*ring.hash F*) (*carrier F*)
⟨*proof*⟩

lemma *hash-space'*:
fixes *R* :: (*'a,'b*) *idx-ring-enum-scheme*
assumes $enum_C$ *R* $field_C$ *R*
assumes *pro-size S dvd order* (*ring-of R*)
assumes $I \subseteq \{..<order\ (ring\text{-}of\ R)\}$ *card* $I \le k$
shows *map-pmf* ($\lambda f.$ ($\lambda i\in I.$ *f i*)) (*sample-pro* (*hash-space' R k S*)) = *prod-pmf I* (λ-. *sample-pro S*)
(**is** *?L = ?R*)
⟨*proof*⟩

lemma *hash-space'-range*:
pro-select (*hash-space' R k S*) *i j* $\in$ *pro-set S*
⟨*proof*⟩

definition *hash-pro* ::
nat $\Rightarrow$ *nat* $\Rightarrow$ (*'a,'b*) *pseudorandom-object-scheme* $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) *pseudorandom-object*
where *hash-pro k d S* = (
let (*p,j*) = *split-power* (*pro-size S*);
l = *max j* (*floorlog p* (*d*−*1*))
in hash-space' (*GF* (*p^l*)) *k S*)

definition *hash-pro-spmf* ::
nat $\Rightarrow$ *nat* $\Rightarrow$ (*'a,'b*) *pseudorandom-object-scheme* $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) *pseudorandom-object spmf*
where *hash-pro-spmf k d S* =
do {
let (*p,j*) = *split-power* (*pro-size S*);

let l = *max* j (*floorlog* p $(d-1)$);
$R \leftarrow GF_R$ (p^l);
return-spmf (*hash-space'* R k S)
}

definition *hash-pro-pmf* ::
nat ⇒ *nat* ⇒ ('*a*,'*b*) *pseudorandom-object-scheme* ⇒ (*nat* ⇒ '*a*) *pseudorandom-object pmf*
where *hash-pro-pmf k d S* = *map-pmf the* (*hash-pro-spmf k d S*)

syntax
-*FLIPBIND* :: ('*a* ⇒ '*b*) ⇒ '*c* ⇒ '*b* (**infixr** ‹=<<› *54*)

syntax-consts
-*FLIPBIND* == *Monad-Syntax.bind*

translations
-*FLIPBIND f g* => *g* ≫= *f*

context
fixes *S*
fixes *d* :: *nat*
fixes *k* :: *nat*
assumes *size-prime-power*: *is-prime-power* (*pro-size S*)
begin

private definition *p* **where** *p* = *fst* (*split-power* (*pro-size S*))
private definition *j* **where** *j* = *snd* (*split-power* (*pro-size S*))
private definition *l* **where** *l* = *max j* (*floorlog p* $(d-1)$)

private lemma *split-power*: (*p*,*j*) = *split-power* (*pro-size S*)
⟨*proof*⟩ **lemma** *hash-sample-space-alt*: *hash-pro k d S* = *hash-space'* (*GF* (p^l)) *k S*
⟨*proof*⟩ **lemma** *p-prime* : *prime p* **and** *j-gt-0*: $j > 0$
⟨*proof*⟩ **lemma** *l-gt-0*: $l > 0$
⟨*proof*⟩ **lemma** *prime-power*: *is-prime-power* (p^l)
⟨*proof*⟩

lemma *hash-in-hash-pro-spmf*: *hash-pro k d S* ∈ *set-spmf* (*hash-pro-spmf k d S*)
⟨*proof*⟩

lemma *lossless-hash-pro-spmf*: *lossless-spmf* (*hash-pro-spmf k d S*)
⟨*proof*⟩

lemma *hashp-eq-hash-pro-spmf*: *set-pmf* (*hash-pro-pmf k d S*) = *set-spmf* (*hash-pro-spmf k d S*)
⟨*proof*⟩

lemma *hashp-in-hash-pro-spmf*:

assumes $x \in$ *set-pmf* (*hash-pro-pmf k d S*)
shows $x \in$ *set-spmf* (*hash-pro-spmf k d S*)
⟨*proof*⟩

lemma *hash-pro-in-hash-pro-pmf*: *hash-pro k d S* $\in$ *set-pmf* (*hash-pro-pmf k d S*)
⟨*proof*⟩

lemma *hash-pro-spmf-distr*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
assumes $I \subseteq \{..<d\}$ *card* $I \leq k$
shows *map-pmf* ($\lambda f.$ ($\lambda i{\in}I.$ *f i*)) (*sample-pro s*) = *prod-pmf I* (λ-. *sample-pro S*)
⟨*proof*⟩

lemma *hash-pro-spmf-component*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
assumes $i < d$ $k > 0$
shows *map-pmf* ($\lambda f.$ *f i*) (*sample-pro s*) = *sample-pro S* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *hash-pro-spmf-indep*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
assumes $I \subseteq \{..<d\}$ *card* $I \leq k$
shows *prob-space.indep-vars* (*sample-pro s*) (λ-. *discrete*) ($\lambda i\ \omega.\ \omega\ i$) *I*
⟨*proof*⟩

lemma *hash-pro-spmf-k-indep*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
shows *prob-space.k-wise-indep-vars* (*sample-pro s*) *k* (λ-. *discrete*) ($\lambda i\ \omega.\ \omega\ i$) $\{..<d\}$
⟨*proof*⟩ **lemma** *hash-pro-spmf-size-aux*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
shows *pro-size s* = (*p*^*l*)^*k* (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *floorlog-alt-def*:
floorlog b a = (*if 1* < *b then nat* ⌈*log* (*real b*) (*real a*+*1*)⌉ *else 0*)
⟨*proof*⟩

lemma *hash-pro-spmf-size*:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
assumes (p',j') = *split-power* (*pro-size S*)
shows *pro-size s* = (p'^(*max* j' (*floorlog* p' (*d*−*1*))))^*k*
⟨*proof*⟩

lemma *hash-pro-spmf-size*′:
assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*) $d > 0$
assumes (p',j') = *split-power* (*pro-size S*)
shows *pro-size s* = (p'^(*k*∗*max* j' (*nat* ⌈*log* p' *d*⌉)))

$\langle proof \rangle$

lemma *hash-pro-spmf-size-prime-power*:
  assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
  assumes $k > 0$
  shows *is-prime-power* (*pro-size s*)
  $\langle proof \rangle$

lemma *hash-pro-smpf-range*:
  assumes $s \in$ *set-spmf* (*hash-pro-spmf k d S*)
  shows *pro-select s i q* $\in$ *pro-set S*
$\langle proof \rangle$

lemmas *hash-pro-size'* = *hash-pro-spmf-size'*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-size* = *hash-pro-spmf-size*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-size-prime-power* = *hash-pro-spmf-size-prime-power*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-distr* = *hash-pro-spmf-distr*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-component* = *hash-pro-spmf-component*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-indep* = *hash-pro-spmf-indep*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-k-indep* = *hash-pro-spmf-k-indep*[*OF hash-in-hash-pro-spmf*]
lemmas *hash-pro-range* = *hash-pro-smpf-range*[*OF hash-in-hash-pro-spmf*]

lemmas *hash-pro-pmf-size'* = *hash-pro-spmf-size'*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-size* = *hash-pro-spmf-size*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-size-prime-power* = *hash-pro-spmf-size-prime-power*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-distr* = *hash-pro-spmf-distr*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-component* = *hash-pro-spmf-component*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-indep* = *hash-pro-spmf-indep*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-k-indep* = *hash-pro-spmf-k-indep*[*OF hashp-in-hash-pro-spmf*]
lemmas *hash-pro-pmf-range* = *hash-pro-smpf-range*[*OF hashp-in-hash-pro-spmf*]

end

open-bundle *pseudorandom-object-syntax*
begin
notation *hash-pro* (‹$\mathcal{H}$›)
notation *hash-pro-spmf* (‹$\mathcal{H}_S$›)
notation *hash-pro-pmf* (‹$\mathcal{H}_P$›)
notation *list-pro* (‹$\mathcal{L}$›)
notation *nat-pro* (‹$\mathcal{N}$›)
notation *geom-pro* (‹$\mathcal{G}$›)
notation *prod-pro* (**infixr** ‹$\times_P$› *65*)
end

end

References

[1] E. Karayel. Interpolation polynomials (in hol-algebra). *Archive of Formal Proofs*, Jan. 2022. https://isa-afp.org/entries/Interpolation_Polynomials_HOL_Algebra.html, Formal proof development.

[2] M. Thorup and Y. Zhang. Tabulation based 5-universal hashing and linear probing. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ALENEX '10, pages 62–76, USA, 2010. Society for Industrial and Applied Mathematics.

[3] S. P. Vadhan. Pseudorandomness. *Foundations and Trends®in Theoretical Computer Science*, 7(1-3):1–336, 2012.

[4] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.