Formalization of Randomized Approximation Algorithms for Frequency Moments

Emin Karayel
Department of Computer Science, Technische Universität München, Germany

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. Higher-order frequency moments provide information about the skew of the data stream which is, for example, critical information for parallel processing. (The k-th frequency moment of a data stream is the sum of the k-th powers of the occurrence counts of each element in the stream.) They introduce both lower bounds and upper bounds on the space complexity of the problems, which were later improved by newer publications. The algorithms have guaranteed success probabilities and accuracies without making any assumptions on the input distribution. They are an interesting use case for formal verification because their correctness proofs require a large body of deep results from algebra, analysis and probability theory. This work reports on the formal verification of three algorithms for the approximation of F_0, F_2 and F_k for $k \geq 3$. The results include the identification of significantly simpler algorithms with the same runtime and space complexities as the previously known ones as well as the development of several reusable components, such as a formalization of universal hash families, amplification methods for randomized algorithms, a model for one-pass data stream algorithms or a generic flexible encoding library for the verification of space complexities.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Higher order logic; Mathematics of computing → Probabilistic algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Formal Verification, Isabelle/HOL, Randomized Algorithms, Frequency Moments

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.21

Supplementary Material *Software (Isabelle/HOL Formalization)*: `https://isa-afp.org/entries/Frequency_Moments.html` [33]

Acknowledgements Special thanks to Tobias Nipkow for all his support, guidance and feedback on this work, to Manuel Eberl for advice and simplifications on the Isabelle/HOL formalization and to the anonymous reviewers for their careful feedback and many helpful comments and suggestions.

1 Introduction

Flajolet and Martin [20] introduced one of the first modern big data algorithms to approximate the number of distinct elements in a stream using a randomized algorithm with logarithmic space usage. In 1999 Alon et al. [3] realize that the estimation of the number of distinct elements is a special case of a more generic problem. They define frequency moments of an input stream $a_1, a_2, \ldots, a_m \in U$ with length m by:

$$F_k := \sum_{u \in U} C(u, a)^k \tag{1}$$

where $C(u, a)$ is the count of occurrences of u in the stream a, i.e., $C(u, a) := |\{i \mid a_i = u\}|$. Then they provide randomized space-efficient algorithms for the estimation of all frequency moments. They also overcome the need for idealized model assumptions about hash functions,

which were used in previous work. Instead they provide concrete solutions using universal hash families. Moreover, they establish lower bounds for the problems, which were later improved and matched [7, 9, 12, 28, 30].

Note that F_0 coincides with the number of distinct elements in the stream. Higher frequency moments are useful to derive information about the *skewness* of the rank-size distribution of the data stream. This is, for example, useful during query planning in database applications [14] and predicting the speed-up factor in parallel data processing [25]. Using estimates of both F_2 and F_0 it is possible to compute several statistical dispersion measures, such as variance, standard deviation or Gini's index of homogeneity [3].

The verification of these algorithms is a distinct challenge that requires a large body of mathematical results from probability theory, such as the Hoeffding, Chebyshev and Hölder[1] inequalities as well as algebraic results about finite fields and polynomials. However, the body of existing strong theoretical results in Isabelle/HOL [40] is growing, both in the Archive of Formal Proofs (AFP) [2] as well as Isabelle's own libraries, such that showing the correctness of such theory-heavy algorithms has become feasible.

On the other hand, it is very hard to gain confidence on the correctness of these algorithms empirically and/or using traditional unit tests, because:

- the correctness properties are probabilistic, and
- the correctness properties are independent of the statistical properties of the input data – however properties established using statistical test can only provide confidence for a specified input distribution.

In the accompanying formalization [33], I verify the correctness of three distinct algorithms for the estimation of frequency moments using the Isabelle/HOL theorem prover. Table 1 summarizes them together with their asymptotic space complexity in bits and the source material they were based on. I have not been able to find any previous publications on the formalization of these algorithms. They all return an approximation F_k^* of F_k with relative error $\delta > 0$ with a probability of at least $1 - \varepsilon$, i.e.,

$$P(|F_k^* - F_k| \leq \delta F_k) \geq 1 - \varepsilon, \tag{2}$$

and require only one-pass over the stream of elements. They are all Monte-Carlo algorithms, i.e., the established probability bounds hold for every input. The fact that the result has a probability distribution stems only from the internal random choices of the algorithms.

Table 1 Formally verified algorithms: n denotes the size of the universe of the elements of the stream, m is the length of the stream, $\varepsilon \in (0, 1)$ is the maximum failure probability, $\delta \in (0, 1)$ is the required relative accuracy. See also Equation 2.

Approximation of	Asymptotic Space Complexity for $n, k, m \to \infty$ and $\varepsilon, \delta \to 0^+$	Based on
F_0	$O\left(\ln(\varepsilon^{-1})\left(\ln n + \delta^{-2}(\ln(\delta^{-1}) + \ln\ln n)\right)\right)$	[5]
F_2	$O\left(\ln(\varepsilon^{-1})\delta^{-2}(\ln n + \ln m)\right)$	[3, §2.2]
F_k for[2] $k \geq 3$	$O\left(\ln(\varepsilon^{-1})\delta^{-2}(\ln n + \ln m)kn^{1-\frac{1}{k}}\right)$	[3, §2.1]

[1] The Hölder inequality is part of the formalization of L_p spaces by Gouezel [24].

[2] The algorithm is actually correct even for $k \geq 1$ but the specialized algorithms for $k \leq 2$ are better in terms of space complexity.

A note about the case F_1: Since $F_1 = m$ an exact solution for the problem only requires $O(\ln m)$ bits of memory (which is just a counter). Alon et al. also discuss a randomized algorithm requiring $O(\ln(\ln m))$ bits using approximate counting [37]. Because it deviates a lot from the more interesting cases $k \neq 1$, I did not formalize that case.

While the algorithm for F_k identically matches the source material, I made some improvements to the F_0, F_2 algorithms, but match the space and runtime complexity bounds of the source material.

In particular, contrary to previous work in this field, the results are established using simple prime fields $GF(p)$ for p prime, instead of fields with a two power order: $GF(2^n)$. This is significant because on common machines computations in simple prime fields can be implemented easily. On the other hand, especially fast multiplications in prime power fields require advanced algorithms [38, 43]. As far as I can tell, the fact that simple prime fields are sufficient to achieve the space complexity bounds in Table 1 has not been observed by previous publications.

In the case of the approximation algorithm for F_0, I could derive a new KMV-type (k-minimum value) algorithm with a rounding component that matches the complexities of Algorithm 3 by Bar-Yossef et al. [5], but its implementation (and hence verification) is simpler than their solution. While there is considerable research on KMV-type algorithms [6, 22, 50], I could not find any previous publications that verify the correctness of this variant.

Some of the results required for the formal verification are reusable general results, which I have contributed as separate entries into the AFP [31, 32, 34, 35, 36]. The formalization of the algorithms for the frequency moments are in the AFP entry: Formalization of Randomized Approximation Algorithms for Frequency Moments [33]. I needed 7349 lines (not including text or empty lines) overall, of which 4779 lines are reusable more general results.

The next section contains theoretical background used in these algorithms, a description of how the randomized algorithms are modeled using the Giry-monad as a shallow-embedding in Isabelle/HOL and how the space complexity is verified. The details about the algorithms follow in Section 3. Related work is presented in Section 4 and Section 5 concludes with future research opportunities.

2 Background

2.1 Universal Hash Families

Universal hash families are a critical component in the algorithms for F_0 and F_2. They are used to randomize the input data such that statistical methods can be employed even without any assumptions about the input distribution. To give a rough idea why randomization is useful, Figure 1 depicts the probability density function of the third smallest element of independent uniformly distributed random variables. It is visible that information about the count of random variables can be deduced by the value of order statistics of the random variables. A first idea would be applying a fully random function $h : U \to \{0, \ldots, n-1\}$ on the values of the input stream before processing it.[3] However, choosing and storing such a function requires $O(n \ln n)$ bits of memory, which is far above the space complexity of the algorithms.

[3] The application of such a function to the input stream may obviously with some extent and probability affect the frequency moment itself, which needs to be taken into account in the design of the algorithms.

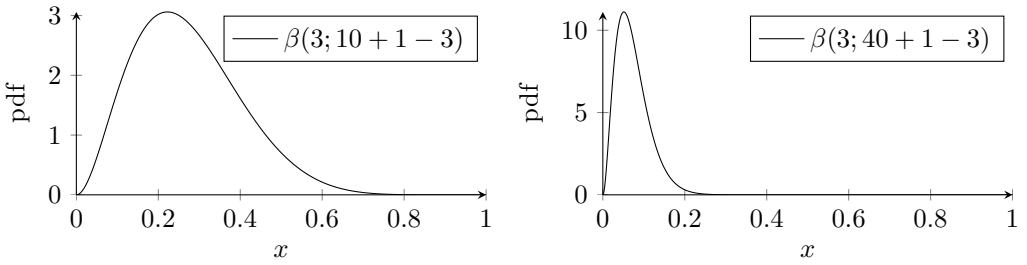

Figure 1 The probability density function for the distribution of the third-smallest element of 10 [left] (resp. 40 [right]) independent uniformly distributed random variables with range $[0, 1]$. The distributions are instances of the β-distribution.

The key insight from Alon et al. [3] was that it is possible to make headway even if the family of functions h is chosen from is only k-*universal* (where $k = 2$ in their F_0 approximation algorithm and $k = 4$ in their algorithm for the approximation of F_2). If a function h is chosen from a k-universal hash family than the restriction of h to any k elements of its domain is like a random function. More precisely:

▶ **Definition 1.** *If we regard a finite family $\mathcal{H}$ of hash functions $U \to V$ as a uniform discrete probability space, then $\mathcal{H}$ is k-universal, if*

- *for each $u \in U$ the evaluation function $h \mapsto h(u)$ is a random variable with uniform distribution on V, i.e., $P(\{h \in \mathcal{H} | h(u) = v\}) = |V|^{-1}$ and*
- *for any k or fewer distinct domain elements $u_1, \ldots, u_l$ the evaluation functions $h \mapsto h(u_1), \ldots, h \mapsto h(u_l)$ are independent random variables.*[4]

A different way of stating the second requirement in the definition is that the evaluation functions must be k-wise independent random variables. Bienaymé's identity is an example where pairwise independence is useful. With it the variance of a sum of pairwise independent variables $X_1, X_2, \ldots, X_n$ can be computed as the sum of the variances of the summands, i.e.

$$\operatorname{Var}\left(\sum_{k=1}^{n} X_i\right) = \sum_{k=1}^{n} \operatorname{Var}(X_i).$$

A generic construction for k-universal hash families was described by Wegman and Carter [49, §1] for the case where the domain and range of the hash family is a finite field $GF(q)$. (In the following I refer to this hash family as the *Carter–Wegman hash-family*.) The result essentially follows from the fact that given k key-value pairs there exists exactly one polynomial with degree strictly less than k interpolating those points. The k-universal hash family consists of the polynomials with coefficients in $GF(q)$ with degree less than k, where the evaluation of the polynomial is the hash function. Let us see how the results are stated in the formalization [36]:

```
definition (in prob-space) k-wise-indep-vars where
  k-wise-indep-vars k M' X' I = (∀ J ⊆ I. card J ≤ k ⟶ finite J ⟶ indep-vars M' X' J)
```

[4] This definition closely follows the definition from [48, §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 modification only has an effect in the corner case, where $|U| < k$ and helps avoid unnecessary special cases in the formalization.

This statement introduces a new definition for indexed sets of random variables, where any subset with no more than k elements is independent. The notation (**in** *prob-space*) means that the definition is in the context of all probability spaces. The function *card* denotes the number elements of a finite set. The predicate *indep-vars* M' X' J is true if X' is a J-indexed set of independent random variables from the probability space to the J-indexed set of measurable spaces M'.

```
definition hash where hash F x ω = ring.eval F ω x
```

This definition introduces the abbreviation *hash* for the evaluation of a polynomial over a ring F.[5]

```
lemma hash-prob-single:[6]
  assumes field F ∧ finite (carrier F)
  assumes {x, y} ⊆ carrier F
  assumes 1 ≤ n
  shows 𝒫(ω in pmf-of-set (bounded-degree-polynomials F n). hash F x ω = y)
     = 1/(card (carrier F))
```

This lemma implies that the Carter–Wegman hash-family fulfills the first condition of Definition 1: Assuming F is a finite field then the *hash* function is a random variable with uniform distribution on the probability space *pmf-of-set* (*bounded-degree-polynomials* F n), i.e., the set of polynomials with coefficients in F and degree less than n. The expression *carrier* F denotes the underlying set of the field and the notation in the last line, $\mathcal{P}(\omega \text{ in } M.\ P\ \omega)$, denotes the probability of the event $\{\omega \mid P\,\omega\}$ in the probability space M.

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

This lemma implies that the Carter–Wegman hash-family fulfills the second condition of Definition 1, i.e., if F is a finite field, then the *hash* functions are k-wise independent random variables.

Besides polynomials there is also a method called tabulation hashing to construct k-universal hash families for $k \leq 5$ [47]. In the case where V has only two values, orthogonal arrays of strength k [11] can also be interpreted as k-universal hash families. However, because of the generic nature of Carter and Wegman's solution, I formalized their construction.

As noted in the introduction, I use the simple prime fields $GF(p)$ where p is prime. In particular, if the universe size is n, then the smallest prime $p \geq n$ is chosen to construct a k-universal hash-family for the stream elements. Because of Bertrand's postulate[7] $p \leq 2n+2$, which is used to bound the space used for the coefficients of the polynomial as well as the hashed values.

[5] Some of the results in the theory file require only that F is a ring. Currently the lemmas are only applied in the context of finite fields, but there is a good reason to avoid the field assumption when it is not necessary. An example is the use of approximate primality tests where F would be a field only with high probability but it would be unconditionally a ring.

[6] To improve readability embeddings between natural numbers, integers, rational, floating point and (extended) real numbers are omitted.

[7] Bertrand's postulate was formalized in Isabelle/HOL by Biendarra and Eberl [8].

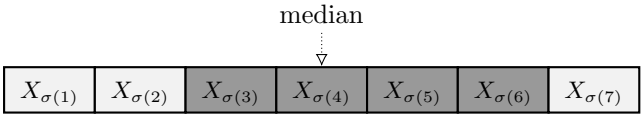

Figure 2 An example for 7 random variables (sorted via the permutation σ). The variables that are inside the desired interval $[a, b]$ are shaded gray.

In some publications in this field standard (or cryptographic) hash functions are assumed to be independent random variables. See for example in the context of F_0-estimation: [19, 20, 22, 26]. While there is also empirical evidence, that practically useful conclusions can be drawn under such model assumptions, in this work I have followed the approach by Alon et al. [3] and use universal hash families to avoid unjustified statistical model assumptions.

2.2 The Median Method

The reader may have observed that a common factor of the space complexities of the algorithms is $\ln(\varepsilon^{-1})$. The factor stems from the application of the median method to amplify the success probability [3]. To understand the method let us consider $2n + 1$ independent random variables $X_1, X_2, \ldots, X_{2n+1}$, that are in an interval $[a, b]$ with a probability of $2/3$. In the case of the algorithms a (resp. b) denote the minimal (respectively maximal) value the algorithm may return given the desired accuracy parameter. The interesting result is that the median of the random variables will be in the interval $[a, b]$ with a considerably higher probability of $1 - \exp\left(\frac{-(n+2)}{9}\right)$.

To see why this works, let us make a preliminary observation:

▶ **Observation 2.** *If at least $n + 1$ of the random variables are in the desired interval, then the median will be as well. This is because if we sort the random variables X_i the random variables that are in the interval form a consecutive subsequence in the sorted sequence. If its length is at least $n + 1$, it will necessarily contain the median. See also Figure 2 for an example.*

Hence we are left with estimating the probability that at least $n + 1$ of the X_i are in the range $[a, b]$. Let us introduce a second set of random variables, indicating whether each of the above random variables X_i are within the desired interval:

$$Y_i := \begin{cases} 1 & \text{if } X_i \in [a, b] \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that the values of random variables Y_i are either 0 or 1 and the expectation of each is bounded from below by $2/3$ because of our assumption about the probability that each X_i is in $[a, b]$ is $2/3$, i.e., $\mathbb{E}(Y_i) \geq 2/3$. Now, Hoeffding's inequality [27] implies for the sum $S = Y_1 + \ldots + Y_{2n+1}$ – the number of random variables X_i whose values lie within the interval $[a, b]$:

$$P(S \leq \mathbb{E}(S) - t) \leq \exp\left(\frac{-2t^2}{2n+1}\right).$$

The case we are interested in is $t = \frac{n}{3} + \frac{2}{3}$. Thus

$$P(S \leq n) \leq \exp\left(\frac{-2(n+2)^2}{9(2n+1)}\right) \leq \exp\left(\frac{-(n+2)}{9}\right).$$

I.e. the probability that n or fewer of the X_i are inside of $[a, b]$ decreases exponentially with n. With Observation 2 this implies, that the probability that the median of the random variables X_i is outside the interval $[a, b]$ decreases exponentially with n.

In the design of the algorithms for F_0, F_2 and F_k, this result is used to amplify the probabilities of the algorithms. To achieve this, the algorithms run $18 \ln(\varepsilon^{-1})$ copies of the same estimation algorithm in parallel (with fully independent random choices). Each copy obtains a result which approximates F_k with a relative error of δ with a probability of $2/3$. In the final step the algorithm returns the median of the results of the copies, which then is in the desired interval with probability $1 - \varepsilon$. The formalization [35] includes a generalized version of this result:

```
lemma (in prob-space) median-bound-1:
  assumes α > 0
  assumes ε ∈ {0<..<1}
  assumes indep-vars (λ-. borel) X {0..<n}
  assumes n ≥ - ln ε / (2 * α²)
  assumes ∀ i ∈ {0..<n}. P(ω in M. X i ω ∈ ({a..b} :: real set)) ≥ 1/2+α
  shows P(ω in M. median n (λi. X i ω) ∈ {a..b}) ≥ 1-ε
```

Assuming M is a probability space, $\alpha > 0$, $0 < \varepsilon < 1$, $n \geq \frac{1}{2} \ln(\varepsilon^{-1})\alpha^{-2}$ and $X_0, \ldots, X_{n-1}$ are independent Borel-measurable random variables: If the probability for each X_i to be in the interval $[a, b] \subset \mathbb{R}$ is $\frac{1}{2} + \alpha$ then the probability that the median is in the same interval is at least $1 - \varepsilon$.

▶ Note. The formalization [35] contains an even more general lemma *median-bound* where the above result is shown for all convex sets in second-countable linearly ordered Borel spaces instead of just finite closed intervals of the form $[a, b] \subset \mathbb{R}$. The generalized version might be useful, if the codomain of the random variables is of another type than the real numbers, such as the rational numbers, or if the interval is not finite and closed. The lemma *median-bound-1* above is a specialization of *median-bound*.

▶ Note. Observation 2 is missing in previous publications. That it is necessary for a rigorous proof became apparent during the formalization. The argument starting from Eq. 3 can be found in [3, §2.1]; it is included here for completeness.

▶ Note. The requirement that the variables are *Borel* measurable in the lemma is essential. Without the assumption, the median of the random variables would not necessarily be measurable. For interested readers: The measurability proof in the formalization for the median relies on the existence of branch-free comparison-sort algorithms.[8] Given the number of elements in the sequence, such an algorithm performs compare-swap operations in a pre-defined order on pre-defined indices. The results of each compare swap operation can be represented as the pair of functions: $\min(X_i, X_j)$, $\max(X_i, X_j)$, which are Borel-measurable if X_i, X_j are. And the entire sorting operation can, using a branch-free comparison sort algorithm, be represented as a repeated application of such compare-swap operations. Thus the median – and any other order statistic – of the random variables is still measurable. [9]

[8] Sometimes these algorithms are also called sorting networks or oblivious comparison sort algorithms [41].

[9] It may be possible to proof the measurability of order statistics directly by verifying that the existing *sort* operation in Isabelle is measurable. However the approach using a branch-free sorting algorithm is more concise, in particular, it circumvents the need for introducing a σ-algebra on lists.

2.3 Formalization of Randomized Algorithms

Eberl et al. [16] built a library in Isabelle/HOL for the formalization of randomized algorithms, in particular a formalization of the Giry monad. To introduce the notation, let us first consider a few minimal examples:

```
example[10] =
  do {
    a ← pmf-of-set {0, 1};
    return-pmf (a+1)
  }
```

This example represents an algorithm, where a is uniformly chosen from the set $\{0, 1\}$ and the successor of a is then returned. On the other hand from the probabilistic perspective, it makes sense to think of $a \mapsto a + 1$ as a random variable on the probability space $\{0, 1\}$ and the entire expression represents the distribution of that random variable. Indeed it is easy to show that:

```
lemma[10] example = pmf-of-set {1, 2}
```

Note in general, the term *pmf-of-set A* is a probability space, which assigns the same probability to each element of A if it is a finite, non-empty set. The abbreviation PMF stands for *probability mass function*, which are a subtype of probability spaces in Isabelle with the condition that the σ-algebra is *discrete*, more precisely, the σ-algebra must be the universe of the type forming the events. They have the advantage that all functions defined on these probability spaces are automatically measurable. The disadvantage is that the support of a measure defined on a discrete σ-algebra must necessarily be a countable set. Hence, probability mass functions are a well-suited model for randomized algorithms, where the probability spaces will be discrete anyway.

Let us investigate the case when multiple random operations are composed:

```
do[10] {
  a ← pmf-of-set {0, 1};
  b ← pmf-of-set {2, 3};
  return-pmf (a, b)
}
```

The resulting probability space is the product space $\{0, 1\} \times \{2, 3\}$, again with uniform probability. This means independent sequential composition can be though of as the construction of the product probability space. However, things can become more complex, when earlier random variables influence later random operations:

```
do[10] {
  a ← pmf-of-set {1, 2};
  b ← pmf-of-set {0..<a};
  return-pmf (a, b)
}
```

Here, the resulting probability space is a dependent sum: $\bigcup_{a \in \{1,2\}} \{a\} \times \{0, .., a - 1\}$. The probability of the pair $(0, 1)$ is $\frac{1}{2}$ while the probability of the pairs $\{(0, 2), (1, 2)\}$ is $\frac{1}{4}$. Note that the components are not independent random variables any more. On a deeper level, these probability spaces are expressions with two combinators:

[10] This is example code. It is not part of the accompanying formalization [33].

- *return-pmf*: This operation returns the Dirac measure, the probability of an event is 1 exactly if it contains the argument of *return-pmf*.
- *bind-pmf*: This operation builds a new probability space, using a first probability space Ω_1 and a function that maps each element of $x \in \Omega_1$ to a new probability space $\Omega_2(x)$. We can write this as $\Omega_1 \gg\!= \Omega_2$, where the probability of an event E in $\Omega_1 \gg\!= \Omega_2$ is: $\int_{\Omega_1} P_{\Omega_2(\omega)}(E)\, d\omega$.[11] Note: In the *do*-notation the bind operator is implicitly inserted, whenever there is a semicolon.

Because the algorithms for the frequency moments are one-pass streaming algorithms, they are represented using three functions over the Giry monad. First, an initialization function that sets up the initial state based on the parameters: the desired relative accuracy δ, the desired success probability ε, the size of the universe of the stream elements n. For simplicity, we assume the stream elements are represented as natural numbers in $\{0, .., n-1\}$. Note: A state of these algorithms is also called *sketch* or synopsis in the context of frequency moments. Second an update function that processes a single stream element and updates the state. And finally a result function that computes an approximation of the frequency moment. We can then describe the distribution of the algorithm for the stream elements $a_1, \ldots, a_m$ like:

```
do[10] {
  s_0 ←init δ ε n;
  s_1 ←update a_0 s_0;
  s_2 ←update a_1 s_1;
  ...
  s_m ←update a_{m-1} s_{m-1};
  result s_m
}
```

which can be written more succinctly as:

$$\mathit{fold}\ (\lambda a\ s.\ s \gg\!= \mathit{update}\ a)\ \mathit{as}\ (\mathit{init}\ \delta\ \varepsilon\ n) \gg\!= \mathit{result} \tag{4}$$

The following snippet is the theorem in the formalization that establishes the correctness property for the F_0 estimation algorithm. The theorems for the correctness of the F_2 and F_k estimation algorithms are analogous:

```
theorem f0-alg-correct:[6]
  assumes ε ∈ {0<..<1}
  assumes δ ∈ {0<..<1}
  assumes set as ⊆ {0..<n}
  defines M ≡ fold (λa state. state ⤜ f0-update a) as (f0-init δ ε n) ⤜ f0-result
  shows P(ω in measure-pmf M. |ω − F 0 as| ≤ δ * F 0 as) ≥ 1 − ε
```

The first two assumptions establish that ε and δ are strictly between 0 and 1. The next assumption is the requirement that the stream elements are elements in $\{0, \ldots, n-1\}$. M is defined – as discussed above (Equation 4) – as the distribution of the estimation algorithm, described by the three functions *f0-init*, *f0-update* and *f0-result* after processing the stream elements *as*. The final line establishes that the relative error of the estimate is less than δ with probability $1 - \varepsilon$. The term *F k as* refers to the actual k-th frequency moment of the stream *as* as defined in Equation 1. While this is exciting, it is also necessary to verify the space usage of the algorithms, which is going to be discussed in the next section.

[11] Especially, in the case, where the probability spaces are not countable, the construction of the resulting probability space is non-trivial. See also Eberl et al. [16] for more details on this and the Giry Monad.

2.4 Verification of the space complexity

Because the algorithms are shallowly embedded as functions in the logic, it is not directly possible to verify the memory-complexity of the algorithms. A possible solution would be to represent the algorithm within a formalized machine model and show its equivalence to the high-level representations in the logic. This is for example discussed by Myreen [39, §1].

However – because of the representation of the streaming algorithms using the three functions as described above – it is still possible to rigorously establish a bound on the memory requirements of the states the algorithms reach before and after processing each element, i.e., the sketch size.[12] For this, I decided to build an encoding of the states into bit sequences and use the length of it as a measure of the size of the data structure. In general any injective function from the state space to lists of booleans would form such an encoding and thus would provide an upper bound on the space usage, i.e., an idea would be to show a statement of the form:

$$\exists f.\ \mathit{inj}\ f \wedge \forall s \in S\ \varepsilon\ \delta\ n.\ \mathit{length}\ (f\ s) \leq F\ \varepsilon\ \delta\ n \qquad *^{10}$$

where $S\ \varepsilon\ \delta\ n$ denotes the set of states the algorithm may reach for all possible inputs and internal random operations with the provided parameters ε, δ, n and F denotes the upper bound on the space usage in bits to be shown.[13]

It turns out that this is too restrictive. The condition *inj f* requires the function to be injective on the entire universe, i.e., all possible elements of the type of the state space, even though the reachable states might be a smaller set. An example where this is an issue is when the type of the state is not a countable set. For example coefficients of the finite field $GF(p)$ can be encoded using $\ln p$ bits, but their type is: *int set*, representing each element of the field as a congruence class. A fix for this problem is allowing the encoding to be a partial function, which still needs to be injective on its domain, and requiring that the reachable states are in the domain of the function:

$$\exists f.\ \mathit{inj\text{-}on}\ f\ (\mathit{dom}\ f) \wedge \forall s \in S\ \varepsilon\ \delta\ n.\ s \in \mathit{dom}\ f \wedge \mathit{length}\ (\mathit{the}\ (f\ s)) \leq F\ \varepsilon\ \delta\ n \qquad *^{10} \qquad (5)$$

Note: Partial functions are represented using the option type: *'a ⇒ 'b option*. If x is outside of the domain of f then $f\ x = \mathit{None}$. If $f\ x = \mathit{Some}\ y$ then x is in the domain and the value of f at x is y. Moreover, *dom f* denotes the domain and *the* $(f\ x)$ is the value of the partial function if $x \in \mathit{dom}\ f$. With the introduction of the new function *bit-count*:

```
fun bit-count :: bool list option ⇒ ereal where[6]
  bit-count None = ∞ |
  bit-count (Some x) = length x
```

the conjunction on the right hand side of Equation 5 can be expressed more concisely as *bit-count* $(f\ s) \leq F\ \varepsilon\ \delta\ n$, i.e., a finite upper bound on bit-count automatically implies that s must have been in the domain of f.[14] The following snippet is the actual result for the space-complexity of the F_0-Algorithm in the formalization [33]:

[12] It is informally easy to see that the capacity bounds are not exceeded even during an update operation.
[13] Note that the order of the quantifiers is important.
[14] Subsection 2.5 contains a second reason for setting the *bit-count* of unencodable values to ∞.

```
definition encode-f0-state :: f0-state ⇒ bool list option where [omitted ...]
fun f0-space-usage :: (nat × rat × rat) ⇒ real where [omitted ...]

lemma inj-on encode-f0-state (dom encode-f0-state)

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

Instead of showing an existence result like in Equation 5, the encoding function *encode-f0-state* is defined explicitly and that it is injective on its domain is verified in a separate lemma. The function *f0-space-usage* is a pure arithmetic expression in terms of n, ε and δ. The theorem establishes that the reachable states are encodable using the encoding function and meet the memory bounds of *f0-space-usage*. The syntax *AE ω in M* stands for *almost everywhere*, this means the predicate must be true up to a set of probability 0. Since the states of the algorithms are probability distributions, the reachable states constitute the elements of M that have a non-zero probability. Besides providing an explicit bound using the function *f0-space-usage*, the next theorem concludes with the asymptotic space complexity.

```
theorem f0-asymptotic-space-complexity:[6]
  f0-space-usage ∈ O[at-top ×_F at-right 0 ×_F at-right 0](λ (n, ε, δ). ln (1 / ε) *
  (ln n + 1 / δ^2 * (ln (ln n) + ln (1 / δ))))
```

This means that the space usage is bounded by $C \ln(\varepsilon^{-1}) \left(\ln n + \delta^{-2}(\ln(\ln n)) + \ln(\delta^{-1})\right)$ for some constant C for sufficiently large n, ε^{-1} and δ^{-1}.

2.5 A flexible encoding library

As noted in the previous section, the memory requirement of the states the algorithms reach is measured by encoding them into bit strings. To achieve this, I have built a small flexible encoding library [31] for Isabelle data structures comprised of encoding functions for primitive types and a set of combinators to handle structured data types. The encoding functions are *prefix-free*, i.e., if $f\ x$ is a (not necessarily strict) prefix of $f\ y$ then $x{=}y$. This implies in particular that they are injective, which implies that the resulting bit strings can be decoded. Moreover prefix-freeness has the useful property that it is preserved under concatenation, i.e., if f, g are prefix-free, then $\lambda(x,\ y).\ f\ x\ @\ g\ y$ is also.[15] The symbol @ stands for the concatenation of two bit strings. To see why this works, let us observe another characterization of prefix-free functions:

$$f\ x_1\ @\ y_1 = f\ x_2\ @\ y_2 \Longrightarrow x_1 = x_2 \wedge y_1 = y_2 \qquad *^{10}$$

with which it is easy to conclude, for example if f, g are prefix-free:

$$\begin{aligned} &f\ x_1\ @\ g\ y_1\ @\ z_1 = f\ x_2\ @\ g\ y_2\ @\ z_2 \Longrightarrow \\ &x_1 = x_2 \wedge g\ y_1\ @\ z_1 = g\ y_2\ @\ z_2 \Longrightarrow \\ &x_1 = x_2 \wedge y_1 = y_2 \wedge z_1 = z_2 \end{aligned} \qquad *^{10}$$

[15] The approach of using prefix-free codes [on the byte-level] is commonly utilized in many serialization libraries. See for example [10, §4.2.1].

Table 2 Encoding functions for primitive types and combinators.

Sym	Description	Bit Count
N_e	Natural numbers[16]	*bit-count* $(N_e\ n) \leq 2 * log\ 2\ (real\ n{+}1) + 1$
I_e	Integers	*bit-count* $(I_e\ x) \leq 2 * log\ 2\ (\lvert x\rvert{+}1) + 3$
F_e	Floating point numbers	*bit-count* $(F_e\ (float\text{-}of\ (m * 2\ powr\ e)) \leq$ $6 + 2 * (log\ 2\ (\lvert m\rvert + 2) + log\ 2\ (\lvert e\rvert + 1))$
$\times_e$	Tuples	*bit-count* $((e \times_e f)\ (x,y)) =$ *bit-count* $(e\ x) +$ *bit-count* $(f\ y)$
$\bowtie_e$	Dependent tuples	*bit-count* $((e \bowtie_e f)\ (x,y)) =$ *bit-count* $(e\ x) +$ *bit-count* $(f\ x\ y)$
L_e	Lists	*bit-count* $(L_e\ f\ xs) = (\sum x \leftarrow xs.$ *bit-count* $(f\ x){+}1) + 1$
S_e	Finite sets	*finite* $S \Longrightarrow$ *bit-count* $(S_e\ e\ S) = (\sum x \in S.$ *bit-count* $(e\ x){+}1){+}1$
$\rightarrow_e$	Functions defined on *set xs*	$f \in$ *extensional* $(set\ xs) \Longrightarrow$ *bit-count* $((xs \rightarrow_e e)\ f) = (\sum x \leftarrow xs.$ *bit-count* $(e\ (f\ x)))$

Table 2 summarizes the encoding functions and combinators that are used to build encodings for the states. An interesting property about the definition of *bit-count* is that the equation:

$$bit\text{-}count\ ((e_1 \times_e e_2)\ (x_1,x_2)) = bit\text{-}count\ (e_1\ x_1) + bit\text{-}count\ (e_2\ x_2)$$

holds unconditionally, e.g., even if x_1 and/or x_2 are not in the domain of e_1/e_2. This is because of the facts: $\infty + x = \infty$, $x + \infty = \infty$ if $x \geq 0$ in the extended reals, and that a pair is in the domain of $e_1 \times_e e_2$ if and only if its first component is in the domain of e_1 and its second component in the domain of e_2. Similar equations hold unconditionally for the other combinators, which means that in the proof of the theorems for space usage it was possible to show the property that the state is part of the domain of the encoding function, as well as the actual space bound using the same reasoning step.

A tempting design question is whether it would be possible to derive such an encoding fully automatically. However, in this use case, it is important to *choose* an efficient encoding for the reachable states of the algorithms.

A key benefit of this approach is that it allows verification of the space complexity of data structures used in high-level specifications of algorithms close to the mathematical representation, for example, using cosets and indexed products.

3 The Algorithms

3.1 Frequency Moment 0

The original plan I had was to formalize Algorithm 3 from Bar-Yossef et al. [5]. It is the solution with the best space-time trade-off in the paper. The authors describe it as a modification of the Gibbons-Tirthapura algorithm. Briefly, it stores only the elements of the stream that have a given count l of leading zeros (or more) in the binary representation of their hash values. The value l is initialized to 0 at the beginning and is incremented during the run of the algorithm, such that the set of filtered elements fit in the allocated space. The modification by Bar-Yossef is to avoid storing the elements themselves in the state, but only a hash of them, for that purpose they introduce a second hash function.

[16] Prefix free codes for natural numbers are also called universal codes. See for example Elias [17].

During the formalization, it became apparent that a simpler algorithm is possible with the same space and amortized runtime cost as that one and an improved worst-case runtime. It uses only a single hash function and does not require its range to be a 2 power. It is based on the first algorithm described in the same paper but with an added rounding operation to the hashed values. Since that kind of algorithm is called KMV synopsis or sketch in newer publications [6, 42, 44], where the abbreviation KMV stands for *k-minimum value*[17], it makes sense to call the new algorithm Rounding-KMV. The general principle of KMV algorithms is to use the t-th smallest element of the hashed stream elements, where t is chosen according to the required accuracy parameter. This can be done by keeping track of the smallest t hashed stream elements during the course of the algorithm and using the maximum for the estimation step. Table 3 summarizes the algorithms discussed in this subsection.

Table 3 Algorithms mentioned in this subsection.

Algorithm	Space usage with respect to δ, n[18]	Hash space	Based on
Gibbons–Tirthapura [21]	$O\left(\delta^{-2}\ln n\right)$	$GF(2^e)$	-
Algorithm 3 [5]	$O\left(\delta^{-2}(\ln\ln n + \ln\delta^{-1}) + \ln n\right)$	$GF(2^e)$	Gibbons–T.
Algorithm 1 [5]	-	$[0,1] \subset \mathbb{R}$	-
Standard KMV (below)	$O\left(\delta^{-2}\ln n\right)$	$GF(p)$	Algorithm 1 [5]
Rounding KMV (below)	$O\left(\delta^{-2}(\ln\ln n + \ln(\delta^{-1})) + \ln n\right)$	$GF(p)$	Standard KMV

Let us first review the correctness proof for the standard KMV algorithm[19] (without rounding operation) with the Carter–Wegman hash family. This means we need to investigate the distribution of the t-th smallest element. For that let $a_1, \ldots, a_m$ be the elements of the stream and $a_i \in \{0, .., n-1\}$, let $A = \{a_1, \ldots, a_m\}$ be the set of distinct elements in the stream. Note that: $F_0 = |A|$, but $m \geq F_0$, since the a_i are not necessarily distinct. Let $p \geq \max(n, 11)$ be a prime and let h be uniformly chosen from the 2-universal Carter–Wegman hash family.

It makes sense to investigate the two closely related random variables X_t and $X_t^{\#}$ denoting the t-th smallest element of the hashed stream elements $H = \{h(a) \mid a \in A\}$, where the second one treats distinct elements of A mapped to the same value by the hash function as separate elements, while the first one does not. See also Figure 3 for an example, where X_t and $X_t^{\#}$ differ. More precisely X_t and $X_t^{\#}$ are the unique random variables fulfilling the following conditions:

$$X_t \in H \qquad\qquad X_t^{\#} \in H$$
$$|\{x \in H \mid x < X_t\}| = k-1 \qquad |\{a \in A \mid h(x) < X_t^{\#}\}| = k-1$$

While it is more space-efficient to compute X_t; it is mathematically easier to investigate the distribution of $X_t^{\#}$, in particular, if [and only if] there are at least t elements $a \in A$ such that $h(a) < u$, then $X_t^{\#}$ must be strictly smaller than u as well.

[17] Since k is being used to denote the order of the frequency moment in this work; the letter t is for the index of the rank instead.
[18] If ε is not assumed to be constant all complexities need to be multiplied by another factor of $ln(\varepsilon^{-1})$
[19] Bar-Yossef et al. provide a proof for the idealized case, where the hash family maps into $[0, 1]$. The proof here differs from theirs to cover the more realistic case with the Carter–Wegman hash family.

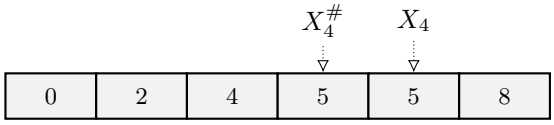

Figure 3 An example for a collision in the application of the hash function, leading to a difference between X_t and $X_t^\#$. The numbers in the boxes denote hash values of distinct elements of A.

The expectation and variance of the random variable Q_u that counts the number of elements $a \in A$ hashed to a value strictly less than u is easy to determine:

$$Z_{u,a} := \begin{cases} 1 & \text{if } h(a) < u \\ 0 & \text{otherwise.} \end{cases} \qquad Q_u := \sum_{a \in A} Z_{u,a}$$

Note that $\mathrm{Var}(Z_{u,a}) \leq \frac{u}{p}$ and $\mathbb{E}(Z_{u,a}) = \frac{u}{p}$. Because the $Z_{u,a}$ are pairwise independent, we can apply Bienaymé's identity to conclude that $\mathrm{Var}(Q_u) \leq \frac{|F_0|u}{p}$ and $\mathbb{E}(Q_u) = \frac{|F_0|u}{p}$.

Now, using our observation from above, we can estimate the probability that $X_t^\#$ is less than u by the probability that $Q_u \geq t$. Since we know expectation and variance of Q_u, Chebyshev's inequality bounds the probability of $Q_u \geq t$. A similar reasoning also works for a lower bound on X_t.

Another observation, we can make is that $X_t^\# = X_t$ if h is injective. The probability that h is not injective is $\frac{1}{p} \leq \frac{1}{9}$ with the 2-universal Carter–Wegman hash family and the lower limit for p. The overall proof works by estimating the probabilities of the following events:
Case 1 The function h is not injective.
Case 2 Less than t elements are hashed to values below $v = \lfloor tp(1-\delta)^{-1}F_0^{-1} \rfloor$, i.e. $Q_v < t$.
Case 3 At least t elements are hashed to values below $u = \lceil tp(1+\delta)^{-1}F_0^{-1} \rceil$, i.e. $Q_u \geq t$.
and showing that the probability of each of these are strictly less than $\frac{1}{9}$. On the other hand if neither of these events occur, i.e., with probability at least $\frac{2}{3}$: h is injective and tpX_t^{-1} is an approximation of F_0 with a relative error of δ.

For the curious reader: This works for $t \geq 6\delta^{-2}$. The case, where $|A| < t$ – i.e., if there is no t-th smallest element – needs to be handled separately. But in that case, an implementation would simply return the count of distinct elements observed so far.

The above strategy requires $O(\delta^{-2} \ln n)$ bits of memory to store the state. Using a rounding operation, it is possible to improve the space complexity to $O(\delta^{-2}(\ln \delta^{-1} + \ln(\ln n)) + \ln n)$ bits of memory[20], which is the new rounding-KMV variant. Let us denote this rounding operation by $\lfloor \cdot \rfloor_r$, which can be defined by:

$$\lfloor x \rfloor_r := \tau(r - \lfloor \log_2 |x| \rfloor, x) \text{ where } \tau(e, x) = 2^{-e} \lfloor 2^e x \rfloor$$

if $x \neq 0$, otherwise: $\lfloor 0 \rfloor_r := 0$. In particular: $\lfloor x \rfloor_r \leq x < \lfloor x \rfloor_r + |x|(1 + 2^{-r})$. Expressed differently $\lfloor x \rfloor_r$ is the largest binary floating point number with a mantissa of at most r bits smaller or equal to x.

Now the new variant uses $\tilde{h}(a) = \lfloor h(a) \rfloor_r$ instead of h. The proof is similar to the above version, however the following issues need to be taken into account:

1. the additional accuracy error introduced by the rounding operation,
2. even if h is injective, collisions due to rounding are possible, i.e., $\tilde{h}$ may still not be injective and hence $\tilde{X}_t^\#$ may differ from $\tilde{X}_t$.

[20] Note that if $\delta^{-2} > n$, the trivial algorithm which tracks for each element of U whether it occurred in the stream using n bits outperforms either of these randomized algorithms. Hence, when judging which complexity is better it makes sense to assume $\delta^{-2} \in O(n)$.

The first issue can be solved by choosing $\tilde{r}$ and $\tilde{t}$ large enough such that the combined relative error due to rounding and the statistics of the t-th smallest element remain below δ. My initial idea for solving the second problem was to choose $\tilde{r}$ large enough, such that the probability of a collision due to rounding is bounded. It however turns out, that that condition would require a choice of $\tilde{r} \in O(\ln n)$, which is too high.

A closer look at the problem reveals that $\tilde{X}_t = \tilde{X}_t^{\#}$ as long as there is no collision within the smallest t hashed values. Estimating the probability of the latter event, requires another insight: It is actually not necessary to bound the probabilities of the events Case 1 to 3 separately.

In particular it is enough, if the probability of Case 2 and 3 happening is bounded by $\frac{2}{9}$ and if the probability of Case 1 happening under the condition that Case 2 and 3 are not is bounded by $\frac{1}{9}$.

Stated differently, it is enough to bound the probability of $\tilde{X}_t \neq \tilde{X}_t^{\#}$ only in the case $\tilde{X}_t^{\#} < v$. Thus it is enough to estimate the probability of a collision due to rounding within $[0, v)$ – the range the first t elements will be hashed to when $\tilde{Q}_v \geq t$. In the formalized proof, this is accomplished by making sure $\tilde{p} \geq 18$ and $\tilde{r} = 4\lceil \log_2(\delta^{-2}) \rceil + 23$ and bounding the probability that h is injective by $\frac{1}{18}$ and the probability of a collision due to rounding in the range $[0, v)$ by the same value.

In the accompanying formalization [33, Appendix A] I have included a detailed "hand-written" proof with the same reasoning as the formalized proof for interested readers.

3.2 Formalization of the F_0 algorithm

The following snippet contains the formalized version of the full algorithm:

```
fun f0-init :: rat ⇒ rat ⇒ nat ⇒ f0-space pmf where[6]
  f0-init δ ε n =
    do {
      let s = ⌈−18 * ln ε⌉;
      let t = ⌈80 / δ²⌉;
      let p = prime-above (max n 19);
      let r = 4 * ⌈log 2 (1 / δ)⌉ + 23;
      h ← prod-pmf {0..<s} (λ-. pmf-of-set (bounded-degree-polynomials (ZFact p) 2));
      return-pmf (s, t, p, r, h, (λ- ∈ {0..<s}. {}))
    }

fun f0-update :: nat ⇒ f0-space ⇒ f0-space 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-space ⇒ rat pmf where
  f0-result (s, t, p, r, h, sketch) = return-pmf (median s (λi ∈ {0..<s}.
      (if card (sketch i) < t then (card (sketch i)) else t * p / (Max (sketch i)))))
```

As explained in Subsection 2.3 the algorithm is formalized using three functions, an initialization function that sets up the state of the algorithm, an update function that updates the state processing a stream element and the result function that returns an estimate for the frequency moment using the state. The parameters of the initialization algorithm are: δ the required relative accuracy; ε the required success probability; and an upper bound n on the stream elements. As explained in Subsection 2.2 the algorithm runs $s = \lceil 18 \ln(\varepsilon^{-1}) \rceil$ independent copies of the rounding KMV algorithm to achieve the desired success probability, and computes the median of the individual estimates in the *f0-result* function. The algorithm

determines the t-th smallest hashed stream element, where $t = \lceil 80\delta^{-2} \rceil$. The function *prime-above* x returns a prime in the range $\{x, \ldots, 2x+2\}$ and is used to select the field over which the polynomials for hashing are chosen. The term *ZFact* p refers to the simple prime field $GF(p)$. The function *truncate-down* r is the rounding method $\lfloor \cdot \rfloor_r$ that was described above. To understand the algorithm a little bit better. The initialization function determines the parameters s, t, p, r and randomly selects s polynomials $h_0, \ldots, h_{s-1}$ of degree less than 2 over the field $GF(p)$. And sets up s empty sets, which will later contain the t smallest hashed stream elements. The state is a 6-tuple composed of the parameters, the hash functions and the sets. In the update step, for each of the hash polynomials, the function computes the rounded hashed value of the stream element and inserts the element into the corresponding set. If the set contains more than t elements, the largest element from the set will be removed. For the estimation, the function checks each of the s sets, if it contains at least t elements, it returns the inverse of the maximal element multiplied by tp, i.e., tp times the inverse of the t-th smallest element. If the set does not contain t elements – there is no t-th smallest element – but in that case the cardinality of the set is a good approximation of F_0 hence the cardinality is used as an approximation.

3.3 Frequency Moment 2

The formalized algorithm for the second frequency moment is based on the solution described in Section 2.2 by Alon et al. [3]. The key idea is to choose h from a 4-universal hash-family with (equiprobable) values $\{-1, 1\} \in \mathbb{Z}$. The algorithm then returns:

$$X = \left(\sum_{i=1}^{m} h(a_i) \right)^2 . \tag{6}$$

Note that:

$$\begin{aligned} \mathbb{E}\, X &= \mathbb{E} \left(\sum_{u \in U} C(u,a) h(u) \right)^2 = \sum_{u,v \in U} C(u,a) C(v,a)\, \mathbb{E}\, h(u) h(v) \\ &= \sum_{u \in U} C(u,a)^2\, \mathbb{E}\, h(u)^2 + \sum_{u,v \in U} C(u,a) C(v,a)\, \mathbb{E}\, h(u)\, \mathbb{E}\, h(v) = F_2 \end{aligned}$$

where $C(u,a)$ is the count of occurrences of u in the stream a. The last equality follows from $\mathbb{E}\, h(u)^2 = 1$ and $\mathbb{E}\, h(u) = 0$ for $u \in U$. An interesting fact is that the sum over $u, v \in U$ could only be evaluated by splitting it into two sums, where the first sum is comprised of the cases where the indices are equal and the second sum is comprised of the cases where the indices are distinct. This is because $h(u)$ and $h(v)$ are independent only if $u \neq v$.

A similar split has to be done for the evaluation of the variance of X where the summation is over four variables. This results in 15 possible ways the indices can form equivalence relations. For the latter I have built a library with which it is possible to automatically split such a sum into terms for each partition of its index variables [32, §5]. With that approach the estimation of the variance happens automatically through symbolic evaluation within Isabelle.

By running independent copies of the algorithm in parallel and computing the median of means, it is possible to return an approximation with the required success probability and accuracy.

The only difference between the method presented by Alon et al. [3] and the formalization in this work is that the algorithm is adapted to work with simple prime fields. While it is impossible to obtain a two-valued uniform distribution (if $p \geq n \geq 3$) a closer look at the proof from Alon et al. reveals that the actual requirements for the hash family are:

1. $\mathbb{E}\ h(a) = 0$, $\mathbb{E}\,(h(a))^2 = 1$ and $\mathbb{E}\,(h(a))^4 \leq 3$.
2. The hash family is 4-wise independent.

If h' is uniformly chosen from the 4-universal Carter–Wegman hash family then h defined by:

$$\begin{aligned} h &: U \to \mathbb{R} \\ h(x) &:= \begin{cases} (p^2-1)^{-\frac{1}{2}}(p-1) & \text{if } h'(x) \text{ is even} \\ (p^2-1)^{-\frac{1}{2}}(-p-1) & \text{otherwise} \end{cases} \end{aligned}$$

fulfills the above requirements. Since the factor $(p^2-1)^{-\frac{1}{2}}$ is constant, it can be factored out of the sum and the squaring operation in Equation 6, so that the resulting algorithm can be implemented without using real arithmetic.

3.4 Frequency Moment k for $k \geq 3$

The formalization of the algorithm for the k-th frequency moment for $k \geq 3$ is exactly the same as the solution described in Section 2.1 by Alon et al. [3]. Contrary to the previous algorithms it does not rely on hash families. Instead a random position $i \in \{1, \ldots, m\}$ of the stream $a_1, \ldots, a_m$ is selected and the count of occurrences of that stream element from that point on is counted, whose value is described by the following random variable:

$$X(i) = |\{j \in \{i, \ldots, m\} \mid a_j = a_i\}|$$

The estimate for the k-th frequency moment is then $R(i) := m(X(i)^k - (X(i)-1)^k)$. Note that:

$$\begin{aligned} \mathbb{E}\ R &= \mathbb{E}\,(m(X^k - (X-1)^k)) = \sum_{i=1}^{m} X(i)^k - (X(i)-1)^k \\ &= \sum_{u \in U} \sum_{v=1}^{C(u,a)} v^k - (v-1)^k = \sum_{u \in U} C(u,a)^k = F_k, \end{aligned}$$

where $C(u,a)$ denotes the count of occurrences of u in the stream $a_1, \ldots, a_m$. The evaluation of the variance of the random variable is a longer calculation resulting in $\text{Var}\ R \leq F_k^2 k n^{1-1/k}$. Similar to the previous algorithm by running independent copies of the algorithm in parallel and computing the median of means, it is possible to improve the accuracy and success probability.

A remaining problem to solve is that the algorithm has to choose a random index uniformly from the stream, without knowing the length of the stream in advance.

Alon et al. [3] describe a refined version of the algorithm that solves the problem: A random boolean is chosen at every update step, which is true with probability $\frac{1}{l+1}$, where l is the number of elements that were processed before. If the boolean is true the algorithm resets the counter to 1 and chooses the element at the current index to count. They then show, that the position of the last reset is uniformly distributed over the length of the stream. The accompanying formalization [33] verifies this second version of the algorithm.

4 Related Work

In 2019 Affeldt et al. [1] formalized two tree-based succinct data structures in Coq, one of them being dynamic. They achieve their results by defining the operations on a high-level inductive data structure and a low-level version implemented on bit arrays and establish correspondence. A similar approach could also be applied here to avoid the need of the encoding functions as discussed in Subsection 2.5.

Eßmann et al. [18] formally verify non-deterministic approximation algorithms for NP-complete problems in Isabelle, such as maximum independent set. In their work, they do not need to reason with probability distributions, as the correctness of the investigated algorithms follows from combinatorial arguments.

In 2020 Gopinathan and Sergey [23] formally verified Bloom Filters using Coq. They rely on a deep embedding and similar to this work rely on probability theory and reasoning about independent random variables.

Tassarotti et al. [46] formally verify an ML procedure learning a classifier using Lean. As in this work, they also represent their algorithms using the Giry-monad.

Bao et al. [4] also tackle Bloom Filters as an application of their separation logic for reasoning about negative dependence. Negative dependence is a weaker property about random variables than independence, i.e., more sets of random variables are negatively dependent, but fewer results about independent random variables apply to negatively dependent variables. For example the property is preserved by composition with monotone functions only. They realize that the random variables induced on the bit vector are negatively dependent; greatly simplifying the proofs about the false-positive rate of bloom filters.

Eberl et al. [15] verify the average runtime of randomized quicksort and derive the expected structure of binary tree structures. They rely on the formalization of the Giry monad in Isabelle/HOL. The formalization approach for randomized algorithms in this work is based on their work.

As far as I can tell there is no prior publication on the formalization of randomized algorithms where derandomization and/or amplification techniques are used.

5 Conclusion and Future Work

While the primary focus of this work, was the formal verification of the algorithms for frequency moments – I could obtain simpler versions of the known algorithms. In particular it was possible to avoid higher order prime fields. The algorithm for F_0 matches the complexity of the best solution from [5] but its design is considerably simpler. Requiring only one hash family instead of two. Most solutions in current production database systems are verified empirically and/or rely on unverified statistical model assumptions, because of the complexity of the known correct solutions [26]. These simplifications may lead to industrial applications of these rigorously verified algorithms.

An interesting direction for future work would be the formalization of the newer results that match the lower bounds [28, 30]. Another interesting problem is the extension of the frequency moments to fractional powers, for which algorithms have been derived in [13, 29].

The algorithms presented here for F_0 and F_2 can be augmented with a merge operation, i.e., it would be possible to run the algorithm in parallel for different sections of the data stream and merge the sketches to obtain an approximation of the frequency moment for the entire stream. It would make sense to extend the obtained theorems to include the merge operation.

Another improvement would be to use probable primes[21] instead of requiring exact primes. For this, the algorithms would need to be reparameterized such that the combined failure probability of the algorithm and the false-positive rate of the primality test remains below the required maximum failure probability ε.

[21] Probabilistic primality tests have been formalized in Isabelle by Stüwe and Eberl [45].

References

1 Reynald Affeldt, Jacques Garrigue, Xuanrui Qi, and Kazunari Tanaka. Proving Tree Algorithms for Succinct Data Structures. In *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITP.2019.5.

2 Archive of Formal Proofs. https://isa-afp.org. Accessed: 2021-11-13.

3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

4 Jialu Bao, Marco Gaboardi, Justin Hsu, and Joseph Tassarotti. A separation logic for negative dependence. *Proceedings of the ACM on Programming Languages*, 6:57:1–57:29, 2022. doi:10.1145/3498719.

5 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques in Computer Science*, pages 1–10. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45726-7_1.

6 Kevin Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 199–210, New York, 2007. doi:10.1145/1247480.1247504.

7 Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 708–713, USA, 2006. Society for Industrial and Applied Mathematics. doi:10.5555/1109557.1109634.

8 Julian Biendarra and Manuel Eberl. Bertrand's postulate. *Archive of Formal Proofs*, January 2017. , Formal proof development. URL: https://isa-afp.org/entries/Bertrands_Postulate.html.

9 Jarosław Błasiok. Optimal streaming and tracking distinct elements with high probability. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2432–2448, 2018. doi:10.1137/1.9781611975031.156.

10 Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). RFC 8949, December 2020. doi:10.17487/RFC8949.

11 R. C. Bose and K. A. Bush. Orthogonal arrays of strength two and three. *The Annals of Mathematical Statistics*, 23(4):508–524, 1952. doi:10.1214/AOMS/1177729331.

12 Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An Optimal Algorithm for Large Frequency Moments Using $O(n^{1-2/k})$ Bits. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 531–544, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.531.

13 Vladimir Braverman, Emanuele Viola, David P. Woodruff, and Lin F. Yang. Revisiting Frequency Moment Estimation in Random Order Streams. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:14, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.25.

14 David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and Sridhar Seshadri. Practical skew handling in parallel joins. In *Proceedings of the 18th VLDB conference*, 1992.

15 Manuel Eberl, Max W. Haslbeck, and Tobias Nipkow. Verified analysis of random binary tree structures. *J. Autom. Reason.*, 64(5):879–910, 2020. doi:10.1007/s10817-020-09545-0.

16 Manuel Eberl, Johannes Hölzl, and Tobias Nipkow. A verified compiler for probability density functions. In *Programming Languages and Systems*, pages 80–104. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-46669-8_4.

17 P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975. doi:10.1109/TIT.1975.1055349.

18 Robin Eßmann, Tobias Nipkow, and Simon Robillard. Verified approximation algorithms. *Automated Reasoning*, 12167:291–306, 2020. doi:10.46298/lmcs-18(1:36)2022.

19 Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science*, pages 137–156, 2007.

20 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

21 Phillip B. Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '01, pages 281–291, 2001. doi:10.1145/378580.378687.

22 Frédéric Giroire. Order statistics and estimating cardinalities of massive data sets. *Discrete Applied Mathematics*, 157(2):406–427, 2009. doi:10.1016/j.dam.2008.06.020.

23 Kiran Gopinathan and Ilya Sergey. Certifying certainty and uncertainty in approximate membership query structures. *Computer Aided Verification*, 12225:279–303, 2020. doi:10.1007/978-3-030-53291-8_16.

24 Sebastien Gouezel. Lp spaces. *Archive of Formal Proofs*, October 2016. , Formal proof development. URL: https://isa-afp.org/entries/Lp.html.

25 Benjamin Gufler., Nikolaus Augsten., Angelika Reiser., and Alfons Kemper. Handling data skew in MapReduce. In *Proceedings of the 1st International Conference on Cloud Computing and Services Science - CLOSER*, pages 574–583. INSTICC, SciTePress, 2011. doi:10.5220/0003391105740583.

26 Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 683–692, New York, 2013. ACM. doi:10.1145/2452376.2452456.

27 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1007/978-1-4612-0865-5_26.

28 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 202–208, New York, 2005. doi:10.1145/1060590.1060621.

29 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1161–1178, USA, 2010. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973075.93.

30 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, 2010. doi:10.1145/1807085.1807094.

31 Emin Karayel. A combinator library for prefix-free codes. *Archive of Formal Proofs*, April 2022. , Formal proof development. URL: https://isa-afp.org/entries/Prefix_Free_Code_Combinators.html.

32 Emin Karayel. Enumeration of equivalence relations. *Archive of Formal Proofs*, February 2022. , Formal proof development. URL: https://isa-afp.org/entries/Equivalence_Relation_Enumeration.html.

33 Emin Karayel. Formalization of randomized approximation algorithms for frequency moments. *Archive of Formal Proofs*, April 2022. , Formal proof development. URL: https://isa-afp.org/entries/Frequency_Moments.html.

34 Emin Karayel. Interpolation polynomials (in hol-algebra). *Archive of Formal Proofs*, January 2022. , Formal proof development. URL: `https://isa-afp.org/entries/Interpolation_Polynomials_HOL_Algebra.html`.

35 Emin Karayel. Median method. *Archive of Formal Proofs*, January 2022. , Formal proof development. URL: `https://isa-afp.org/entries/Median_Method.html`.

36 Emin Karayel. Universal hash families. *Archive of Formal Proofs*, February 2022. , Formal proof development. URL: `https://isa-afp.org/entries/Universal_Hash_Families.html`.

37 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978. `doi:10.1145/359619.359627`.

38 R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson. Optimal normal bases in GF(p^n). *Discrete Applied Mathematics*, 22(2):149–161, 1988. `doi:10.1016/0166-218X(88)90090-X`.

39 Magnus O. Myreen. The CakeML Project's Quest for Ever Stronger Correctness Theorems. In *12th International Conference on Interactive Theorem Proving (ITP 2021)*, volume 193 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:10, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ITP.2021.1`.

40 Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, 1 edition, 2002.

41 Vaughan R. Pratt. Shellsort and sorting networks. In *Outstanding Dissertations in the Computer Sciences*, 1972.

42 Pedro Reviriego, Alfonso Sanchez-Macian, Shanshan Liu, and Fabrizio Lombardi. On the security of the k minimum values (KMV) sketch. *IEEE Transactions on Dependable and Secure Computing*, 2021. `doi:10.1109/TDSC.2021.3101280`.

43 Joseph H. Silverman. Fast multiplication in finite fields GF(2^n). In *Cryptographic Hardware and Embedded Systems*, pages 122–134. Springer Berlin Heidelberg, 1999. `doi:10.1007/3-540-48059-5_12`.

44 Hagen Sparka, Florian Tschorsch, and Björn Scheuermann. P2KMV: A privacy-preserving counting sketch for efficient and accurate set intersection cardinality estimations. Cryptology ePrint Archive, Report 2018/234, 2018. `doi:10.14279/DEPOSITONCE-8374`.

45 Daniel Stüwe and Manuel Eberl. Probabilistic primality testing. *Archive of Formal Proofs*, February 2019. , Formal proof development. URL: `https://isa-afp.org/entries/Probabilistic_Prime_Tests.html`.

46 Joseph Tassarotti, Koundinya Vajjha, Anindya Banerjee, and Jean-Baptiste Tristan. A formal proof of PAC learnability for decision stumps. In *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 5–17, 2021. `doi:10.1145/3437992.3439917`.

47 Mikkel Thorup and Yin 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. `doi:10.1137/1.9781611972900.7`.

48 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1-3):1–336, 2012. `doi:10.1561/0400000010`.

49 Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. `doi:10.1016/0022-0000(81)90033-7`.

50 Yang Yang, Ying Zhang, Wenjie Zhang, and Zengfeng Huang. GB-KMV: An augmented KMV sketch for approximate containment similarity search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 458–469, 2019. `doi:10.1109/ICDE.2019.00048`.