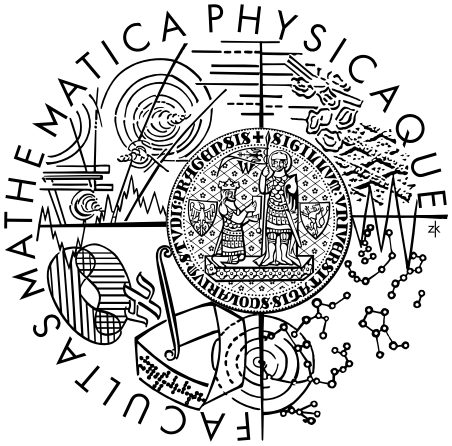
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Martin Babka

Properties of Universal Hashing

Department of Theoretical Computer Science
and Mathematical Logic

Supervisor: doc. RNDr. Václav Koubek, DrSc.

Study programme: Theoretical Computer Science

2010

At this place, I would like to thank my supervisor doc. RNDr. Václav Koubek, DrSc. for the invaluable advice he gave to me, for the great amount of knowledge he shared and time he generously spent helping me when working on the thesis. I would also like to say thank you to my friends and family.

I hereby proclaim, that I worked out this master thesis myself, using only the cited resources. I agree that the thesis may be publicly available.

In Prague on 16$^{\text{th}}$ April, 2010 Martin Babka

Contents

Názov práce: Vlastnosti univerzálneho hashovania
Autor: Martin Babka
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedúci bakalárskej práce: doc. RNDr. Václav Koubek, DrSc.
E-mail vedúceho: vaclav.koubek@mff.cuni.cz

Abstrakt: Cieľom tejto práce je navrhnúť model hashovania s akceptovateľnou časovou zložitosťou aj v najhoršom prípade. Vytvorený model je založený na princípe univerzálneho hashovania a výsledná očakávaná časová zložitosť je konštantná. Ako systém univerzálnych funkcií sme zvolili množinu všetkých lineárnych zobrazení medzi dvomi vektorovými priestormi. Tento systém už bol študovaný s výsledkom predpovedajúcim dobrý asymptotický odhad. Táto práca nadväzuje na predchádzajúci výsledok a rozširuje ho. Tiež ukazuje, že očakávaný amortizovaný čas navrhnutej schémy je konštantný.

Kľúčové slová: univerzálne hashovanie, lineárne zobrazenie, amortizovaná časová zložitosť

Title: Properties of Universal Hashing
Author: Martin Babka
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: doc. RNDr. Václav Koubek, DrSc.
Supervisor's e-mail address: vaclav.koubek@mff.cuni.cz

Abstract: The aim of this work is to create a model of hashing with an acceptable worst case time complexity. The model is based on the idea of universal hashing and the expected time of every operation is constant. The used universal class of functions consists of linear transformations between vector spaces. This class has already been studied with a good asymptotic result. This work improves the result and proves that the expected amortised time of the scheme is constant.

Keywords: universal hashing, linear transformation, amortised time complexity

Chapter 1

Introduction

One of the successful tasks done by the computers today is the data storage and its representation. By this term we not only mean the database management systems storing large amounts of data. This term includes various forms of the *set representation problem* [23] – representation and storing sets of elements. There are a lot of approaches, that solve this problem and deal with the situations from representing small sets stored in the main memory to enormous databases stored distributively on distinct systems.

Solving this problem has many apparent practical applications from databases in business environment to more hidden ones. Consider bus stops of a city represented by objects of a programming language. Think about solving the shortest path problem by graph library. This library does not provide an association of the library's node object with its city. But you need a fast mapping of the nodes onto stops to build the path. This mapping is usually provided by the data structures solving the set representation problem. Nowadays these data structures are commonly present as HashTables, ArrayLists or Dictionaries in the standard libraries of many current programming languages. Their fast implementation plays an important role in the quality of solutions provided by programmers.

The set representation problem, also known as the *dictionary problem*, is a storage of a set consisting of chosen elements of a universe. The universe includes all the representable elements. Basic operations provided by a *data structure* solving this problem allow a modification of the stored set and querying if an element is stored within it.

- $Member(x)$ – returns true if the element x is contained by the represented set.
- $Access(x)$ – returns the data associated with the element x if it is contained within the stored set.
- $Insert(x)$ – adds the element into the stored set.
- $Delete(x)$ – removes the element from the stored set.

There are plenty of different data structures solving the problems providing us with various running times of these operations [8]. If our algorithms prefer some

operations to the others, we can gain an asymptotic improvement in their running times by a good choice of the underlying data structure.

The data structures can also guarantee the worst case running time of every operation. Other promise to be quick in the average case but their worst case running time can be linear with the number of elements stored. For example the running time of balanced trees, AVL trees [17], red-black trees [18] or B-trees [5] is logarithmic for every operation. Hash tables [25], [23], [6], [22] should run in $O(1)$ expected time but we may get unlucky when iterating a chain of large length. The most simple data structures such as arrays or lists are preferred because they have certain operations running in constant times but the other are definitely linear.

The warranties, provided by the data structures, mean a trade off with the expected times. For instance binary search trees are expected to be faster in the expected case when assuming uniformity of the input than red black trees or AVL trees. These comparisons are discussed in [4] and later results in [30] or [14]. If the uniformity of input may not be assumed we are made to use the conservative solution.

There are special data structures, such as heaps, that do not implement the basic operations effectively. We can not consider it as a flaw, their use is also special. In the case of the heaps it means efficient finding of minimal or maximal element of the set. On the other hand some data structures may allow other operations.

- $Ord(k)$, $k \in \mathbb{N}$ – returns the k^{th} element of the stored set.
- $Pred(x)$, $Succ(x)$ – returns the predecessor or successor of x in the set.
- $Min(x)$, $Max(x)$ – returns the minimal or maximal element of the stored set.

In this work we try to design a hash table which has the constant expected running time of every operation. In addition it also guarantees a reasonable worst case bound, $O(\log n \log \log n)$. We can not expect running it as fast as classic hashing when assuming the input's uniformity. The computations indicate that it runs much faster than balanced or binary search trees.

The concept of hashing is introduced in Chapter 2. It continues by description of different models of hashing and finally mentions current approaches and fields of interests of many authors. In the third chapter the principle of universal hashing is discussed. It also introduces many universal classes of functions and states their basic properties. In Chapter 4 we show how to compute the expected length of the longest chain and discuss its importance. The work later continues by the chapter where we show the already known results regarding the system of linear transformations. In the following chapters we improve the results obtained when using universal hashing with the system. In the last chapter, we show the properties of the proposed model based on the new results.

Chapter 2

Hashing

Hashing is one of the approaches designed for the efficient solving of the dictionary problem. Various implementations differ in many ways. However usage of a hash function and a quickly accessible table, typically represented by an array, is common to most of them. Every hash function transforms the elements of the universe into the addresses of the table.

Represented sets are always small when compared to the size of the universe. In order to prevent waste of space we are forced to use tables as small as possible. Typically the size of the hash table containing the represented elements is chosen so that it is comparable to the size of the stored set.

The fact that the hash table is much smaller than the universe and any two elements may represented means that the elements may share the same address after hashing. This event is called a *collision*. For hashing it is very important how collisions are handled. This is also the most interesting distinctive feature for distinct hashing models. In fact, collision handling is crucial when determining the time complexity of the scheme.

When two elements collide they should be stored in a single *bucket*, cell of the hash table. A bucket is often represented by the simplest data structure possible. For instance, a singly linked list should be sufficient in many cases. More sophisticated schemes, like perfect hashing, represent every chain in another hash table without collisions.

The find operation of a hash table works in the following way. Every element is placed as an argument for the hash function. The element's address is then computed and used as an index of the hash table. Then we look into the bucket lying at the returned address if the element is stored inside or not.

If buckets are represented by linked lists, then the expected time of the find operation is proportional to one half of the length of the list.

Lemma 2.1. *Let S be a set represented by a linked list. Assume that for every element $x \in S$:*

$$\mathbf{Pr}\,(x \textit{ is argument of the find operation}) = \frac{1}{|S|}.$$

Then the expected time of the find operation is $\frac{|S|+1}{2}$.

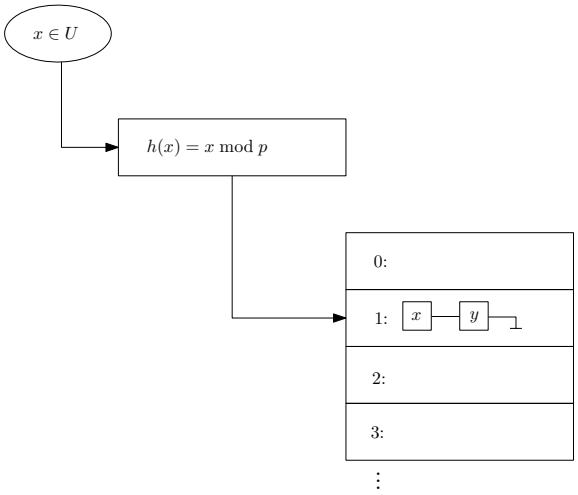


Figure 2.1: Concept of a hash table.

Proof. Let $x_i \in S$ be the i-th element of the list, $1 \leq i \leq |S|$. Time to find the element x_i inside the list equals i. The expected time of the find operation can be expressed directly from its definition

$$\begin{aligned}\mathbf{E}\,(\text{time of the find operation}) &= \sum_{i=1}^{|S|} i\mathbf{Pr}\,(x_i \text{ is the argument of the find operation}) \\ &= \frac{\sum_{i=1}^{|S|} i}{|S|} = \frac{|S|(|S|+1)}{2|S|} = \frac{|S|+1}{2}.\end{aligned}$$

□

As seen in the previous lemma the time complexity of an operation needs not to be measured by its worst case time. Compare $|S|$, which is the worst case, to the expected value $\frac{|S|+1}{2}$ which is better. Considering only the worst case times does not tell much about the structure's real behaviour. We should use probability based characteristics that give more accurate results. These characteristics include the expected operation's time or its expected worst case time which is usually far more difficult to analyse. For hashing this difference means an asymptotic improvement.

2.1 Formalisms and Notation

Notation and formalisms that mathematically describe a hash table are crucial part of an exact analysis. Assume that we are hashing a *universe* U on a *hash table* of size m. The table consists of buckets each having its unique address. The addresses are usually quite simple. When the table consists of m buckets the addresses are just $0, \ldots, m-1$. Buckets are always identified by their addresses and thus we can refer to the set $B = \{0, \ldots, m-1\}$ as a hash table.

Universe consists of all the representable elements, examples include the objects of a programming language, strings over an alphabet, numbers or anything else. Hash

function is a way to transform these elements into an address of the table, usually a natural number. Hash function h can be described by a function $h : U \to B$. The other requirements placed on the function h are discussed later.

The letter S typically denotes the *stored set*. We often refer to the variable n as to the size of the stored set, $n = |S|$. As already mentioned we always assume that $S \subset U$ and $|S| \ll |U|$.

Since the universes are really large, recall the previous examples, sizes of the tables are much smaller when compared to those of the universes. Space waste is typically caused by allocating a large table containing many empty buckets. The definition of the table's *load factor* allows an exact analysis of the phenomena connected with the table's filling – its performance or waste of space.

Definition 2.2 (Load factor)**.** *Let n be the size of the represented set and m be the size of the hash table. The variable α defined as*

$$\alpha = \frac{n}{m}$$

is called the load factor *of the table.*

To take control of the table's overhead it is sufficient to keep the load factor in a predefined interval. Not only the empty buckets cause troubles. The overfilled ones are another extreme especially if there are too many collisions.

Definition 2.3 (Collision)**.** *Let h be a hash function and $x, y \in U$ be two distinct elements. We say that the elements x and y* collide *if $h(x) = h(y)$.*

Above all the collision handling is what differentiates distinct hash tables and determines their performance.

To completely explain the notation we have to mention that we refer to the function log as to the binary logarithm. The function ln denotes the natural logarithm.

2.2 Assumptions of Classic Hashing

In Lemma 2.1 we computed the expected time of the find operation and used it as a measure of its time complexity. Known probability distribution of the input enables us to compute the expected value. For the lemma we assumed the uniform distribution. In hashing similar situation occurs. If we want to find the expected results, then we need corresponding probabilistic assumptions on the input. The assumptions we make may be found in [23] or in [24]. They are similar to ours.

- Hash function h distributes the elements of universe uniformly across the hash table:
$$||h^{-1}(x)| - |h^{-1}(y)|| \leq 1 \text{ for every } x, y \in B.$$
- Every set has the same probability of being stored among the sets of the same size:
$$\mathbf{Pr}\left(S \text{ is stored}\right) = \frac{1}{\binom{|U|}{|S|}} \text{ for every set } S \subset U.$$

- Every element of the universe has the same probability of being an argument of an operation:

$$\mathbf{Pr}\,(x \text{ is used as an argument of an operation}) = \frac{1}{|U|} \text{ for every } x \in U.$$

These assumptions provide us with a probabilistic model for the average case analysis of classic hashing.

2.3 Separate Chaining

Separate chaining [22], [24] and [25] may be considered the most basic hash table implementation. However, it provides a sufficient framework for illustration of various problems and analyses. Separate chaining usually utilises one hash function mapping elements to an address – an index of the array representing the hash table. Every represented element is stored within a bucket given by its hash address. Every bucket is represented by a singly linked list.

Separate chaining, like the other classic models of hashing, is quite dependent on the stored input. The previous assumptions are required for its average case analysis. This model assumes hashing of the universe $U = \{0, \ldots, N-1\}$ for $N \in \mathbb{N}$ into a table of size m. Number m is much smaller then N and moreover it is chosen to be a prime. Primality of m improves the ability of the hash function to uniformly distribute the stored set across the table. The following example clarifies why primality is important for the choice of m.

Example 2.4. *Consider the set* $S = \{5n \mid n \in \{0, \ldots, 5\}\}$ *and a table consisting of* 10 *buckets. Set* S *contains only 6 elements, so the table's load factor equals* 0.6*. If we use function* $x \bmod 10$*, then it maps the elements of the set* S *just onto two addresses,* 0 *and* 5*. The hash table is not used effectively. Choosing* m *to be a prime number improves the results but is not a remedy.*

The apparent disadvantage of classic hashing is usage of a single hash function known in advance. For example if the stored set S is chosen, by an adversary, so that $S \subseteq f^{-1}(b)$ for a single bucket $b \in B$, we run into problems. Because our assumptions say that the probability of having such an input is quite low, these inputs may be neglected.

To illustrate how separated chaining works the algorithms are presented. Although the model is quite simple the subroutines regarding the singly linked lists are not discussed. They are considered elementary enough. We remark that the running time is proportional to length of the represented chain. Also notice that the insert procedure has to check whether the element is already stored inside the list else it can be stored twice.

Algorithm 1 Find operation of the separate chaining.

Require: $x \in U$
  $h = h(x)$

  if $T[h]$ contains x **then**
    return true {operation is successful}
  else
    return false {operation is unsuccessful}
  end if

Algorithm 2 Insert operation of the separate chaining.

Require: $x \in U$
  $h = h(x)$

  if $T[h]$ does not contain x **then**
    insert x into chain $T[h]$
  end if

Algorithm 3 Delete operation of the separate chaining.

Require: $x \in U$
  $h = h(x)$

  if $T[h]$ contains x **then**
    remove x from chain $T[h]$
  end if

Theorem 2.5 (Average case of the separate chaining)**.** *Let $\alpha > 0$ be the load factor of the hash table resolving collisions by separate chaining. Then the expected time of the successful find operation converges to $1 + \frac{\alpha}{2}$ and to $e^{-\alpha} + \alpha$ in the unsuccessful case.*

Proof. The theorem is proved in [24]. □

Various modifications of separate chaining are known. If the universe is an ordered set then ordering the elements in chains monotonously may bring a speedup. This is caused by the fact that the operations concerning the chains do not have to iterate throughout the whole list and stop earlier.

There is a modification utilising the move to front principle [2]. This principle is motivated by the rule of the locality of reference [1]. The rule says that whenever an element is accessed, it is very likely that it is going to be accessed again in a short future. Therefore it is quite convenient to put the last accessed element to the front of the list so that it is accessed fast.

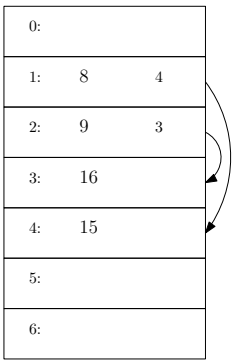


Figure 2.2: Singly linked lists represented directly in the hash table. The first number is the bucket's address, the second is the represented key and the last one is the next pointer.

Another set of modifications changes the representation of chains. They are no longer represented by common linked lists since their cache behaviour [20] is poor as stated in [32]. The chains are stored directly in the hash table. So when resolving a collision an empty place in the hash table is taken. There are problems connected with this approach that need to be solved. Above all chains may not be merged together. This appears when a new chain should be started but the place of the first element is already taken. It may be solved by moving the element, not belonging to the chain, to another empty place. To perform the replacement quickly we are forced to use double linkage. Or, instead of using doubly linked lists, at every address an additional begin pointer may point to the first element of the chain. In addition, we have to deal with the fact that schemes using only the hash table suffer from bad behaviour when the load factor is almost one. And of course a uniform random choice of an empty bucket has to be provided when prolonging a chain.

2.4 Coalesced Hashing

In coalesced hashing, the chains of colliding elements are also stored directly in the hash table. However, they are allowed to fuse together. Every chain is represented by a singly linked list. The simpler methods such as LISCH and EISCH do not use any special additional memory. In addition, the EISCH method utilises the move to front rule. Methods such as LICH, EICH and VICH use the additional memory which is not directly addressable by the hash function. Collision resolution uses the additional memory to store growing chains. When an empty place for a colliding element is needed, it is first sought in the additional memory. Only if the special memory is full, then the addressable memory is used. The methods are distinct in a way they use the move to front rule. The EICH method always obeys it, VICH uses it only in the addressable memory and the LICH method does not obey the rule at all.

2.5 Open Addressing

Methods of coalesced hashing and separate chaining represent the chains explicitly by linked lists. The additional pointer of the list not only consumes an extra memory. It also worsens the cache behaviour of the hash table. Open addressing also merges the chains together. They are stored directly in the hash table and their representation is more implicit. To remove the need of an additional pointer, a secondary function for conflict resolution is used.

Models of open addressing assume existence of two hash functions $h_1 : U \to B$ and $h_2 : U \to B$. Assume $x \in U$ is the i^{th} element of the chain and the order of elements in a chain starts from zero. Its address is then determined by a compound hash function $h(x, i) = h_1(x) + ih_2(x)$. When we insert the element $x \in U$ we find a minimal i such that the bucket at the address $h(x, i)$ is empty.

This approach also removes the problem of the choice of a free bucket when a collision occurs. Unfortunately, there is no fast implementation of the delete operation known so far. To delete an element we can mark its bucket as free and reuse it when possible. Marking the deleted element's bucket is necessary if we do not want to accidentally loose the elements lying behind the deleted one in the chains.

2.5.1 Linear Probing

Linear probing is a scheme of open addressing with $h_2(x) = 1$ for every $x \in U$. The last element of the prolonged chain is placed into the first empty bucket behind the one with the address $h_1(x)$.

Another problem of linear hashing is its degradation. Clusters of non-empty buckets emerge when load factors reach one. Delete operation implemented by marking only emphasises this problem. Despite of the mentioned problems, linear probing takes the advantage of the good cache behaviour. No wonder, that its modifications are studied nowadays, e.g hopscotch hashing mentioned in Section 2.8.

2.5.2 Double Hashing

If we want to distribute the stored set uniformly across the hash table, then the choice of a free bucket should be random. Whenever function $h(x, i)$ of argument i is a random permutation of B, then the mentioned choice becomes random. A necessary condition is that $h_2(x)$ and m are relatively prime. Therefore the size of the hash table is chosen so that it is a prime number again.

Double hashing has theoretical analyses yielding remarkable results.

Theorem 2.6. *Let $\alpha \in (0, 1)$ be the load factor of the hash table resolving collisions by double hashing. Then the running time of the find operation converges to $\frac{1}{1-\alpha}$ in the unsuccessful case and to $\frac{1}{\alpha} \log\left(\frac{1}{1-\alpha}\right)$ when the operation is successful.*

Proof. The proof is shown in [24]. □

2.6 Universal Hashing

Universal hashing uses a universal system of functions instead of a single function. This removes the dependence of universal hashing on the uniformity of its input. It does not require the assumptions of standard hashing made in Section 2.2. Universal hashing is a randomised algorithm and the probability space is determined by the uniform choice of a hash function, instead of the selection of a stored set. We study universal hashing and its systems in a more detailed way in Chapter 3.

2.7 Perfect Hashing

Assume that the stored set S is known in advance. The problem solved by perfect hashing is how to create a hash table and a hash function such that the find operation takes only a constant time. Insert and delete operations are forbidden, the stored set S remains fixed. Additional requirements are placed on the size of the scheme. Size of the hash table is not the only problem. We must also take care about the space needed to represent the constructed hash function since it becomes quite complex. The problem is solved in [16].

2.8 Modern Approaches

Modern approaches to hashing are nowadays often based on their probabilistic properties. Authors adapt and change their models to improve the asymptotic results in the average case and in the expected worst case. The algorithms still remain fast and are enriched by simple rules.

Straightforward use of a single function is not enough and various universal systems are used. Theoretical analyses of the models do not necessarily involve the ideas of universal hashing. Universal classes are used as a heuristic. Although the algorithms are not complicated, their analyses become more and more difficult.

A model called *robin hood hashing* [7], [10] is an improvement of the double hashing. In double hashing we are able to access the chain at an arbitrary position in constant time. The main idea of robin hood hashing is to minimise the variance of the expected chain length. If probing starts from the average chain length the expected time of the find operation becomes constant.

Another model, *hopscotch hashing* [21], is a modification of the linear probing utilising the computer's cache behaviour. A chain is kept together in one place as long as possible. Algorithms in control of the processor cache store whole blocks of memory inside it when only a single part of the block is accessed. Provided that the chains are compact, storing them in the cache is quite probable. This optimisation makes probing substantially faster.

Next investigated model is *cuckoo hashing* [9]. However, the interest fades these days since there are some problems connected with it [13], [12].

2.9 Advantages and Disadvantages

Hashing, compared to other data structures solving the dictionary problem, e.g. trees, does not provide the complete set of operations. If the universe is ordered, we can not query the hash table for the k^{th} stored element. The hash functions are supposed to distribute the elements across the table evenly. This assumption is, somehow, in the opposition to the preservation of the ordering. Especially, different functions of a universal class can not preserve the ordering, they are required to be different. So in general, hashing does not provide the operations based or related to ordering, such as *Succ*, *Pred*, *Order* of an element and the already mentioned operation – k^{th} stored element. Despite this, it can also be a rare advantage. Trees require an ordering of the elements they store, so when the universe is not naturally ordered a problem occurs. Usually we solve the problem by adding a unique identifier to every element before storing. This consumes a little memory and in distributed environments causes small troubles with uniqueness, too.

The obvious advantage of hashing is its simplicity. In addition, its constant expected time of the find operation may seem far better than the logarithmic time of trees. On the other hand, if provided with an unsuitable, input hashing often degrades. Balanced trees provide warranties for every input they are provided with. This is a tradeoff that has to be decided by the user – excellent expected and poor worst case time compared to the logarithmic time in every case.

To sum up, when we need a fast and simple to implement mapping of an object to another one, hashing is a good solution. Especially, when the assumptions from Section 2.2 are satisfied. These conditions are often met with objects having artificial identifiers. However, the worst case guarantee is poor. Despite this, the expected worst case warranty is similar to that of trees, as shown later in the work. Nowadays simplicity does not mean any extra advantage. Standard libraries of current programming languages provide us with many implementations of hashing and trees.

Chapter 3

Universal Classes of Functions

Universal classes of functions play an important role in hashing since they improve independence of the associative memory on its input. Their greatest advantage is removal of the two assumptions of classic hashing – uniformity of input. The first one states that the probability of storing a set is distributed uniformly among sets having the same size. The second one says that every element of the universe has the same probability of being hashed.

In addition, models of standard hashing take probability of collision relative to the choice of the stored set. In models of universal hashing probability space corresponds to the choice of a random function selected from the universal class. Collision is caused by an inconvenient choice of a universal function more than by an unsuitable input. Notice that the relation between a collision and the input, the represented set, is not so tight as in the classic hashing. Another advantage of universal hashing is that any obtained bound holds for every stored set.

Probability that the selected universal function is not suitable for a stored set is low. So every function from a universal system is suitable for many stored sets. If the selected function is not suitable for a stored set we are free to pick another one. Rehashing the table using a new function is not likely to fail. The probability of failure for several independent choices of a hash function decreases exponentially with respect to the number of choices.

3.1 Universal classes of functions

Various definitions of universal classes may be found in [6], [23], [25] and [24]. Their various application are shown in [34] and [27]. Originally for strongly n-universal classes the equality in the probability estimate holds in [34]. Various situations may satisfy only inequality and therefore their definition is a bit changed in this work. We also define nearly strongly n-universal classes.

Definition 3.1 (c-universal class of hash functions)**.** *Let $c \in R$ be a positive number and H be a multiset of functions $h : U \to B$ such that for each pair of elements*

$x, y \in U$, $x \neq y$ *we have*

$$|\{h \in H \mid h(x) = h(y)\}| \leq \frac{c|H|}{|B|}$$

where the set on the left side is also considered a multiset. The system of functions H *is called* c-universal class *of hash functions.*

First notice that the bound on the number of colliding functions for each pair of elements is proportional to the size of the class H. More interesting is the inverse proportionality to the size of the hash table. A generalisation of the above definition for more than two elements provides us with a better estimate on the number of colliding functions. This count becomes inversely proportional to the power of the table size.

Definition 3.2 (Nearly strongly n-universal class of hash functions)**.** *Let* $n \in \mathbb{N}$, $c \in \mathbb{R}, c \geq 1$ *and* H *be a multiset of functions* $h : U \to B$ *such that for every choice of* n *different elements* $x_1, x_2, \ldots, x_n \in U$ *and* n *elements* $y_1, y_2, \ldots, y_n \in B$ *the following inequality holds*

$$|\{h \in H \mid h(x_1) = y_1, h(x_2) = y_2, \ldots, h(x_n) = y_n\}| \leq \frac{c|H|}{|B|^n}$$

where the set on the left side is also considered a multiset. The system of functions H *is called* nearly strongly n-universal class *of hash functions with constant* c.

Definition 3.3 (Strongly n-universal class of hash functions)**.** *Let* $n \in \mathbb{N}$ *and* H *be a multiset of functions* $h : U \to B$ *such that for every choice of* n *different elements* $x_1, x_2, \ldots, x_n \in U$ *and* n *elements* $y_1, y_2, \ldots, y_n \in B$ *the following inequality holds*

$$|\{h \in H \mid h(x_1) = y_1, h(x_2) = y_2, \ldots, h(x_n) = y_n\}| \leq \frac{|H|}{|B|^n}$$

where the set on the left side is also considered a multiset. The system of functions H *is called* strongly n-universal class *of hash functions.*

Every strongly n-universal class of hash functions is nearly strongly n-universal class with constant one. Most of the presented strongly universal classes not only satisfy the inequality from Definition 3.3 but the bound equals the number of the colliding functions.

Definition 3.4 (Strongly ω-universal class of hash functions)**.** *Family of functions* H *is* strongly ω-universal *if it is strongly* n*-universal for every* $n \in \mathbb{N}$.

The strongly ω-universal systems provide us with an estimate of the expected length of the longest chain. This bound can be proved directly from the above property without regarding any other special attributes of a system. As shown later the above property is difficult to be satisfied by a small class of functions. A straightforward example of a strongly ω-universal class is the set of all functions

from a domain U to a hash table B. It is obvious that the system is not very convenient because of its size. Simple construction of the system of all functions is briefly discussed later and shown in [34].

One may ask if there are classes that are strongly ω-universal other than the empty class or the class of all functions. However, there are classes that are not necessarily strongly ω-universal, but give the asymptotic bound on the expected length of the longest chain $O\left(\frac{\log n}{\log \log n}\right)$. The bound is the same for standard hashing and for strongly ω-universal systems as shown in Chapter 4. The families are based on the system of polynomials or linear systems and are quite large and not very efficient compared to the standard systems. "Distinct large families of functions were given by Siegel [11] and by Dietzfelbinger and Meyer auf der Heide [33]. Both provide families of size $|U|^{n^\epsilon}$. Their functions can be evaluated in a constant time on a RAM." The families are somewhat more complex to implement when compared to the class of linear functions as stated in [34]. However, these classes may be cosidered "nearly" strongly ω-universal, in fact they are strongly $\log n$-universal.

The above definitions of universal classes can be rewritten in terms of probability as well. The probability space corresponds to the uniform choice of a function from the system of functions. For example in Definition 3.1 of c-universal class H, probability of collision of two different elements x and y can be rewritten as

$$\mathbf{Pr}\left(h(x) = h(y)\right) = \frac{|\{h \in H \mid h(x) = h(y)\}|}{|H|} = \frac{c}{|B|}.$$

In the same manner we can reformulate and use definitions for the other universal systems.

In the remainder of this chapter we sum up some basic facts regarding universal systems, show remarks about their combinations and derive some theoretic bounds of strongly k-universal systems.

Theorem 3.5. *Every strongly* 2*-universal class of hash functions is also 1-universal.*

Proof. Let H be strongly 2-universal class containing functions from a universe U into a set B. We have to find the number of functions in H which make a given pair of different elements $x, y \in U$ collide.

First, define the set of functions that map both x and y on the image $t \in B$

$$H_t = \{h \in H \mid h(x) = h(y) = t\}.$$

These sets are disjoint and moreover $|H_t| \leq \frac{|H|}{|B|^2}$. By summing throughout all $t \in B$ we have

$$|\{h \in H \mid h(x) = h(y)\}| = \sum_{t \in B} |H_t| \leq |B| \frac{|H|}{|B|^2} = \frac{H}{|B|}.$$

The number of all colliding functions is then less or equal to $\frac{H}{|B|}$ and the system H is clearly 1-universal. □

Theorem 3.6. *Every strongly* n*-universal class of functions is strongly* k*-universal for every* $1 \leq k \leq n$*.*

Proof. Let H be a n-universal class consisting of functions from a universe U into a hash table B. Similarly to the previous proof there are k different elements $x_1, \ldots, x_k \in U$ that should be mapped to the prescribed buckets $y_1, \ldots, y_k \in B$. For every choice of their images $y_1, \ldots, y_k \in B$ we need to find the number of functions mapping the element x_i to its image y_i, $1 \leq i \leq k$.

The only estimate we can use comes from the strong n-universality of H and holds for n elements only. We extend the set of images by other $n-k$ elements $x_{k+1}, \ldots, x_n$. They are let to be mapped on arbitrary element of B. Such extension must be done carefully so that the final sequence consists of different elements $x_1, \ldots, x_n$, too. From now on fix one such extension, $x_{k+1}, \ldots, x_n$.

The set of functions mapping given k elements onto their prescribed images, $\bar{H}$, may be seen as

$$\begin{aligned}\bar{H} &= \{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, k\}\} \\ &= \bigcup_{y_{k+1}, \ldots, y_n \in B} \{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, n\}\}.\end{aligned}$$

Notice that sets of functions that map elements $x_{k+1}, \ldots, x_n$ onto different images $y_{k+1}, \ldots, y_n$ are disjoint. The sum of their sizes is then equal to $|\bar{H}|$. It follows that

$$\begin{aligned}|\bar{H}| &= \left| \bigcup_{y_{k+1}, \ldots, y_n \in B} \{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, n\}\} \right| \\ &= \sum_{y_{k+1}, \ldots, y_n \in B} |\{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, n\}\}| \\ &\leq \sum_{y_{k+1}, \ldots, y_n \in B} \frac{H}{|B|^n} = |B|^{n-k} \frac{H}{|B|^n} = \frac{H}{|B|^k}.\end{aligned}$$

Now we can see that the class of functions H is strongly k-universal, too. □

3.2 Examples of universal classes of functions

In this section we show some examples of common universal classes of hash functions. With each system we provide a simple proof of its universality or strong universality. Presented systems not only differ in contained functions but also in their domains and co-domains. However every system can be thought of as a mapping from a subset of natural numbers onto another subset of natural numbers.

Linear system. System of linear functions was among the first systems of universal hash functions that were discovered. They were introduced by Carter and Wegman in [6] where they showed basic properties of universal hashing. Above all they mentioned the expected constant time of the find operation of universal hashing. Nowadays various modifications of linear systems are known and are presented in the following text. The system is also mentioned in [23] and [25].

Definition 3.7 (Linear system)**.** *Let N be a prime, $m \in \mathbb{N}$ and let $U = \{0, \ldots, N-1\}$ and $B = \{0, \ldots, m-1\}$ be sets. Then the class of linear functions*

$$LS = \{h_{a,b}(x) : U \to B \mid a, b \in U \text{ where } h_{a,b}(x) = ((ax + b) \bmod N) \bmod m\}$$

is called linear system.

Remark 3.8. *Linear system is $\frac{\left\lceil \frac{N}{m} \right\rceil^2}{\left(\frac{N}{m}\right)^2}$-universal.*

Proof. Consider two different elements $x, y \in U$. We need to find the number of functions $h_{a,b} \in H$ such that

$$(ax + b \bmod N) \bmod m = (ay + b \bmod N) \bmod m.$$

This is equivalent to the existence of numbers r, s, t that satisfy the following constraints:

$$\begin{gathered} r \in \{0, \ldots, m-1\} \\ t, s \in \left\{0, \ldots, \left\lceil \frac{N}{m} \right\rceil - 1\right\} \\ (ax + b) \bmod N = sm + r \\ (ay + b) \bmod N = tm + r. \end{gathered}$$

Since N is a prime number and $x \neq y$ for every choice of parameters r, s, t there is an exactly unique solution; parameters a, b that satisfy the equalities.

Number of all possible choices of parameters r, s, t is $m \left\lceil \frac{N}{m} \right\rceil^2$. So there are exactly $m \left\lceil \frac{N}{m} \right\rceil^2$ functions $h_{a,b}$ that map the elements x, y to the same bucket.

We compute the system's c-universality constant:

$$m \left\lceil \frac{N}{m} \right\rceil^2 = m \left\lceil \frac{N}{m} \right\rceil^2 \frac{\frac{N^2}{m}}{\frac{N^2}{m}} = \frac{\left\lceil \frac{N}{m} \right\rceil^2}{\frac{N^2}{m^2}} \frac{N^2}{m} = \frac{\left\lceil \frac{N}{m} \right\rceil^2}{\frac{N^2}{m^2}} \frac{|LS|}{|B|}.$$

Hence the linear system is $\frac{\left\lceil \frac{N}{m} \right\rceil^2}{\left(\frac{N}{m}\right)^2}$-universal. □

For examining the properties of the original linear system simpler and smaller multiplicative system may be used. They have many common properties and share some same drawbacks. This multiplicative system may provide us with some insights for which choices of hash function and stored set they both fail.

Definition 3.9 (Multiplicative system)**.** *Let p be a prime and m, $m < p$ be the size of the hash table. Then the system*

$$\text{Mult}_{m,p} = \{h : \mathbb{Z}_p \to \mathbb{Z}_m \mid h(x) = (kx \bmod p) \bmod m \text{ for } k \in \mathbb{Z}_p - \{0\}\}$$

is called the multiplicative system.

An important application of the multiplicative system is the construction of a perfect hash function shown in [16] by Fredman, Komlós and Szemeredi. The system and its other properties are also mentioned in [25].

Remark 3.10. *For a prime p and $m < p$ the multiplicative system is 2-universal.*

Proof. Let x and y be different elements of the universe. We have to find the number of functions in the multiplicative system that make their images the same. For a collision function we have that

$$\begin{aligned} kx \bmod p \bmod m &= ky \bmod p \bmod m \\ kx - ky \bmod p \bmod m &= 0. \end{aligned}$$

This may be rewritten as:

$$k(x-y) \bmod p = lm \qquad \text{for } l \in \left\{ -\left\lfloor \frac{p}{m} \right\rfloor, \ldots, \left\lfloor \frac{p}{m} \right\rfloor \right\} - \{0\}.$$

Notice that the parameter $l \neq 0$ since $k \neq 0$. Thus for every choice of x and y there are at most $2\left\lfloor \frac{p}{m} \right\rfloor$ functions. For every value of l there is exactly one k satisfying the equality because parameter p is a prime but k can be a solution for distinct l. For the number of colliding functions we have

$$|\{h \in \mathrm{Mult}_{m,p} \mid h(x) = h(y)\}| \leq \frac{2p}{m}$$

and the system is then 2-universal. □

Previous systems contain only simple linear functions from natural numbers to natural numbers. An analogous class can be constructed by using linear transformations between vector spaces. The systems have various similar properties but the latter one gives us a suitable bound on the size of the largest bucket.

System of linear transformations. Systems of linear transformations are studied later in Chapter 5. We use them in Chapter 6 to create a new model of universal hashing. The system is discussed in a very detail way in [3].

Definition 3.11 (The set of all linear transformations, the set of all surjective linear transformations)**.** *Let U and B be two vector spaces. Denote* the set of all linear transformations *as*

$$LT(U,B) = \{T : U \to B \mid T \textit{ is a linear transformation}\}$$

and let

$$LTS(U,B) = \{T : U \to B \mid T \textit{ is a surjective linear transformation}\}$$

denote the set of all surjective linear transformations *between vector spaces U and B.*

Definition 3.12 (System of linear transformations)**.** *Let U and B be two vector spaces over the field $\mathbb{Z}_2$. The set $LT(U, B)$ is called the* system of linear transformations *between vector spaces U and B.*

Remark 3.13. *System of linear transformations between vector spaces U and B is* 1*-universal.*

Proof. Let m and n denote the dimensions of vector spaces U and B respectively. Then every linear transformation $L : U \to B$ is associated with a unique $n \times m$ binary matrix T.

Let $\vec{x}, \vec{y} \in U$ be two distinct vectors mapped on a same image. We need to find the number of linear mappings confirming this collision. Let k be a position such that $x_k \neq y_k$. Such k exists since $\vec{x} \neq \vec{y}$. We show the following statement. If the matrix T is fixed except the k^{th} column, then the k^{th} column is determined uniquely. We just lost n degrees of freedom when creating matrix T causing the collision of $\vec{x}$ and $\vec{y}$. This lost implies 1-universality of the system, this is proved later, too.

In order to create a collision of $\vec{x}$ and $\vec{y}$ their images $T\vec{x}$, $T\vec{y}$ must be the same. The system of equalities must then be true for every i, $1 \leq i \leq n$:

$$\sum_{j=1}^{m} t_{i,j} x_j = \sum_{j=1}^{m} t_{i,j} y_j.$$

For $t_{i,k}$ we have:

$$t_{i,k} = (x_k - y_k)^{-1} \sum_{j=1, j\neq k}^{m} t_{i,j}(y_j - x_j).$$

The elements in the k-th column are uniquely determined by the previous equality.

The number of all linear functions, equivalently the number of all binary matrices, is 2^{mn}. Notice that the size of the target space B is 2^n. The number of all linear functions mapping $\vec{x}$ and $\vec{y}$ to a same value is 2^{mn-n} since we are allowed to arbitrarily choose every element of the matrix T except the n ones in the k-th column.

System of linear transformations is 1-universal because

$$|\{L \in LT(U, B) \mid L(x) = L(y)\}| = 2^{mn-n} = \frac{2^{mn}}{2^n} = \frac{|LT|}{|B|}.$$

□

We are not limited only to vector spaces over field $\mathbb{Z}_2$. Systems of all linear transformations between vector spaces over any finite field $\mathbb{Z}_p$ are 1-universal, too. However the most interesting results are achieved for the choice of $\mathbb{Z}_2$.

Bit sum of a string over the alphabet {0, 1}. Presented family is another linear system that is 1-universal, too. It is clear that every natural number can be written as a string over alphabet consisting from two characters only, 0 and 1, it is the number's binary form. Weighted digit sum is considered the number's hash value. This reminds us of the system of linear transformations.

If we are hashing strings that are $k+1$-bit long we also need $k+1$ coefficients. Also assume hashing into a table of size $p \in \mathbb{N}$ where p is a prime. Coefficients may be chosen arbitrarily but are not greater than a parameter l, $0 < l \leq p$. To transform a digit sum into an address of the hash table simple modulo p operation is used. Parameter l may seem quite artificial. However it sets the range for coefficients and therefore determines the size of the whole system.

Definition 3.14 (Bit string system)**.** *Let p be a prime, $k \in \mathbb{N}$, $B = \{0, \ldots, p-1\}$ and $l \in \mathbb{N}$, $l \leq p$. System of functions*

$$BSS_{p,l} = \left\{ h_{\vec{c}} : \{0,1\}^{k+1} \to B \mid h(x) = \left(\sum_{i=0}^{k} c_i x_i \right) \bmod p, \vec{c} = (c_0, \ldots, c_k) \in \mathbb{Z}_l^{k+1} \right\}$$

is called bit string system *with parameters p and l.*

Remark 3.15. *Bit string system for binary numbers of length $k+1$ modulo prime p and constant $l \in \mathbb{N}$, $l \leq p$ is $\frac{p}{l}$-universal.*

Proof. Let x and y be two different bit strings $x, y \in \{0,1\}^{k+1}$. We must estimate the number of all sequences of coefficients $0 \leq c_0, \ldots, c_k < l$ that make the two elements collide. Every collision sequence must satisfy:

$$\left(\sum_{i=0}^{k} c_i x_i \right) \bmod p = \left(\sum_{i=0}^{k} c_i y_i \right) \bmod p.$$

Since $x \neq y$ there is an index j, $0 \leq j \leq k$ such that $x_j - y_j \neq 0$. This allows us to exactly determine the j^{th} coefficient c_j from the others:

$$\begin{aligned}
\left(\sum_{i=0}^{k-1} c_i x_i \right) \bmod p &= \left(\sum_{i=0}^{k-1} c_i y_i \right) \bmod p \\
c_j (x_j - y_j) \bmod p &= \left(\sum_{i=0, i \neq j}^{k-1} c_i (x_i - y_i) \right) \bmod p \\
c_j &= (x_j - y_j)^{-1} \left(\sum_{i=0, i \neq j}^{k-1} c_i (x_i - y_i) \right) \bmod p.
\end{aligned}$$

Last equality holds since p is a prime number and the number $x_j - y_j$ is invertible in the field $\mathbb{Z}_p$. The computed c_j may be from the set $\{0, \ldots, p-1\}$ but we need it less then l. For such choice of the other coefficients c_i, $i \neq j$ there is no function

causing a collision. However we still have a valid upper bound on the number of colliding functions. There is one degree of freedom lost when choosing a function that makes x and y collide. The number of all functions is l^{k+1} and the size of the table is p. The number of colliding functions can be computed as:

$$|\{h \in BSS_{p,l} | h(x) = h(y)\}| \leq l^k = \frac{p}{l}\frac{l^{k+1}}{p} = \frac{p}{l}\frac{|BSS_{p,l}|}{|B|}.$$

The bit string system is thus $\frac{p}{l}$-universal. □

Polynomials over finite fields. The following system is a generalisation of so far discussed linear functions and linear transformations to polynomials. The advantage delivered by the system of polynomials is its strong universality. Unfortunately it is traded for the bigger size of the system. The class is briefly mentioned in [34].

Definition 3.16 (System of polynomial functions)**.** *Let N be a prime number and $n \in \mathbb{N}$. Define $U = \{0, \ldots, N-1\}$ and $B = \{0, \ldots, m-1\}$ The system of functions*

$$P_n = \{h_{c_0,\ldots,c_n}(x) : U \to B \mid c_i \in U, 0 \leq i \leq n\}$$

where

$$h_{c_0,\ldots,c_n}(x) = \left(\left(\sum_{i=0}^{n} c_i x^i\right) \bmod N\right) \bmod m$$

is called the system of polynomial functions *of the n-th degree.*

Remark 3.17. *Let N be a prime number and $n \in \mathbb{N}$. If $B = U$ then the system of polynomial functions of the n-th degree is strongly $n+1$-universal. When $B \neq U$ the system is nearly strongly $n+1$-universal.*

Proof. Let $x_1, x_2, \ldots, x_{n+1}$ be different elements of U and $y_1, y_2, \ldots, y_{n+1}$ are their prescribed images. We can write down a system of linear equations that can be used to find the coefficients $c_0, c_1, \ldots, c_n$:

$$h(x_i) = y_i, \quad 0 \leq i \leq n.$$

If $U = B$ the function $h(x)$ is reduced to the form:

$$h(x) = \left(\sum_{i=0}^{n} c_i x^i\right) \bmod N.$$

Since N is a prime number and the elements x_i are different the corresponding Vandermond matrix is regular. There is exactly one solution of the above system, let $c_0, \ldots, c_n$ denote the solution. Size of the system is $|U|^{n+1}$ and $1 = \frac{|U|^{n+1}}{|B|^{n+1}} = \frac{|U|^{n+1}}{|U|^{n+1}}$. Now it is clear that the system of polynomial functions is strongly $n+1$-universal.

Let $U \neq B$. We use the idea from the proof of c-universality of linear system. First we write down the equations $h(x_i) = y_i$ in the field $\mathbb{Z}_N$. Instead using mod$N$ we introduce new variables r_i as in the proof for linear system such that

$$\left(\sum_{j=0}^{n} c_j x_i^j\right) \bmod N = y_i + r_i m \qquad r_i \in \left\{0, \ldots, \left\lceil \frac{N}{m} \right\rceil - 1\right\}.$$

For every choice of all r_i, $0 \leq i \leq n$, we obtain an unique solution of the above system of equations since we are in the field $\mathbb{Z}_N$. The number of all choices of r_i is $\left\lceil \frac{N}{m} \right\rceil^{n+1}$. From the number of functions which map x_i to y_i for $1 \leq i \leq n$ it follows that

$$|\{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, n+1\}\}| \leq \left\lceil \frac{N}{m} \right\rceil^{n+1} = \frac{\left\lceil \frac{N}{m} \right\rceil^{n+1}}{\left(\frac{N}{m}\right)^{n+1}} \left(\frac{N}{m}\right)^{n+1}.$$

Hence this system is nearly strongly $n+1$-universal with the constant $\frac{\left\lceil \frac{N}{m} \right\rceil^{n+1}}{\left(\frac{N}{m}\right)^{n+1}}$. □

Although the choice of $U = B$ is not very practical one, however it gives us a clue that the system might be nearly strongly universal.

System of all functions. The strongly ω-universality is a powerful property. The question is whether a system that satisfies the property exists. One example, although not very useful, can be immediately constructed – it is a set of all functions between two sets. The applicable construction of the system is taken from [34].

Definition 3.18 (System of all functions)**.** *Let U and B be sets such that $|B| < |U|$. The family of functions*

$$H = \{h : U \to B\}$$

is called the system of all functions *between U and B.*

Remark 3.19. *System of all functions between sets U and B is strongly ω-universal.*

Proof. We are given n different elements $x_1, \ldots, x_n$ and their images $y_1, \ldots, y_n$. We must compute size of the set $H_n = \{h \in H \mid h(x_i) = y_i, i \in \{1, \ldots, n\}\}$. Now note that the system's size is $|H| = |B|^{|U|}$. Since the values for elements $x_1, \ldots, x_n$ are fixed we have

$$|H_n| = |B|^{|U|-n} = \frac{|B|^{|U|}}{|B|^n} = \frac{|H|}{|B|^n}$$

and thus the system is strongly ω-universal. □

The main problem when using this system is that fact that to encode a function $h \in H$ is highly inconvenient. To store one we need $|U| \log |B|$ bits. When using this simple approach we need to encode the values of elements that with great probability will never be stored in the hash table. An alternative is to construct random partial function from the set of all functions from B to U as shown in [34].

These ideas come from the existence of fast associative memory having expected time of find and insert operations $O(1)$ and existence of a random number generator. Whenever we have to store an element we must determine its hash value. First, by

using the associative memory we find out if the stored element already has a hash value associated. If not we use the random number generator to obtain an element from B and remember this connection, element – its hash value, to the associative memory. This sequentially constructs a random mapping from the system of all functions. Because the random number generator used chooses the elements of B uniformly created system is certainly strongly ω-universal. In addition we do not store the hash values for irrelevant elements.

By using this idea to construct a fast hash table we are trapped in a cycle since we already supposed existence of a fast associative memory. But ω-universal class can not only be used to construct a fast solution to the dictionary problem. It can solve other problems like the set equality as shown in [34].

3.3 Properties of systems of universal functions

In this section we sum up some properties of strongly k-universal classes of functions. We concentrate on the generalisation of the known properties of the c-universal classes to the strongly universal classes. These results may inspire us to create more powerful or better behaving systems without losing strong universality.

The first theorems show the results of combinations of two strongly universal systems.

Theorem 3.20. *Let H and I be strongly k-universal systems both from a universe U into a hash table B with $m = |B|$. For $h \in H$ and $i \in I$ let us define*

$$f_{(h,i)}(x) = (h(x), i(x))$$

for all $x \in U$. Then the system

$$J = \{f_{(h,i)} : U \to B \times B \mid h \in H, i \in I\}$$

is strongly k-universal.

Proof. Let $x_1, \ldots, x_k$ be distinct elements of U and $y_1, \ldots, y_k, z_1, \ldots, z_k \in B$. We find the number of pair functions $f_{(h,i)}$ such that for both functions h and i we have $h(x_1) = y_1, \ldots, h(x_k) = y_k$, $i(x_1) = z_1, \ldots, i(x_k) = z_k$. From strong k-universality of both systems we know that there are at most $\frac{|H|}{m^k}$ functions h and $\frac{|I|}{m^k}$ functions i satisfying mentioned criteria. So the number of all functions $f_{(h,i)}$ such that $f_{(h,i)}(x_1) = (y_1, z_1), \ldots, f_{(h,i)}(x_k) = (y_k, z_k)$ is at most $\frac{|H||I|}{m^{2k}}$. The size of the table is m^2 and thus the constructed system is strongly k-universal. □

Combination of two universal systems does not have to increase the size of the table. From the previous result we obtain a smaller probability of collision by paying the expensive price of the table expansion. When using single class of hash functions the same effect on probability decrease can be achieved by simple squaring the table's size. We could also combine two strongly universal systems of functions and use a suitable operation to merge the given results into a small table. The created system remains strongly universal as it is stated in the next theorem and the combination does not bring us any obvious advantage.

Theorem 3.21. *Let B denote the set of buckets of a hash table of size m, $|B| = m$. Let both H and I be strongly k-universal classes of functions over the table B. Let o be a binary left cancellative operation on B, ie. for every $a, c \in B$ there exists exactly one $b \in B$ with $o(a, b) = c$. Then the system of functions*

$$F = \{f_{(h,i)} | h \in H, i \in I\}$$

where

$$f_{(h,i)}(x) = o(h(x), i(x))$$

is strongly k-universal.

Proof. To prove the statement assume that $x_1, \ldots, x_k$ are pairwise distinct elements of U and $y_1, \ldots, y_k$ are elements of B. For every y_i, $1 \leq i \leq k$ choose a pair (a_i, b_i) with $o(a_i, b_i) = y_i$. The number of such choices is m^k. From Theorem 3.20 for a fixed choice of (a_i, b_i), $1 \leq i \leq k$, it follows that the probability of the event

$$h(x_1) = a_1, \ldots, h(x_k) = a_k \text{ and } i(x_1) = b_1, \ldots, i(x_k) = b_k$$

is less than $\frac{1}{m^{2k}}$. Hence $\mathbf{Pr}\left(f(x_i) = y_i, 1 \leq i \leq k\right) = \frac{m^k}{m^{2k}} = \frac{1}{m^k}$. Now we see that the system F is strongly k-universal. □

Corollary 3.22. *Combining the results of pair functions created from strongly k-universal classes by operations*

- *xor where the table size is a power of 2*
- *$+, -$ over an additive group Z_l where l is the table size*

gives strongly k-universal classes of functions.

Proof. Both of the operations are left cancellative because they are group operations. □

Next step is to discover the dependence of the parameter k of strongly k-universal classes on their size. Theorem 3.23 shows us the best possible probability bound for the collision of k different elements depending on the size of the system when considering the strongly k-universality of a system. Original proof for c-universal classes may be found in [24] and [23]. We slightly modify it to obtain a lower bound on the size of a strongly k-universal class.

Theorem 3.23. *Let H be a strongly k-universal system of functions from a universe U of size N to a table B of size m. Size of the system H is then bounded by:*

$$|H| > m^{k-1} \log_m \left(\frac{N}{km}\right).$$

Proof. Let $H = \{h_1, \ldots, h_{|H|}\}$. By induction we create sequence $U_0, U_1, \ldots$ of sets such that $U_0 = U$ and U_i is a greatest subset of U_{i-1} such that $h_i(U_i)$ is a singleton.

From the Dirichlet's principle size of every set U_i satisfies $|U_i| \geq \frac{|U_{i-1}|}{m}$. Hence we obtain $|U_i| \geq \frac{|U|}{m^i}$.

Let i be the greatest index with $|U_i| \geq k$. Functions $h_1, h_2, \ldots, h_i$ map at least k elements of set U_i to a singleton. By Remark 3.6, the probability of this event is less or equal to $\frac{1}{m^{k-1}}$. It follows that $i \leq \frac{|H|}{m^{k-1}}$. Since i is the greatest index with the property $|U_i| \geq k$ we also have that $\frac{|U|}{m^{i+1}} \leq |U_{i+1}| < k$.

From $\frac{N}{m^{i+1}} < k$ it follows $\frac{N}{km} < m^i$ and thus $\log_m\left(\frac{N}{km}\right) < i$.

Obtaining the bound on the size of system of functions:

$$\frac{|H|}{m^{k-1}} \geq i > \log_m\left(\frac{N}{km}\right)$$
$$|H| > m^{k-1}\log_m\left(\frac{N}{km}\right).$$

□

Corollary 3.24. *If H is a strongly k-universal system of functions from a universe U of size N to a table B of size m, then*

$$k < 1 + \frac{\log|H| - \log\log_m\left(\frac{N}{km}\right)}{\log m} < 1 + \frac{\log|H|}{\log m}.$$

Proof. Theorem 3.23 can be reformulated to get a bound on k:

$$m^{k-1} < \frac{|H|}{\log_m\left(\frac{N}{km}\right)}$$
$$k < 1 + \frac{\log|H| - \log\log_m\left(\frac{N}{km}\right)}{\log m}$$
$$k < 1 + \frac{\log|H|}{\log m}.$$

□

Chapter 4

Expected Length of the Longest Chain

4.1 Length of a Chain

In Lemma 2.1 we explain why the length of a chain is so important for hashing. Consider a model of hashing which assumes that chains, representing the elements of a single slot, are stored by linked lists. To find an element in a bucket (or chain) we need to iterate it. Thus the time of the find operation is linear with respect to the number of elements of a chain. An approach, how to estimate the expected length of the longest chain, for both classic and universal hashing is shown.

Definition 4.1 (Equality indicator)**.** *Let x and y be two variables from the same domain B. Then the function $\mathbf{I} : B \times B \to \{0, 1\}$ such that*

$$\mathbf{I}(y = x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

is called equality indicator*.*

Definition 4.2 (Length of a chain, length of the longest chain)**.** *Let U be a universe, B be a hash table and $h : U \to B$ be a hash function. Let $b \in B$ be a bucket and $S \subset U$ be a stored set. Then the* length of the chain *representing the bucket b when hashing with the function h is denoted by $\mathbf{psl}(b, h) = \sum_{x \in S} \mathbf{I}(h(x) = b)$. The value $\mathbf{lpsl}(h) = \max_{b \in B}(\mathbf{psl}(b, h))$ is called the* length of the longest chain *when hashing with the function h.*

Notation *probe sequence* comes from the open addressing. Open addressing assumes that chains are represented directly in the hash table. In addition, the chains from various buckets may be merged together. Thus they may contain elements with different hash values. Many authors refer to the process of finding an element in a chain as to the probing a sequence. Authors denote corresponding random variables by names **psl** and **lpsl** as well, for example in [7].

When taking the probability into an account we define random variables $\mathbf{psl}(b)$ and **lpsl** over the probability space determined by a choice of a hash from a universal class.

Definition 4.3 (Randomised variables **psl** and **lpsl**)**.** *Random variable* $\mathbf{psl}(b)$ *denotes the* length of the chain *representing a bucket* $b \in B$*. Its values are natural numbers and probability density function is defined as:*

$$\mathbf{Pr}\left(\mathbf{psl}(b) = k\right) = \frac{\sum_{h \in H} \mathbf{I}(\mathbf{psl}(b, h) = k)}{|H|} \text{ for } k \in \mathbb{N}.$$

Random variable **lpsl** *denoting the* length of the longest chain *is defined as*

$$\mathbf{lpsl} = \max_{b \in B}(\mathbf{psl}(b)).$$

In universal hashing the expected value of both random variables **psl** and **lpsl** is taken over the choice of a function $h \in H$. Hence the bounds on the variable **lpsl** are valid for any stored set $S \subset U$. They frequently depend on the size of the stored set and on the size of the hash table but are independent of the stored elements. The value $\mathbf{E}\left(\mathbf{lpsl}\right)$ is not only important characteristic of the expected worst case. As shown later in Chapter 6, if we are able to find a tight upper bound on $\mathbf{E}\left(\mathbf{lpsl}\right)$, then we are also able to guarantee the worst case behaviour of the model.

4.2 Estimate for Separate Chaining

The standard result of classic hashing is the bound $O\left(\frac{\log n}{\log \log n}\right)$ on the length of the longest chain. We can reuse calculations in its proof found in [24] to show the same bound for universal hashing with a strongly ω-universal class. In either case when proving this result, we need to estimate the probability of collision of k elements. In the classic model of hashing the estimate follows from the assumptions given in Section 2.2. The assumptions are strong enough to solve the problem in the area of classic hashing. However, they are not necessarily satisfied for many real world situations.

Definition 4.3 of the random variables **lpsl** and **psl** are made for models of universal hashing. Yet, they are easy to understand with the standard hashing, too.

Remark, when using the standard model, we do not have to mention the additional parameter h of the variables **psl** and **lpsl** because the standard model assumes a single hash function only. Recall that the probability space of this model corresponds to the choice of a stored set.

Additional restriction placed on the hash function requires that it distributes the elements of the universe uniformly across the hash table. Hence the other parameter, bucket $b \in B$, of the variable **psl** is no longer necessary, too. From this fact it follows that for two arbitrary buckets $a, b \in B$ the random variables $\mathbf{psl}(a)$ and $\mathbf{psl}(b)$ are equal:

$$\mathbf{Pr}\left(\mathbf{psl}(a) = k\right) = \mathbf{Pr}\left(\mathbf{psl}(b) = k\right) \text{ for every } k \in \mathbb{N}.$$

In this chapter we use the notation from Section 2.1. So U is a universe, m denotes the size of the hash table B and n is the size of the stored set $S \subset U$.

Theorem 4.4. *Assume the model of separate chaining with $\alpha < 1$. Then*

$$\mathbf{E}(\mathbf{lpsl}) \in O\left(\frac{\log n}{\log \log n}\right).$$

Proof. The probability estimate for the variable **lpsl** is simply obtained from the probability estimate of **psl** as:

$$\mathbf{Pr}(\mathbf{lpsl} \geq i) \leq m\mathbf{Pr}(\mathbf{psl}(b) \geq i \text{ for any } b \in B).$$

We use the definition of the expected value to find $\mathbf{E}(\mathbf{lpsl})$:

$$\begin{aligned}\mathbf{E}(\mathbf{lpsl}) &= \sum_{i=0}^{\infty} i\mathbf{Pr}(\mathbf{lpsl} = i) \\ &= \sum_{i=0}^{\infty} i(\mathbf{Pr}(\mathbf{lpsl} \geq i) - \mathbf{Pr}(\mathbf{lpsl} \geq i+1)) \\ &= \sum_{i=0}^{\infty} \mathbf{Pr}(\mathbf{lpsl} \geq i).\end{aligned}$$

We obtained:

$$\mathbf{E}(\mathbf{lpsl}) \leq \sum_{i=0}^{\infty} m\mathbf{Pr}(\mathbf{psl} \geq i).$$

Since the hash function divides the universe U uniformly across the table, for every $x \in U$, $b \in B$ we have that $\mathbf{Pr}(h(x) = b) = \frac{1}{m}$. Now we use the assumption of the independent and uniform choice of the hashed element. Provided the assumption, the probability of collision of $i \in \mathbb{N}$ elements may be estimated by a binomial random variable as. Hence for the collision of at least i elements it follows

$$\mathbf{Pr}(\mathbf{psl} \geq i) \leq \binom{n}{i}\left(\frac{1}{m}\right)^i.$$

Remark that this estimate holds only when the universe is substantially larger than the stored set. Selection of the first element slightly changes the probability of the choice of the second one and so on. This fact is neglected for large or infinite universes.

We can finish the estimate of the variable **lpsl** by computing:

$$\begin{aligned}\mathbf{E}(\mathbf{lpsl}) &\leq \sum_{i=0}^{\infty} m\mathbf{Pr}(\mathbf{psl} \geq i) \\ &\leq \sum_{i=0}^{\infty} m \min\left(1, \binom{n}{i}\left(\frac{1}{m}\right)^i\right) \\ &= O\left(\frac{\log n}{\log \log n}\right).\end{aligned}$$

If we assume that the table's load factor $\alpha < 1$, we can complete the proof. The details of the estimate can be found in [24] and in [23]. □

4.3 Estimate for Universal Hashing

In universal hashing we move to a different probability estimate of collision of k elements. The probability bound follows from strongly ω-universality of the class H, we use. We neither assume nor exploit specific properties other than strongly ω-universality.

Definition 4.5 (Set indicator)**.** *Let* $\mathbf{I}$ *be a function such that*

$$\mathbf{I} : 2^U \to \{0, 1\}$$

$$\mathbf{I}(M) = \begin{cases} 0 & M = \emptyset \\ 1 & M \neq \emptyset. \end{cases}$$

Then I *is called a* set indicator*.*

The relation between the size of a set M and its indicator is described by the next lemma.

Lemma 4.6.

$$\mathbf{I}(M) \leq |M|.$$

Proof. We distinguish between the cases $M = \emptyset$ and $M \neq \emptyset$.

- If $M = \emptyset$ then $I(M) = 0 = |M|$ and the inequality holds.
- If $M \neq \emptyset$ then $I(M) = 1 \leq |M|$ so the statement is true.

□

Now we present a method how to estimate the probability of collision of k elements.

Definition 4.7 (Collision sets of k elements)**.** *Let* $h \in H$ *be a function,* $S \subset U$ *be a set and* $b \in B$*. For a natural number* k *define*

$$M_{\geq k}(h, b) = \{Y \subseteq S \mid |Y| \geq k, h(Y) = \{b\}\},$$
$$M_{=k}(h, b) = \{Y \subseteq S \mid |Y| = k, h(Y) = \{b\}\}.$$

When it is clear, we can omit parametrisation of the sets and use the notation $M_{\geq k}$ or $M_{=k}$. Simply said, sets $M_{\geq k}$ and $M_{=k}$ denote the chains of length at least k and equal to k, respectively. Formally, chain of length k is a subset of the stored set S consisting of k elements which are hashed to a singleton.

Lemma 4.8. *Let* U *be a universe,* B *represent a hash table,* $b \in B$ *a bucket,* $S \subset U$ *be a stored set and* $h : U \to B$*. Then the non-emptiness of the set* $M_{\geq k}(h, b)$ *is equivalent to the non-emptiness of the set* $M_{=k}(h, b)$*. Equivalently*

$$\mathbf{I}(M_{\geq k}(h, b)) = \mathbf{I}(M_{=k}(h, b)).$$

Proof. Let $M_{\geq k}$ be a non empty set, then:

$$\begin{aligned} Y \in M_{\geq k} &\Rightarrow \forall Y' \subseteq Y, |Y'| = k : h(Y') = \{b\} \\ &\Leftrightarrow \forall Y' \subseteq Y, |Y'| = k : Y' \in M_{=k} \\ &\Rightarrow \exists Y' \subseteq Y, |Y'| = k : Y' \in M_{=k}. \end{aligned}$$

Let $M_{=k}$ be a non empty set:

$$Y' \in M_{=k} \Rightarrow Y' \in M_{\geq k}.$$

Now we have:

$$\begin{gathered} M_{\geq k} \neq \emptyset \Leftrightarrow M_{=k} \neq \emptyset \\ \mathbf{I}(M_{\geq k}) = \mathbf{I}(M_{=k}). \end{gathered}$$

□

Now fix any arbitrary bucket $b \in B$. If the chain in the bucket b has $\mathbf{psl}(b, h) \geq k$, then there is a subset $Y \subseteq S$, $|Y| \geq k$, such that $h(Y) = \{b\}$. If $\mathbf{psl}(b, h) = k$, then set Y, $|Y| = k$, is maximal considering its size. It means that the set Y may not be extended to a larger one hashed just onto b. We can write down the probability density functions of $\mathbf{psl}(b)$ as:

$$\begin{aligned} \mathbf{Pr}\left(\mathbf{psl}(b) \geq k\right) &= \frac{\sum_{h \in H} \mathbf{I}(\{Y \subseteq S \mid |Y| \geq k, h(Y) = \{b\}\})}{|H|}, \\ \mathbf{Pr}\left(\mathbf{psl}(b) = k\right) &= \frac{\sum_{h \in H} \mathbf{I}(\{Y \subseteq S \mid |Y| = k, h(Y) = \{b\}, Y \text{ is maximal}\})}{|H|}. \end{aligned}$$

We add some remarks clarifying the following calculations:

- Every chain is determined by a bucket $b \in B$.
- Probability estimate is always computed for a fixed set S.
- We have to find an estimate

  $$\mathbf{Pr}\left(\mathbf{psl}(b) \geq k\right) \leq p(B, U, H, |S|)$$

  for some $p(B, U, H, |S|)$. This estimate is then used to estimate the distribution function of **lpsl**:

  $$\mathbf{Pr}\left(\mathbf{lpsl} \geq k\right) \leq |B| p(B, U, H, |S|).$$

- The obtained estimate does not depend on the elements of the stored set S. However, it depends on its size $n = |S|$.

From Lemma 4.8 it follows that the probability of collision of more than k elements can be bound as

$$\begin{aligned}\mathbf{Pr}\left(\mathbf{psl}(b) \geq k\right) &= \frac{\sum_{h\in H} \mathbf{I}(\{Y \subseteq S \mid |Y| \geq k, h(Y) = \{b\}\})}{|H|} \\ &= \frac{\sum_{h\in H} \mathbf{I}(\{Y \subseteq S \mid |Y| = k, h(Y) = \{b\}\})}{|H|} \\ &\leq \frac{\sum_{h\in H} |\{Y \subseteq S \mid |Y| = k, h(Y) = \{b\}\}|}{|H|} \\ &= \frac{|\{(h, Y) \mid |Y| = k, h(Y) = \{b\}\}|}{|H|} \\ &= \frac{\sum_{Y\subseteq S, |Y|=k} |\{h \in H \mid h(Y) = \{b\}\}|}{|H|}.\end{aligned}$$

Claim 4.9. *Let $k \in \mathbb{N}$, $b \in B$ be a bucket and H be a universal class. If we assume universal hashing with the class H, then*

$$\mathbf{Pr}\left(\mathbf{psl}(b) \geq k\right) \leq \frac{\sum_{Y\subseteq S, |Y|=k} |\{h \in H \mid h(Y) = \{b\}\}|}{|H|}.$$

Proof. Follows from the previous calculation. □

We further estimate the above fraction from the properties of the class H. In this case we use strong ω-universality as stated in the next theorem.

Theorem 4.10. *If H is a strongly ω-universal system, then $\mathbf{E}\left(\mathbf{lpsl}\right) \in O\left(\frac{\log n}{\log\log n}\right)$.*

Proof. Estimate the probability of the existence of a chain consisting of at least $k \in \mathbb{N}$ elements:

$$\begin{aligned}\mathbf{Pr}\left(\mathbf{psl}(b) \geq k\right) &\leq \frac{\sum_{Y\subseteq S, |Y|=k} |\{h \in H | h(Y) = \{b\}\}|}{|H|} \\ &\leq \frac{\sum_{Y\subseteq S, |Y|=k} \frac{|H|}{|B|^k}}{|H|} \\ &= \binom{|S|}{k} \frac{1}{|B|^k}.\end{aligned}$$

Now we carry on as in the case of standard hashing. The probability estimate of collision of k elements is exactly the same. Thus we can obtain the same result $\mathbf{E}\left(\mathbf{lpsl}\right) = O\left(\frac{\log n}{\log\log n}\right)$. □

A construction of a small strongly ω-universal system is difficult. Later in this work we investigate the system of linear transformations. Although the system is not strongly ω-universal, its specific properties, independent of the strong ω-universality, give an interesting upper bound on the distribution function of the variable **lpsl**.

Chapter 5

The System of Linear Transformations

From now on, we concentrate on the systems of linear transformations between vector spaces over the field $\mathbb{Z}_2$. The reason is simple, they allow a construction of a remarkable model of universal hashing. Like for the other universal models, the expected length of a chain is constant and so is the expected running time of the find operation. In addition, the expected length of the longest chain is bounded by a sub-linear function of a single variable – the size of the hash table. In our model, we have also managed to prove that the expected amortised time of every operation, find, insert and delete, is constant.

Once again, the main result is the upper bound on the expected length of the longest chain when a system of linear maps is used as a universal class. As already mentioned, probability of collision of k-elements is not estimated directly from the properties common to all universal systems. Hence the proof of the bound can not be based on the idea shown in the previous chapter.

First, we show a few models of the uniform random choice of a linear function. They describe other ways of the uniform random choice of a linear function from the universal class – choice of the universal hash function. Later, we prove some technical lemmas regarding the vector spaces. Finally, we state the required facts in terms of system of linear transformations. Then we interpret the facts in terms of hashing and propose a new model. Many theorems come from [3]. Our work lies in their correction, improvement and modification leading to a better and practical result.

In the following sections we restrict ourselves only to the vector spaces over the field $\mathbb{Z}_2$. If vector spaces are taken over other finite field than the field $\mathbb{Z}_2$, then we can not expect such good results as shown in [3]. The original result proposes hashing of $m \log m$ elements into a hash table of size of m slots and leads to the bound $O(\log m \log \log m)$. Although the result may already be used for hashing m elements, too, it is later improved for $\alpha \leq 1$.

When unsure about any exact definition or fact from linear algebra, see Appendix A where we exactly summarise and accurately state them.

5.1 Models of the Random Uniform Choice

Our technical preparations for key statements include models of the random uniform choice of a linear map. These models are used later to simplify proofs assuming the random uniform choice of a linear transformation – choice of a hash function. The simple random choice is transformed to the random uniform selection of two or more objects that precisely correspond to the linear function.

We recall Definition 3.11 saying that for two vector spaces A and B, the sets

$$LT(A,B) = \{T : A \to B \mid T \text{ is a linear transformation}\},$$
$$LTS(A,B) = \{T : A \to B \mid T \text{ is a surjective linear transformation}\}$$

and Definition A.8 stating that for two affine vector spaces A_A and B_A, the set

$$LT_A(A_A,B_A) = \{T_A : A_A \to B_A \mid T_A \text{ is an affine linear transformation}\}.$$

Definition 5.1 (Uniform selection)**.** *Let $S = \{s_1,\ldots,s_n\}$. By the random* uniform selection *of an element $s \in S$ we understand the model of selection with*

$$\mathbf{Pr}\,(s \in S \textit{ is selected}) = \frac{1}{n}.$$

Definition 5.2 (Set of all permutations)**.** *Let b be a natural number. Then Π_b denotes the* set of all permutations *of the set $\{1,\ldots,b\}$.*

The first model shows the correspondence between the random uniform choice of a basis of the source space that is mapped onto a fixed basis of the smaller target space and the random uniform choice of a surjective linear transformation.

Remark 5.3 (Surjective linear map selection)**.** *Let $\mathcal{B}$ be the set of all bases of the vector space $\mathbb{Z}_2^f$ and the set $\{\vec{y_1},\ldots,\vec{y_b}\}$ be a basis of the vector space $\mathbb{Z}_2^b$ where $b,f \in \mathbb{N}$ and $b \leq f$. Let us denote S the set of all subsets of $\{1,2,\ldots,f\}$ of size $f-b$. For the random uniform choice of a basis $\beta = \{\vec{b_1},\ldots,\vec{b_f}\} \in \mathcal{B}$, a set $s \in \mathcal{S}$ and a permutation $\pi \in \Pi_b$ we define the linear map $T_{\beta,s,\pi}$ as*

$$T_{\beta,s,\pi}(b_i) = \begin{cases} y_{\pi(i)} & \textit{if } i \notin s \\ 0 & \textit{if } i \in s. \end{cases}$$

If we perform the random uniform choices of a basis $\beta \in \mathcal{B}$, a set $s \in \mathcal{S}$ and a permutation $\pi \in \Pi_b$, then we obtain a randomly and uniformly selected surjective linear map $T_{\beta,s,\pi}$.

Proof. First, remark that $T_{\beta,s,\pi}$ is a surjective linear map for every choice of $\beta \in \mathcal{B}$, $s \in \mathcal{S}$ and $\pi \in \Pi_b$. From the fact that $T_{\beta,s,\pi}$ is defined for every vector of the basis β we known it may be uniquely extended to a linear map. It is surjective since for every vector $\vec{y_i}$, $i \in \{1,\ldots,b\}$, there is a vector $\vec{b_j}$, $j \in \{1,\ldots,f\}$, such that $T_{\beta,s,\pi}(\vec{b_j}) = \vec{y_i}$.

We show that for every surjective linear transformation T there is a choice of β, s and π such that $T_{\beta,s,\pi} = T$. Consider set $T^{-1}(\vec{0})$, it certainly contains $f-b$ linearly independent vectors, denote them as β_0.

Now for every $i = 1, 2, \ldots, b$ choose $\vec{c_i} \in T^{-1}(\vec{y_i})$, then $\beta_1 = \{\vec{c_1}, \vec{c_2}, \ldots, \vec{c_b}\}$ is a linearly independent set because the set $\{\vec{y_1}, \ldots, \vec{y_b}\}$ is a basis of the vector space $\mathbb{Z}_2^b$. Since $T(\beta_0) = \{\vec{0}\}$ we deduce that $\beta = \beta_0 \cup \beta_1$ is a linearly independent set of size f, thus it is a basis.

Let δ denote the number of all bases in the vector space $\mathbb{Z}_2^{f-b}$ and T be a fixed surjective linear transformation. The number of choices generating the transformation T equals $f!\delta t 2^{f-b}$. To choose a basis β of $\mathbb{Z}_2^f$ the sets β_0 and β_1 are selected independently of each other as already stated. Number of choices for β_0 is δ. The process of selection of β_1 is already described and implies that the number of choices is $t2^{f-b}$. The factor $f!$ appears because the bases of the vector space $\mathbb{Z}_2^f$ are considered ordered. Since we want to create the fixed function T the choice for s and π is already exactly determined by the previous choice of the basis β. □

Remark 5.4. *Let u, f, b be natural numbers such that $b \leq f$. Assume the random uniform choices of a linear transformation $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ among $LT(\mathbb{Z}_2^u, \mathbb{Z}_2^f)$ and a surjective function $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ among $LTS(\mathbb{Z}_2^f, \mathbb{Z}_2^b)$. Then we obtain the random uniform choice of a linear transformation $T = T_1 \circ T_0$ among $LT(\mathbb{Z}_2^u, \mathbb{Z}_2^b)$.*

Proof. The idea of the proof is to show that there exists a natural number γ such that every linear map T is generated by γ choices of T_0 and T_1.

To prove this fact, let T_1 be a fixed surjective linear map. First we show that the number of choices of T_0, such that $T = T_1 \circ T_0$, equals $u2^{f-b}$. Let β be a fixed basis of the vector space $\mathbb{Z}_2^u$. From Lemma A.10 it follows that for every $\vec{y}$ of β there are 2^{f-b} choices of $T_0(\vec{y}) \in T_1^{-1}(T(\vec{y}))$. Clearly $T = T_1 \circ T_0$.

Now for every T there are exactly $\gamma = u2^{f-b}|LTS(\mathbb{Z}_2^f, \mathbb{Z}_2^b)|$ choices of transformations T_0 and T_1. □

See Appendix A for the definitions and facts regarding the orthogonal complement, affine vector spaces and affine linear maps.

Remark 5.5. *Let $u, f, b \in \mathbb{N}$ such that $f \geq b$. Assume the random uniform choices of $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ among $LT(\mathbb{Z}_2^u, \mathbb{Z}_2^f)$ and $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ among $LTS(\mathbb{Z}_2^f, \mathbb{Z}_2^b)$. Set $T = T_1 \circ T_0$. Assume that $\vec{y} \in \mathbb{Z}_2^b$ and denote $U_A = T^{-1}(\vec{y})$ and $F_A = T_1^{-1}(\vec{y})$. If $U_A \neq \emptyset$, then $T_0|_{U_A} : U_A \to F_A$ is a random affine linear map chosen uniformly from the set $LT_A(U_A, F_A)$.*

Proof. First set $U = \mathbb{Z}_2^u$, $U_0 = T^{-1}(\vec{0})$, $F_0 = T_1^{-1}(\vec{0})$ and $U_1 = U_0^\perp$.

Now we show that there is a one-to-one relationship among the linear transformations $T_0 : U \to F$ and pairs of mappings $T_0|_{U_0}$ and $T_0|_{U_1}$. Assume that β_0 and β_1 are orthonormal bases of the vector spaces U_0 and U_1. The set $\beta = \beta_0 \cup \beta_1$ is an orthonormal basis of the vector space U, too. For a vector $\vec{x} \in \beta$ set the value $T_0(\vec{x})$ equal to $T_0|_{U_0}(\vec{x})$ or $T_0|_{U_1}(\vec{x})$, according to the presence of $\vec{x}$ in β_0 or β_1. Thus the uniform choice of a mapping T_0 corresponds to the uniform independent choices of linear functions $T_0|_{U_0}$ and $T_0|_{U_1}$.

From the observed fact it follows that the uniform choice T_0 gives the uniform selection of a linear transformation $T_0|_{U_0}$. When $\vec{y} = \vec{0}$, then, by the previous statement, we simply perform the random uniform selection of a mapping $T_0|_{U_A} = T_0|_{U_0}$.

When $\vec{y} \neq \vec{0}$, we have to randomly and uniformly select an affine mapping $T_0|_{U_A}$. By setting $T_0|_{U_A}(\vec{x}) = T_0(\vec{v}) + T_0|_{U_0}(\vec{x} - \vec{v})$ for an arbitrary but fixed vector $\vec{v} \in U_A$, the uniform choice of $T_0|_{U_0}$ implies the uniform choice of $T_0|_{U_A}$. □

5.2 Probabilistic Properties

Many of the following claims are taken from [3]. It is convenient to show the original proofs and then modify them according to our future needs. Technical definitions and statements which follow are useful in order to show our goal – the bound on the length of the longest chain.

Once again, see Appendix A for the exact definitions of the set

$$\vec{v} + A = \{\vec{v} + \vec{a} \mid \vec{a} \in A\}$$

and the set

$$A + B = \{\vec{a} + \vec{b} \mid \vec{a} \in A, \vec{b} \in B\}.$$

Original Lemma 5.6 can be found in [24], however our proof is more exact and clear.

Lemma 5.6. *Let V be a finite vector space and A be its subset. Define $\mu = 1 - \frac{|A|}{|V|}$ as the inverse density of the set A in the vector space V. Let $\vec{v} \in V$ be a random uniformly chosen vector independent of A. Then*

$$\mathbf{E}\left(1 - \frac{|A \cup (\vec{v} + A)|}{|V|}\right) = \mu^2$$

The expectation is taken over the possible choices of $\vec{v} \in V$.

Proof. To simplify further computations define $X_{\vec{v}} = |A \cup (\vec{v} + A)|$ as a random variable taken over the random uniform choice of a vector $\vec{v} \in V$. The most difficult part of the proof is to compute $\mathbf{E}(X_{\vec{v}})$. From its definition we have

$$\mathbf{E}(X_{\vec{v}}) = \sum_{\vec{v} \in V} |A \cup (\vec{v} + A)| \cdot \mathbf{Pr}(\vec{v} \text{ is chosen}) = \sum_{\vec{v} \in V} \frac{|A \cup (\vec{v} + A)|}{|V|}.$$

The size of the set $A \cup (\vec{v} + A)$ can be expressed using the indicator function as

$$|A \cup (\vec{v} + A)| = \sum_{\vec{u} \in V} I(\vec{u} \in A \vee \vec{u} \in (\vec{v} + A)).$$

To compute the sum $\sum_{\vec{v} \in V} |A \cup (\vec{v} + A)|$ we count the number of pairs of vectors $\vec{u}, \vec{v} \in V$ satisfying the indicator's condition. Notice, if $\vec{u} \in A$, then there are exactly $|V|$ vectors $\vec{v} \in V$ satisfying the above condition. To count the number of vectors confirming to the second one assume that $\vec{u} \notin A$ and $\vec{a} \in A$. There is exactly one vector $\vec{v} \in V$ for every choice of $\vec{u}$ and $\vec{a}$ such that $\vec{a} + \vec{v} = \vec{u}$. It follows that there are

exactly $|A|(|V|-|A|)$ such vectors $\vec{u}$ and $\vec{v}$ satisfying the condition. The remaining choices are refused. Thus

$$\begin{aligned}|\{(\vec{u},\vec{v}) \mid \vec{u}\in A \vee \vec{u}\in(\vec{v}+A), \vec{u},\vec{v}\in V\}| &= |A|(|V|-|A|)+|V||A| \\ &= 2|V||A|-|A|^2.\end{aligned}$$

Substituting into the definition of $\mathbf{E}(X_{\vec{v}})$ and rewriting sums into the just computed number of pairs gives that

$$\begin{aligned}\mathbf{E}(X_{\vec{v}}) &= \frac{\sum_{\vec{v}\in V}\sum_{\vec{u}\in V} I(\vec{u}\in A \vee \vec{u}\in(\vec{v}+A))}{|V|} \\ &= \frac{|\{(\vec{u},\vec{v}) \mid \vec{u}\in A \vee \vec{u}\in(\vec{v}+A), \vec{u},\vec{v}\in V\}|}{|V|} \\ &= \frac{2|V||A|-|A|^2}{|V|}.\end{aligned}$$

Now we finally compute the expected value

$$\begin{aligned}\mathbf{E}\left(1-\frac{|A\cup(\vec{v}+A)|}{|V|}\right) &= 1-\frac{\mathbf{E}(|A\cup(\vec{v}+A)|)}{|V|} \\ &= 1-\frac{\mathbf{E}(X_{\vec{v}})}{|V|} \\ &= 1-\frac{2|V||A|-|A|^2}{|V|^2} \\ &= \frac{|V|^2-2|V||A|+|A|^2}{|V|^2} \\ &= \left(1-\frac{|A|}{|V|}\right)^2 = \mu^2.\end{aligned}$$

□

The next lemma and its corollaries are so simple that one can think they should be omitted. However, they are used later in a few complicated situations. Sometimes it is hard to realise why the inequalities hold and recalling this lemma and its corollaries often helps.

Lemma 5.7. *Let $f:(0,1)\to\mathbb{R}$ be a decreasing function. Then the function $x^{f(x)}$ is increasing in the interval $(0,1)$.*

Proof. To prove the lemma assume that $a,b\in(0,1)$ and $a<b$. It follows that $f(a)>f(b)$. Observe that the function a^x of variable x is decreasing and the function $x^{f(b)}$ of variable x is increasing. Hence

$$a^{f(a)} < a^{f(b)} < b^{f(b)}.$$

□

Corollary 5.8. *The function* $x^{c+\log\log\left(\frac{1}{x}\right)}$ *is increasing in the interval* $(0,1)$ *for every* $c \in \mathbb{R}$.

Proof. Use Lemma 5.7 and note that the function $\log\log\left(\frac{1}{x}\right)$ is decreasing. □

Corollary 5.9. *The function* $x^{c-\log x-\log\log x}$ *is decreasing in the interval* $(1,\infty)$ *for every* $c \in \mathbb{R}$.

Proof. Set the function $g(y) = y^{-c+\log\left(\frac{1}{y}\right)+\log\log\left(\frac{1}{y}\right)}$ for $0 < y < 1$. From Lemma 5.7 for $g(y)$ it follows that g is increasing in $(0,1)$. Then for every $x > 1$ the function $f(x) = x^{c-\log x-\log\log x} = g\left(\frac{1}{x}\right)$. Thus the function f is decreasing since for $1 < a < b$ we have

$$f(a) = g\left(\frac{1}{a}\right) > g\left(\frac{1}{b}\right) = f(b).$$

□

The following lemma is a very technical one and is taken from [24]. It is later used to estimate probabilities of events related to the system of linear maps. These events correspond to the existence of a large set having a singleton image – to the existence of a long chain.

Lemma 5.10. *Let* $0 < \mu_0 < 1$ *be a constant and for every* i, $1 \le i \le k$, *let* μ_i *be a random variable. Let us assume that for every random variable* μ_i, $1 \le i \le k$, *the following holds:*

$$0 \le \mu_i \le \mu_{i-1}$$
$$\mathbf{E}\left(\mu_i \mid \mu_{i-1}, \ldots, \mu_1\right) = \mu_{i-1}^2.$$

Then for every constant $t \in [0,1]$

$$\mathbf{Pr}\left(\mu_k \ge t\right) \le \mu_0^{k-\log\log\left(\frac{1}{t}\right)+\log\log\left(\frac{1}{\mu_0}\right)}.$$

Proof. We prove the statement by induction over k.

The initial step, $k = 0$**.** Since μ_0 is constant we have

$$\mathbf{Pr}\left(\mu_0 \ge t\right) = \begin{cases} 0 & \text{if } \mu_0 < t \\ 1 & \text{if } \mu_0 \ge t. \end{cases}$$

From $\mu_0 > 0$ it follows that

$$\mu_0^{0-\log\log\left(\frac{1}{t}\right)+\log\log\left(\frac{1}{\mu_0}\right)} \ge 0$$

and thus the estimate holds for $0 < \mu_0 < t$.

If $t \le \mu_0 < 1$, then $-\log\log\left(\frac{1}{t}\right) + \log\log\left(\frac{1}{\mu_0}\right) \le 0$ and hence

$$\mu_0^{0-\log\log\left(\frac{1}{t}\right)+\log\log\left(\frac{1}{\mu_0}\right)} \ge 1.$$

The statement thus holds for $k = 0$.

The induction step. We prove the statement for $k+1$ using the assumption that the lemma holds for $k \geq 0$. Let $t \in (0,1)$ be fixed. For simplicity, let us denote $c = k - \log\log\left(\frac{1}{t}\right)$. Then we have to prove

$$\mathbf{Pr}\left(\mu_{k+1} \geq t\right) \leq \mu_0^{c+1+\log\log\left(\frac{1}{\mu_0}\right)}.$$

Whenever exponent $c+1+\log\log\left(\frac{1}{\mu_0}\right) \leq 0$, the estimate holds because $\mu_0 < 1$. We can restrict ourselves to the case when $c+1+\log\log\left(\frac{1}{\mu_0}\right) > 0$. To prove our statement we fix the value of the variable μ_1 and use the induction hypothesis for k. For a value $a \in [0, \mu_0]$ define $g(a) = \mathbf{Pr}\left(\mu_{k+1} \geq t \mid \mu_1 = a\right)$. First, we give the following claim.

Claim 5.11.

$$\mathbf{Pr}\left(\mu_{k+1} \geq t\right) = \int_0^{\mu_0} \mathbf{Pr}\left(\mu_{k+1} \geq t \mid \mu_1 = a\right) \mathbf{Pr}\left(\mu_1 = a\right)\, da = \mathbf{E}\left(g(\mu_1)\right).$$

Proof. This claim is a corollary of Lemma B.20. □

The following functions are convenient for using the induction hypothesis. For every $x \in (0,1)$ define functions f and f_0 as

$$f_0(x) = \begin{cases} x^{c+\log\log\left(\frac{1}{x}\right)} & \text{if } 0 < x < 1 \\ 0 & \text{if } x = 0, \end{cases}$$
$$f(x) = \min\{1, f_0(x)\}.$$

If $\beta_0 = a \in [0, \mu_0]$ is constant and for $i = 1, \ldots, k$ β_i are random variables satisfying the conditions of this lemma, then apparently $\mathbf{Pr}\left(\beta_k \geq t\right) = g(a)$.

Claim 5.12. *For every $a \in [0, \mu_0]$ we have $g(a) \leq f(a)$.*

Proof. We omit the constant μ_0 from the sequence of random variables and use the previous fact for $\beta_i = \mu_{i+1}$ for $i = 0, \ldots, k$. The induction hypothesis for k then states $g(a) \leq f(a) = a^{k-\log\log\left(\frac{1}{t}\right)+\log\log\left(\frac{1}{a}\right)}$. □

These claims are used later in the proof. Now we investigate the behaviour of the function $\frac{f_0}{x}$ in the interval $(0,1)$. First, we compute the first derivation of the function $\frac{f_0}{x}$ for every $x \in (0,1)$.

$$\begin{aligned}\left(\frac{f_0(x)}{x}\right)' &= \frac{x f_0(x)\left[\ln(x)\left(c+\log\log\left(\frac{1}{x}\right)\right)\right]' - f_0(x)}{x^2} \\ &= \frac{f_0(x)\left[c+\log\log\left(\frac{1}{x}\right) + x \cdot \frac{\ln x}{\log\left(\frac{1}{x}\right)\ln 2} \cdot \frac{x}{\ln 2} \cdot \frac{-1}{x^2}\right] - f_0(x)}{x^2} \\ &= \left(c - 1 + \log\log\left(\frac{1}{x}\right) + \log e\right)\frac{f_0(x)}{x^2}.\end{aligned}$$

The investigated function $\frac{f_0(x)}{x}$ has the unique *stationary point* $x_s = 2^{-2^{-c+1-\log e}}$. First it is increasing in the interval $(0, x_s]$ and then decreasing in the interval $[x_s, 1)$.

Let us also define $x_1 = 2^{-2^{-c}}$, the *point where* $f_0(x)$ *reaches 1 for the first time*,

$$f_0(x_1) = {x_1}^{c+\log\log\left(\frac{1}{x_1}\right)} = {x_1}^{c-c} = 1.$$

Since $-c > -c+1-\log e$, the inequality $x_1 < x_s$ holds and thus point x_1 is still in the increasing phase of the function $\frac{f_0(x)}{x}$. By Corollary 5.8 the function $f_0(x)$ is increasing in the interval $[x_1, 1)$. Therefore for every $x \in [x_1, 1)$ we have that $f_0(x) \geq f_0(x_1) = 1$ and $f(x) = \min\{1, f_0(x)\} = 1$.

At last, define the *third point* $x_2 = \sqrt{x_1} = 2^{-2^{-c-1}}$. The inequality $x_2 > x_s$ follows from $-c-1 < -c+1-\log e$. The point x_2 then lies in the decreasing phase of the function $\frac{f_0(x)}{x}$. For x_2 the following statement holds:

$$\begin{aligned}
\frac{f_0(x_2)}{x_2} &= \frac{\left(2^{-2^{-c-1}}\right)^{c+\log\left(-\log\left(2^{-2^{-c-1}}\right)\right)}}{2^{-2^{-c-1}}} \\
&= \frac{\left(2^{-2^{-c-1}}\right)^{c+\log\left(2^{-c-1}\right)}}{2^{-2^{-c-1}}} \\
&= \frac{\left(2^{-2^{-c-1}}\right)^{-1}}{2^{-2^{-c-1}}} \\
&= \frac{1}{x_2^2} = \frac{1}{x_1}.
\end{aligned}$$

Claim 5.13. *The following holds:*

(1) $\frac{f_0(x_1)}{x_1} = \frac{1}{x_1}$,

(2) $\frac{f_0(x_2)}{x_2} = \frac{1}{x_1}$,

(3) $\frac{f(x)}{x} \leq \frac{f_0(\mu_0)}{\mu_0}$ *if* $0 < x \leq \mu_0 \leq x_s$,

(4) $\frac{1}{x_1} \leq \frac{f_0(\mu_0)}{\mu_0}$ *if* $\mu_0 \in [x_s, x_2]$.

Proof. The first fact $\frac{f_0(x_1)}{x_1} = \frac{1}{x_1}$, follows from the definition of x_1 because $f_0(x_1) = 1$.

The proof of the second one is above.

The third one comes from the observation that the function $\frac{f_0(x)}{x}$ is increasing in the interval $(0, x_s]$. In addition, $f(x) \leq f_0(x)$ in the interval $(0, 1)$.

The last one comes from the equality $\frac{1}{x_1} = \frac{f_0(x_2)}{x_2}$ and the fact that both μ_0 and x_2 lie in the decreasing phase. Thus the relation $\mu_0 \leq x_2$ implies the result

$$\frac{f_0(x_2)}{x_2} \leq \frac{f_0(\mu_0)}{\mu_0}.$$

□

The proof is now divided into three cases. The first and the second case take care about the situation when the exponent $c+1+\log\log\left(\frac{1}{\mu_0}\right)$ is non-negative. In the first two cases it is proved that $f(x) \leq \frac{f_0(\mu_0)x}{\mu_0}$ for every $x \in (0, \mu_0]$.

Constant μ_0 is in the increasing phase, $\mu_0 \leq x_s$. From Claim 5.13(3) it follows that

$$\frac{f(x)x}{x} \leq \frac{f_0(\mu_0)x}{\mu_0}.$$

Constant μ_0 is in the decreasing phase, $x_s \leq \mu_0 \leq x_2$.

Assume that $x \in (0, x_1]$. The Claim 5.13(1) and the fact that the function is increasing in the interval imply

$$\frac{f(x)}{x} \leq \frac{f_0(x)}{x} \leq \frac{f_0(x_1)}{x_1} = \frac{1}{x_1}.$$

Let $x \in [x_1, \mu_0]$. Because of the choice $x \geq x_1$ we have that $f(x) = 1$ and it follows that

$$\frac{f(x)}{x} = \frac{1}{x} \leq \frac{1}{x_1}.$$

For every $x \in (0, \mu_0]$ we may estimate the function $f(x)$ as

$$f(x) = \frac{f(x)x}{x} \leq \frac{x}{x_1}.$$

The Claim 5.13(4) proves that

$$f(x) \leq \frac{x}{x_1} \leq \frac{f_0(\mu_0)x}{\mu_0}.$$

Proof of the lemma assuming either of the first two cases. In both cases for every $x \in (0, \mu_0]$ the following holds

$$f(x) \leq \frac{f_0(\mu_0)x}{\mu_0}.$$

Now use the previous estimate, Claim 5.12, Claim 5.11 and properties of the expected value stated in Lemma B.18 to prove the lemma,

$$\begin{aligned}\mathbf{Pr}\left(\mu_{k+1} \geq t\right) = \mathbf{E}\left(g(\mu_1)\right) \leq \mathbf{E}\left(f(\mu_1)\right) \leq \mathbf{E}\left(\frac{f_0(\mu_0)\mu_1}{\mu_0}\right) = \frac{f_0(\mu_0)}{\mu_0}\mathbf{E}\left(\mu_1 \mid \mu_0\right) \\ = \frac{f_0(\mu_0)}{\mu_0}\mu_0^2 = \mu_0 f_0(\mu_0) = \mu_0^{\,c+1+\log\log\left(\frac{1}{\mu_0}\right)}.\end{aligned}$$

Constant μ_0 is set so that the exponent is negative, $\mu_0 \geq x_2$. To prove the lemma in this case, it is sufficient to show that the exponent, $c + 1 + \log\log\left(\frac{1}{\mu_0}\right)$, is not positive. As already observed before the estimate is then at least 1. The exponent is non-positive since

$$\begin{aligned} c + 1 + \log\log\left(\frac{1}{\mu_0}\right) &\leq c + 1 + \log\log\left(\frac{1}{x_2}\right) \\ &= c + 1 + \log\left(-\log\left(2^{-2^{-c-1}}\right)\right) \\ &= c + 1 + \log\left(2^{-c-1}\right) \\ &= c + 1 - c - 1 = 0. \end{aligned}$$

The inequality, $\mathbf{Pr}\left(\mu_{k+1} \geq t\right) \leq \mu^{c+1+\log\log\left(\frac{1}{\mu_0}\right)}$, holds for every of the three cases and the induction step is thus complete. □

Theorem 5.14 is also taken from [24] and is used to estimate the probability of the event that a subset of the domain is not mapped onto whole target space.

Theorem 5.14. *Let $f, b \in \mathbb{N}$ such that $b \leq f$. Let S be a proper and non-empty subset of the vector space $\mathbb{Z}_2^u$. Set $\mu = 1 - \frac{|S|}{2^f}$ as the inverse density of the set S in the vector space $\mathbb{Z}_2^f$. Then for a random uniformly chosen surjective linear map $T : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ we have*

$$\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) \leq \mu^{f-b-\log b+\log\log\frac{1}{\mu}}.$$

Proof. Set $k = f - b$. Choose vectors $\vec{v_1}, \ldots, \vec{v_k} \in \mathbb{Z}_2^f$ independently and randomly using the uniform distribution. Note that the vectors $\vec{v_1}, \ldots, \vec{v_k} \in \mathbb{Z}_2^f$ are not necessarily linearly independent.

We perform the random uniform choice of a linear surjective transformation T according to Model 5.3. First fix a basis of the target space $\mathbb{Z}_2^b$. Secondly, maximal linearly independent subset of vectors $\vec{v_1}, \ldots, \vec{v_k}$ is extended to a random basis β of the vector space $\mathbb{Z}_2^f$. Now choose a random permutation $\pi \in \Pi_b$ as stated in the mentioned model. Because of the random uniform selection of the vectors $\vec{v_1}, \ldots, \vec{v_k}$, the basis β may be chosen uniformly as well. If we place the vectors $\vec{v_1}, \ldots, \vec{v_k}$ into the kernel of the created mapping, then they define a subset s' of the needed set $s \in \mathcal{S}$. The set s' is then uniformly extended to a set $s \in S$ such that $s' \subseteq s$ so that we perform the uniform choice of a set $s \in \mathcal{S}$. The random uniform choice of a surjective linear function T is finally complete. Just note that we have $T(\vec{v_i}) = \vec{0}$ for all $i = 1, \ldots, k$.

Let us define a bounded sequence of sets $S_0 = S$ and $S_i = S_{i-1} \cup (S_{i-1} + \vec{v_i})$ and set $\mu_i = 1 - \frac{|S_i|}{2^f}$ for $i \in \{1, \ldots, k\}$. When considering μ_i random variables, then by using Lemma 5.6 we can derive that $\mathbf{E}\left(\mu_i\right) = \mu_{i-1}^2$ for every $i \in \{1, \ldots, k\}$. Because every set S_i is an extension of the previous set S_{i-1} it follows that $0 < \mu = \mu_0 < 1$ and $\mu_i \leq \mu_{i-1}$ for all $i = 1, \ldots, k$. The assumptions of Lemma 5.10 are now satisfied

and we obtain

$$\begin{aligned}\mathbf{Pr}\left(\mu_k \geq 2^{-b}\right) &\leq \mu^{k-\log\log\left(\frac{1}{2^{-b}}\right)+\log\log\left(\frac{1}{\mu}\right)} \\ &= \mu^{k-\log b+\log\log\left(\frac{1}{\mu}\right)} \\ &= \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)}.\end{aligned}$$

If we prove that the event $T(S_k) = \mathbb{Z}_2^b$ occurs whenever $\mu_k < 2^{-b}$, then the statement will be almost proved. Since $\mu_k = 1 - \frac{|S_k|}{2^f}$ the size of the set S_k equals $2^f(1 - \mu_k)$. Using the assumption $\mu_k < 2^{-b}$ it follows that $|S_k| > 2^f - 2^{f-b}$. To get the contradiction we assume that there is a vector $\vec{x} \in \mathbb{Z}_2^b - T(S_k)$ or equivalently $T(S_k) \neq \mathbb{Z}_2^b$. Under these conditions it is clear that $T^{-1}(\vec{x})$ and S_k are disjoint. From Lemma A.10 we have $|T^{-1}(\vec{x})| = 2^{f-b}$. It follows that

$$2^f = |\mathbb{Z}_2^f| \geq |S_k \cup T^{-1}(\vec{x})| > 2^f - 2^{f-b} + 2^{f-b} = 2^f.$$

This is a contradiction.

To prove the theorem rewrite the statement, $\mu_k < 2^{-b} \Rightarrow T(S_k) = \mathbb{Z}_2^b$, in terms of probability:

$$\mathbf{Pr}\left(\mu_k < 2^{-b}\right) \leq \mathbf{Pr}\left(T(S_k) = \mathbb{Z}_2^b\right).$$

Because $\mathbb{Z}_2^f$ is a vector space over the field $\mathbb{Z}_2$, by induction over $i \in \{1, \ldots, k\}$, we obtain that $S_i = S_{i-1} \cup (\vec{v_i} + S_{i-1}) = S_0 + \operatorname{span}(\vec{v_1}, \ldots, \vec{v_i})$. Note that $T(\vec{v_i}) = \vec{0}$ because the mapping T is chosen so that every vector $\vec{v_i}$ is placed in the kernel of T. This simply implies $T(S_k) = T(S)$.

The proof of the theorem is finished by putting the previous notes together,

$$\begin{aligned}\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) &= \mathbf{Pr}\left(T(S_k) \neq \mathbb{Z}_2^b\right) \\ &= 1 - \mathbf{Pr}\left(T(S_k) = \mathbb{Z}_2^b\right) \\ &\leq 1 - \mathbf{Pr}\left(\mu_k < 2^{-b}\right) \\ &= \mathbf{Pr}\left(\mu_k \geq 2^{-b}\right) \\ &\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)}.\end{aligned}$$

□

Theorem 5.15 shows the probability of the complementary event, $T(S) = \mathbb{Z}_2^b$, if the set S is large enough. It is taken from [24] and is consequently improved by our Statements 5.18 and 5.23.

Theorem 5.15. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear map. For every $\epsilon \in (0, 1)$ there is a constant $c_\epsilon > 0$ such that for every subset S of the domain $\mathbb{Z}_2^u$ with $|S| \geq c_\epsilon b 2^b$ we have that*

$$\mathbf{Pr}\left(T(S) = \mathbb{Z}_2^b\right) \geq 1 - \epsilon.$$

Proof. The idea of the proof is to factorise the transformation T through the factor space $\mathbb{Z}_2^f$, where the dimension f is specified later, into two linear functions T_0 and T_1 such that $T = T_1 \circ T_0$. In fact, we estimate the probability of the complementary event, $T(S) \neq \mathbb{Z}_2^b$. To successfully use the estimate from Theorem 5.14 we must bound the inverse density μ. The bound is obtained by using the Law of Total Probability, Theorem B.13 as

$$\begin{aligned}&\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right)\\ &\quad = \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| \leq \frac{|S|}{2}\right) + \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{2}\right).\end{aligned}$$

Then we use the Markov's inequality to estimate the expression $\mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{2}\right)$ and Theorem 5.14 on the remaining one.

First set $f = \left\lceil \log(\frac{2|S|}{\epsilon}) \right\rceil$. Let $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be a random uniformly chosen surjective linear mapping. Since c_ϵ is later chosen large enough, we have that $f \geq b$ and thus such onto mapping T_1 exists. Fix it. Secondly, perform the random uniform choice of a linear mapping $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$. From Model 5.4 it follows that the linear mapping $T = T_1 \circ T_0$ is chosen randomly and uniformly, too.

Since the family of all linear mappings from $\mathbb{Z}_2^u$ into $\mathbb{Z}_2^f$ is 1-universal we conclude that

$$\mathbf{Pr}\left(T_0(\vec{x}) = T_0(\vec{y})\right) = 2^{-f}$$

for all distinct vectors $\vec{x}$ and $\vec{y}$ from the vector space $\mathbb{Z}_2^u$. If d_S is the number of all pairs of distinct vectors $\vec{x}, \vec{y} \in S$ with $T_0(\vec{x}) = T_0(\vec{y})$, then the expected value of the random variable d_S is

$$\mathbf{E}\left(d_S\right) = \binom{|S|}{2} 2^{-f}.$$

If $|T_0(S)| \leq \frac{|S|}{2}$ then there exist at least $\frac{|S|}{2}$ pairs of distinct vectors $\vec{x}, \vec{y} \in S$ with $T_0(\vec{x}) = T_0(\vec{y})$. By the Markov's inequality, Theorem B.21, for every $k > 1$ we have

$$\mathbf{Pr}\left(d_S \geq k\binom{|S|}{2}2^{-f}\right) \leq \frac{1}{k}.$$

Thus if we set $k = \frac{|S|2^f}{2\binom{|S|}{2}}$, then

$$k \geq \frac{2|S|^2}{\epsilon(|S|^2 - |S|)} \geq 2\epsilon^{-1} > 1$$

and

$$k\binom{|S|}{2}2^{-f} = \frac{|S|2^f}{2\binom{|S|}{2}}\binom{|S|}{2}2^{-f} = \frac{|S|}{2}$$

and, by the Markov's inequality, we obtain

$$\mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{2}\right) \leq \mathbf{Pr}\left(d_S \geq \frac{|S|}{2}\right) \leq \frac{2\binom{|S|}{2}}{|S|2^f} = \frac{|S|-1}{2^f} < \frac{|S|}{2^f} \leq \frac{\epsilon|S|}{2|S|} = \frac{\epsilon}{2}.$$

We can summarise that

$$\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| \leq \frac{|S|}{2}\right) \leq \frac{\epsilon}{2}.$$

Secondly we compute $\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{2}\right)$. By Theorem 5.14 used for the mapping $T_1 : \mathbb{Z}_2^f \rightarrow \mathbb{Z}_2^b$ and the set $T_0(S) \subseteq \mathbb{Z}_2^f$ we have

$$\mathbf{Pr}\left(T(S) = T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{2}\right) \leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)}$$

where $\mu = 1 - \frac{|T_0(S)|}{2^f}$. Since $2^f \leq \frac{4|S|}{\epsilon}$, clearly

$$\mu = 1 - \frac{|T_0(S)|}{2^f} < 1 - \frac{|S|}{2 \cdot 2^f} \leq 1 - \frac{\epsilon|S|}{8|S|} \leq e^{-\frac{\epsilon}{8}}.$$

In the following constant c_ϵ is chosen as $4\left(\frac{2}{\epsilon}\right)^{\frac{8}{\epsilon}}$. Then we can estimate:

$$\begin{aligned}
&-\frac{\epsilon}{8}\left(f - b - \log b + \log\log\left(\frac{1}{\mu}\right)\right) \\
&\quad = -\frac{\epsilon}{8}\left(\left\lceil \log\left(\frac{2|S|}{\epsilon}\right)\right\rceil - b - \log b + \log\log\left(\frac{1}{\mu}\right)\right) \\
&\quad \leq -\frac{\epsilon}{8}\left(\left\lceil \log\left(\frac{8\left(\frac{2}{\epsilon}\right)^{\frac{8}{\epsilon}} b2^b}{\epsilon}\right)\right\rceil - b - \log b + \log\log\left(\frac{1}{\mu}\right)\right) \\
&\quad \leq -\frac{\epsilon}{8}\left(3 + \frac{8}{\epsilon}\log\frac{2}{\epsilon} - \log\epsilon + \log b + b - b - \log b + \log\left(\left(\frac{\epsilon}{8}\right)\log e\right)\right) \\
&\quad = -\frac{\epsilon}{8}\left(3 - \log\epsilon + \frac{8}{\epsilon}\log\frac{2}{\epsilon} + \log\epsilon - 3 + \log\log e\right) \\
&\quad = -\frac{\epsilon}{8}\left(\frac{8}{\epsilon}\log\frac{2}{\epsilon} + \log\log e\right) \\
&\quad = \log\frac{\epsilon}{2} - \frac{\epsilon}{8}\log\log e \\
&\quad \leq \log\frac{\epsilon}{2}.
\end{aligned}$$

And for the calculated probability we derive:

$$\begin{aligned}
\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T(S)| > \frac{|S|}{2}\right) &\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)} \\
&\leq e^{-\frac{\epsilon\left(f-b-\log b+\log\log\left(\frac{1}{\mu}\right)\right)}{8}} \\
&\leq e^{\log\left(\frac{\epsilon}{2}\right)} \leq e^{\ln\left(\frac{\epsilon}{2}\right)} = \frac{\epsilon}{2}.
\end{aligned}$$

If we put both alternatives together, we deduce that

$$\mathbf{Pr}\left(T(S) = T_1(T_0(S)) \neq \mathbb{Z}_2^b\right) \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

From this observation follows the required estimate:

$$\mathbf{Pr}\left(T(S) = \mathbb{Z}_2^b\right) \geq 1 - \epsilon.$$

□

The biggest disadvantage of Theorem 5.15 is the great value of constant c_ϵ. As follows from Theorem 5.38 and Theorem 5.40 it is necessary to decrease its value. Since the other constants are proportional to it, we need to find the smallest value of c_ϵ. It is the aim of the following theorems and corollaries. When we successfully manage the goal, then we are able to propose a reasonable chain limit rule, bounding the length of the longest chain.

As observed in the next corollary, we may apply Theorem 5.15 for two affine vector subspaces and an affine linear transformation between them, too.

Corollary 5.16. *Let U_A and F_A be affine vector subspaces of the vector spaces $\mathbb{Z}_2^u$ and $\mathbb{Z}_2^f$. Let $T_A : U_A \to F_A$ be a random uniformly chosen affine linear map, $\epsilon \in (0, 1)$, constant c_ϵ be chosen so that Theorem 5.15 is satisfied and $S_A \subseteq U_A$ such that $|S_A| \geq c_\epsilon |F_A| \log |F_A|$. Then*

$$\mathbf{Pr}\left(T_A(S) = F_A\right) \geq 1 - \epsilon.$$

Proof. See Appendix A for the definitions of affine linear spaces and transformations. Recall Definition A.3 saying that $U_A = \vec{u} + U_0$ for a vector subspace $U_0 \leq U$ and a vector $\vec{u} \in U$. Similarly we can assume that $F_A = \vec{f} + F_0$ for $F_0 \leq F$ and $\vec{f} \in F$. From Definition A.8 of the affine linear mapping T_A it follows that $T_A(\vec{x}) = \vec{f} + T_0(\vec{x} - \vec{u})$ for a fixed linear transformation $T_0 : U_0 \to F_0$ and for every vector $\vec{x} \in F_A$.

We known that U_0, F_0 are vector spaces. By setting $S_0 = S_A - \vec{u} \subseteq U_0$ it follows that $|S_0| \geq c_\epsilon |F_0| \log |F_0|$. We can apply Theorem 5.15 and we obtain $\mathbf{Pr}\left(T_0(S_0) = F_0\right) \geq 1 - \epsilon$.

Now we must realise that the result is valid in the affine case, too. This is simple, since $T_0(S_0) = F_0$ if and only if $T_A(S_A) = F_A$. From Lemma A.4 it follows that $F_A = T_A(\vec{u}) + F_0$. The relation $T_0(S_0) = F_0 \Leftrightarrow T_A(S_A) = F_A$ holds since

$$\begin{aligned}(\vec{x_0} \in S_0 \Rightarrow T_0(\vec{x_0}) \in F_0) &\Leftrightarrow (\vec{x_0} \in S_0 \Rightarrow T_A(\vec{u} + \vec{x_0}) \in T_A(\vec{u}) + F_0) \\ &\Leftrightarrow (\vec{x_A} = \vec{u} + \vec{x_0} \in S_A, \vec{x_0} \in S_0 \Rightarrow T_A(\vec{x_A}) \in F_A).\end{aligned}$$

□

5.3 Parametrisation of The Original Proofs

In order to obtain the minimal value of the constant c_ϵ we present a simple parametrisation of the proof of the previous theorem. Combination of this simple powerful technique with the novel ideas of Statement 5.23 brings a new reasonable result. The result is used later in Theorem 5.41 and exploited by the proposed model of hashing in Chapter 6.

As mentioned, parametrisation of the proof of Theorem 5.15 is done. The optimisation of the value c_ϵ itself is not performed analytically because of its complexity caused by the emerged constraints. Instead, we created a straightforward computer program that makes it numerically. Each parameter is assigned a value from a predefined interval. The intervals were chosen manually so that the assumptions of the theorems are satisfied and a reasonable result is achieved. Program enumerates the values of the intervals uniformly with the prescribed step set for each interval. The constant c_ϵ is then computed for every assignment and the minimal value is remembered along with the parametrisation. The parameters then allow the verification of all the assumptions of all the claims stated in the proof of a parametrised theorem.

In the following text we use the assumptions and notation from the proof of Theorem 5.15. The next lemma allows a parametrisation of the size of the set $T_0(S)$. In the moment when we use the Law of Total Probability we split the proof of Theorem 5.15 into two cases. We distinguish between them according to the size of the set $T_0(S)$. The question is what happens if another value than $\frac{|S|}{2}$ is chosen. The parameter k, $k \geq 1$, corresponds to the choice of a different size, now set to $\frac{|S|}{k}$.

Lemma 5.17. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ be a function, $S \subseteq \mathbb{Z}_2^u$ and $k \in \mathbb{R}$ such that $k \geq 1$. If $|T_0(S)| \leq \frac{|S|}{k}$, then T_0 has at least $\frac{|S|(k-1)}{2}$ collisions of elements from S.*

Proof. Define the family $\{b_i \in \mathbb{N}_0\}_{i \in T_0(S)}$ such that $b_i = \left|S \cap T_0^{-1}(i)\right|$. First note that $\sum_{i \in T_0(S)} b_i = |S|$. The Cauchy Bunyakovsky Schwarz inequality for the vectors $\{\vec{b_i}\}_{i \in T_0(S)}$ and $\{\vec{1}\}_{i \in T_0(S)}$ states that

$$\left(\sum_{i \in T_0(S)} b_i^2\right)\left(\sum_{i \in T_0(S)} 1^2\right) \geq \left(\sum_{i \in T_0(S)} b_i \cdot 1\right)^2$$
$$\sum_{i \in T_0(S)} b_i^2 \geq \frac{|S|^2}{|T_0(S)|}.$$

The number of all colliding pairs can be computed now as

$$\begin{aligned}|\{\{x,y\} \mid x \neq y \in S, T_0(x) = T_0(y)\}| &= \frac{1}{2}\sum_{i \in T_0(S)} b_i(b_i - 1) \\ &\geq \frac{|S|}{2}\left(\frac{|S|}{|T_0(S)|} - 1\right) \\ &\geq \frac{|S|(k-1)}{2}.\end{aligned}$$

□

Statement 5.18. *For every $\epsilon \in (0,1)$ Theorem 5.15 holds if the constant c_ϵ for some parameters $k, l \geq 1$ satisfies the inequalities:*

$$c_\epsilon \geq 2^{\frac{\log\left(\left(\epsilon - \frac{\epsilon}{(k-1)2^l}\right)\mu'^{\log\epsilon - l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log\mu'}} \text{ where } \mu' = 1 - \frac{\epsilon}{2k2^l} \tag{5.19}$$

and

$$l + \log c_\epsilon + \log b - log\epsilon \geq 0.$$

Proof. We parametrise the original proof of Theorem 5.15 by two real variables k and l. As already mentioned in the introduction of Lemma 5.17, the parameter k controls the size of the set $T_0(S)$. The limit is changed to $\frac{|S|}{k}$ for $k \in \mathbb{R}$, $k \geq 1$. Another constant is present in the dimension f of the factor space F, $f = \left\lceil \log\left(\frac{2|S|}{\epsilon}\right)\right\rceil$. The second parameter l is obtained by setting $f = \left\lceil \log\left(\frac{2^l|S|}{\epsilon}\right)\right\rceil$.

We summarise the claims of the original proof and recall its objects that are present in the current one. We factorise a random uniformly chosen linear transformation $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ through the vector space $\mathbb{Z}_2^f$. From Model 5.4 we obtained two linear mappings $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ and a T_1 from $\mathbb{Z}_2^f$ onto $\mathbb{Z}_2^b$. For the existence of a surjective transformation T_1 we need $f \geq b$. Since

$$f = \left\lceil \log\left(\frac{2^l|S|}{\epsilon}\right)\right\rceil \geq l + \log c_\epsilon + b + \log b - \log \epsilon$$

then from the inequality $l + \log c_\epsilon + \log b - \log \epsilon \geq 0$ it follows $f \geq b$.

We continue by using the Law of Total Probability as in the original proof:

$$\begin{aligned}
&\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) \\
&\quad = \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| \leq \frac{|S|}{k}\right) + \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right) \\
&\quad \leq \mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{k}\right) + \mathbf{Pr}\left(T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right).
\end{aligned}$$

To satisfy the required statement we show $\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) \leq \epsilon$. The required inequality is obtained by choosing a convenient value of c_ϵ. Of course, the value is computed according to the values of the parameters. The estimate of c_ϵ is further modified when compared to the original proof. Value of c_ϵ is computed directly without any inevitable estimates only specifying its accuracy.

The probability of the first event, $|T_0(S)| \leq \frac{|S|}{k}$, is estimated using the Markov's inequality, too. By Lemma 5.17 the number of collisions d_S is at least $\frac{|S|(k-1)}{2}$. The expected number of collisions remains equal to $\binom{|S|}{2}2^{-u}$. We estimate the probability as

$$\begin{aligned}
\mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{k}\right) &\leq \mathbf{Pr}\left(d_S \geq \frac{|S|(k-1)}{2}\right) \\
&\leq \frac{\binom{|S|}{2}2^{-f}}{\frac{|S|(k-1)}{2}} \\
&= \frac{|S|-1}{(k-1)2^f}.
\end{aligned}$$

We state the obtained bound as a standalone claim.

Claim 5.20.

$$\mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{k}\right) \leq \frac{|S|-1}{(k-1)2^f}.$$

Our estimate of the probability of the event, $T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}$, is based on Theorem 5.14. This is also similar to the original proof. Recall that Theorem 5.14 is used for the set $T_0(S)$, the transformation T_1, the source vector space $\mathbb{Z}_2^f$ and the target vector space $\mathbb{Z}_2^b$. The corresponding inverse density is $\mu = 1 - \frac{|T_0(S)|}{2^f}$. Then

$$\mathbf{Pr}\left(T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right) \leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)}.$$

The following upper bound on the inverse density μ becomes soon very useful. In the last inequality we used a tight upper bound following from the definition of the dimension $f = \left\lceil \log\left(\frac{2^l|S|}{\epsilon}\right)\right\rceil$,

$$\mu = 1 - \frac{|T_0(S)|}{2^f} \leq 1 - \frac{|S|}{k2^f} \leq 1 - \frac{\epsilon}{2k2^l}.$$

The assumption of Theorem 5.14 placed on the set S

$$|S| \geq c_\epsilon 2^b b$$

and the choice of the dimension f give that

$$f = \left\lceil \log\left(\frac{2^l|S|}{\epsilon}\right)\right\rceil \geq l + \log c_\epsilon + b + \log b - \log\epsilon.$$

The bound on the investigated probability is obtained from Corollary 5.8 and by putting together all the above inequalities:

$$\begin{aligned}
&\mathbf{Pr}\left(T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right)\\
&\quad\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)}\\
&\quad\leq \left(1-\frac{\epsilon}{2k2^l}\right)^{l+\log c_\epsilon+b+\log b-\log\epsilon-b-\log b+\log\log\left(\frac{1}{1-\frac{\epsilon}{2k2^l}}\right)}\\
&\quad\leq \left(1-\frac{\epsilon}{2k2^l}\right)^{l+\log c_\epsilon-\log\epsilon+\log\log\left(\frac{1}{1-\frac{\epsilon}{2k2^l}}\right)}.
\end{aligned}$$

Put $\mu' = 1 - \frac{\epsilon}{2k2^l}$. These ideas completed an important part of the proof and are summarised in the following claim.

Claim 5.21.

$$\mathbf{Pr}\left(T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right) \leq \mu'^{l+\log c_\epsilon-\log\epsilon+\log\log\left(\frac{1}{\mu'}\right)}.$$

The probability estimate of $\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right)$ follows from Claim 5.20 and Claim 5.21.

$$\begin{aligned}
&\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) \\
&\quad \leq \mathbf{Pr}\left(|T_0(S)| \leq \frac{|S|}{k}\right) + \mathbf{Pr}\left(T_1(T_0(S)) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{|S|}{k}\right) \\
&\quad \leq \frac{|S|}{(k-1)2^f} + \mu^{l+\log c_\epsilon - \log \epsilon + \log\log\left(\frac{1}{\mu}\right)} \\
&\quad \leq \frac{\epsilon|S|}{(k-1)|S|2^l} + \mu'^{l+\log c_\epsilon - \log \epsilon + \log\log\left(\frac{1}{\mu'}\right)} \\
&\quad \leq \frac{\epsilon}{(k-1)2^l} + \mu'^{l+\log c_\epsilon - \log \epsilon + \log\log\left(\frac{1}{\mu'}\right)}.
\end{aligned}$$

Our goal is to choose c_ϵ such that it is minimal possible and this probability will be less than ϵ. Thus we can compute

$$\begin{aligned}
\frac{\epsilon}{(k-1)2^l} + \mu'^{\log c_\epsilon + l - \log \epsilon + \log\log\left(\frac{1}{\mu'}\right)} &\leq \epsilon \\
\mu'^{\log c_\epsilon}\mu'^{l - \log \epsilon + \log\log\left(\frac{1}{\mu'}\right)} &\leq \epsilon - \frac{\epsilon}{(k-1)2^l} \\
\left(\epsilon - \frac{\epsilon}{(k-1)2^l}\right)\mu'^{\log \epsilon - l - \log\log\left(\frac{1}{\mu'}\right)} &\geq \mu'^{\log c_\epsilon} \\
\frac{\log\left(\left(\epsilon - \frac{\epsilon}{(k-1)2^l}\right)\mu'^{\log \epsilon - l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log \mu'} &\leq \log c_\epsilon.
\end{aligned}$$

Thus c_ϵ may be chosen minimal so that it satisfies Inequality 5.19:

$$c_\epsilon \geq 2^{\frac{\log\left(\left(\epsilon - \frac{\epsilon}{(k-1)2^l}\right)\mu'^{\log \epsilon - l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log \mu'}}.$$

□

In order to obtain a better value of c_ϵ we parametrised the proof of Theorem 5.15. The value is computed for every choice of parameters k and l directly from the above expression. The next corollary states the best value of c_ϵ we have managed to achieve.

Corollary 5.22. *For $\epsilon = 0.98$ the value of the constant c_ϵ may be chosen as 67.77.*

Proof. Choose $k = 3.28$ and $l = 0.5$, then certainly $f \geq b$. Since there are no other assumptions this choice is valid. □

5.4 Improving the Results

When we use the estimate of Statement 5.18 in Theorem 5.38, of course, we get smaller values of the multiplicative constant in the estimate of $\mathbf{E}\,(\mathbf{lpsl})$. However, the constant is on the border of being practical. More precisely said, usage of the bound becomes suitable only for hashing enormous sets. Although such sets are not uncommon, they are rarely present in everyday use. The worst case guarantee plays even more important role for such large sets. However the question is, if we are able to further improve it. And the answer is yes, if we bring some new ideas. In addition, the chain limit rule based on the proposed improvement starts beating the linear one, $\mathbf{lpsl} \leq n$, for n in the order of hundreds.

We bring novel ideas and again improve the value of c_ϵ in the next statement.

Statement 5.23. *For every $\epsilon \in (0,1)$ Theorem 5.15 holds if the constant c_ϵ for some parameters $k, l \geq 1$ satisfies the inequalities:*

$$c_\epsilon \geq 2^{\frac{\log\left(\left(\epsilon - \frac{2^l}{2^l k - 2}\right)\cdot {\mu'}^{l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log \mu'}} \quad \textit{where } \mu' = 1 - \frac{1}{k} \tag{5.24}$$

and

$$\log b + \log c_\epsilon - l \geq 0.$$

Proof. The proof we present is fully parametrised like that of Statement 5.18. We can choose values of its arguments to optimise the value of the constant c_ϵ. To briefly present the sketch of the proof recall that we need a factor space $\mathbb{Z}_2^f$, $f \geq b$. Then we decompose a uniformly chosen random linear map T into two mappings T_0 and a surjective T_1 such that $T = T_1 \circ T_0$ going through the vector space $\mathbb{Z}_2^f$.

Instead of making the factor space $\mathbb{Z}_2^f$ larger than the set S, we bound its size between $\frac{|S|}{2^l}$ and $\frac{2|S|}{2^l}$ where $l \geq 1$ is a parameter. In order to exist a surjective linear transformation from $\mathbb{Z}_2^f$ onto $\mathbb{Z}_2^b$ it must be $|\mathbb{Z}_2^f| \geq |\mathbb{Z}_2^b|$. By the inequality $\log b + \log c_\epsilon - l \geq 0$ and the requirement $|S| \geq c_\epsilon b 2^b$ it will be satisfied.

For every value $\frac{|S|}{2^l}$, $l \in \mathbb{R}, l \geq 1$, there is an uniquely determined number $f \in \mathbb{N}$ such that $\frac{|S|}{2^l} \leq 2^f \leq \frac{2|S|}{2^l}$. More formally $f = \left\lceil \log\left(\frac{|S|}{2^l}\right)\right\rceil$. Then

$$f = \lceil \log |S| \rceil - l \geq \lceil \log c_\epsilon + \log b \rceil - l + b \geq b.$$

The second novel idea is not to make the size of the set $T_0(S)$ relative to that of the set S. We refer to the size of the factor space $\mathbb{Z}_2^f$ instead. We introduce a new parameter $k \in \mathbb{R}$, $k \geq 1$. We show, if $|T_0(S)| \leq \frac{2^f}{k}$, then there are at least $\frac{|S|}{2}\left(\frac{k|S|}{2^f} - 1\right)$ collisions caused by T_0. If $|T_0(S)| \leq \frac{|S|}{k'}$ for some $k' \geq 1$ then, by Lemma 5.17, there are at least $\frac{|S|(k'-1)}{2}$ collisions for S and T_0. Hence if $\frac{|S|}{k'} = \frac{2^f}{k}$, then $k' = \frac{k|S|}{2^f}$ and thus if $|T_0(S)| \leq \frac{S}{k'} = \frac{2^f}{k}$, then there exist at least $\frac{|S|}{2}(\frac{k|S|}{2^f} - 1)$ collisions for S and T_0.

As in the previous proof we estimate the two probabilities obtained by the Law of Total Probability. Recall that the random variable d_S denotes the number of collisions and $\mathbf{E}(d_S) = \binom{|S|}{2} 2^{-f}$. The bound on the first probability, $\mathbf{Pr}\left(|T_0(S)| \leq \frac{2^f}{k}\right)$, is found again using the Markov's inequality,

$$\begin{aligned}
\mathbf{Pr}\left(|T_0(S)| \leq \frac{2^f}{k}\right) &\leq \mathbf{Pr}\left(d_S \geq \frac{|S|}{2}\left(\frac{k|S|}{2^f} - 1\right)\right) \\
&\leq \frac{|S|(|S|-1)}{2 \cdot 2^f \frac{|S|}{2}\left(\frac{k|S|}{2^f} - 1\right)} \\
&\leq \frac{|S|}{k|S| - 2^f} \\
&\leq \frac{|S|}{k|S| - \frac{2|S|}{2^l}} \\
&= \frac{2^l}{2^l k - 2}.
\end{aligned}$$

As in the previous statement we state the fact in a claim.

Claim 5.25.

$$\mathbf{Pr}\left(|T_0(S)| \leq \frac{2^f}{k}\right) \leq \frac{2^l}{2^l k - 2}.$$

The remaining case occurs when $|T_0(S)| > \frac{2^f}{k}$. Define μ as the inverse density of the set $T_0(S)$ in the factor space $\mathbb{Z}_2^f$. Then the assumption $|T_0(S)| > \frac{2^f}{k}$ implies

$$\mu = 1 - \frac{|T_0(S)|}{2^f} < 1 - \frac{1}{k} < 1.$$

From the choice of the dimension f and the assumption of Theorem 5.14

$$|S| \geq c_\epsilon b 2^b$$

it follows that

$$f = \left\lceil \log\left(\frac{|S|}{2^l}\right) \right\rceil \geq \log c_\epsilon + \log b + b - l.$$

From Theorem 5.14 used for the set $T_0(S)$, the source space $\mathbb{Z}_2^f$, the inverse density μ, the target space $\mathbb{Z}_2^b$ and the transformation T_1 we have

$$\begin{aligned}
\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{2^f}{k}\right) &\leq \mu^{f - b - \log b + \log\log\frac{1}{\mu}} \\
&\leq \mu^{\log c_\epsilon + \log b + b - l - b - \log b + \log\log\frac{1}{\mu}} \\
&\leq \left(1 - \frac{1}{k}\right)^{\log c_\epsilon - l + \log\log\left(\frac{1}{1-\frac{1}{k}}\right)}.
\end{aligned}$$

This result is worth remembering.

Claim 5.26.

$$\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{2^f}{k}\right) \leq \left(1 - \frac{1}{k}\right)^{\log c_\epsilon - l + \log\log\left(\frac{1}{1-\frac{1}{k}}\right)}.$$

Recall, we use the Law of Total Probability, Theorem B.13, to split the expression $\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right)$. To prove the statement, the probability of event $T(S) \neq \mathbb{Z}_2^b$ must be less than ϵ. Hence it suffices

$$\begin{aligned}
&\mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b\right) \\
&\quad = \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| \leq \frac{2^f}{k}\right) + \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{2^f}{k}\right) \\
&\quad \leq \mathbf{Pr}\left(|T_0(S)| \leq \frac{2^f}{k}\right) + \mathbf{Pr}\left(T(S) \neq \mathbb{Z}_2^b \wedge |T_0(S)| > \frac{2^f}{k}\right) \\
&\quad \leq \epsilon.
\end{aligned}$$

Put $\mu' = 1 - \frac{1}{k}$. From Claims 5.25 and 5.26 we obtain

$$\begin{aligned}
\frac{2^l}{2^l k - 2} + {\mu'}^{\log c_\epsilon + \log\log\left(\frac{1}{\mu'}\right) - l} &\leq \epsilon \\
\left(\epsilon - \frac{2^l}{2^l k - 2}\right) \cdot {\mu'}^{l - \log\log\left(\frac{1}{\mu'}\right)} &\geq {\mu'}^{\log c_\epsilon} \\
\frac{\log\left(\left(\epsilon - \frac{2^l}{2^l k - 2}\right) \cdot {\mu'}^{l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log \mu'} &\leq \log c_\epsilon.
\end{aligned}$$

Thus the value of the constant c_ϵ may be chosen as the minimal one satisfying for some $k, l \geq 1$ the inequality:

$$c_\epsilon \geq 2^{\frac{\log\left(\left(\epsilon - \frac{2^l}{2^l k - 2}\right) \cdot {\mu'}^{l - \log\log\left(\frac{1}{\mu'}\right)}\right)}{\log \mu'}}.$$

□

The following corollary shows an interesting value of c_ϵ achieved by the previous statement.

Corollary 5.27. *For $\epsilon = 0.8967$ the value of the constant c_ϵ may be chosen as* 17.31.

Proof. Use the previous statement for $k = 2.06$ and $l = 2$. Clearly,

$$\log c_\epsilon + \log b - l \geq 0.$$

□

5.5 Probability Distribution of the Variable lpsl

Theorems 5.14 and 5.15 and Corollaries 5.8, 5.9 and 5.16 give us enough power to achieve our first goal – asymptotic restriction of the expected length of the chain length. But first, we find the probability distribution of the variable **lpsl** for various situations depending on the size of the stored set.

In the original article [3] authors suppose hashing even super-linear amount of $m \log m$ elements into a table consisting of m slots. Common models use load factors lower than one and we suppose hashing of αm elements. When storing sets of size equal to αm for a bounded load factor α, the expected length can not grow, if compared to hashing $m \log m$ elements. Every stored set can be further extended into $m \log m$ elements and the estimate must still remain valid. Unfortunately, this argument does not bring a reasonable result.

Therefore we further generalise and refine our statements. We discover an important dependence of $\mathbf{E}(\mathbf{lpsl})$ on the table's load factor. We later state that $\mathbf{E}(\mathbf{lpsl})$ is proportional to the load factor α.

First let us summarise what is shown in the following pages. We use two basic ideas – factorisation and probability estimates of two highly correlated events. The two events E_1 and E_2 allow us to estimate the probability of the existence of a chain with length at least l, $l \in \mathbb{N}$.

Then we show Remark 5.36 that comes from the original work [3]. It is immediately improved and the results are later applied for hashing. The theorems we state are not expressed in the notation of hashing. Rather we say them in general terms of linear algebra. Then we use the theorems in the model proposed by us.

The notation of the section is the following. The source space, represents later the universe, is the vector space $\mathbb{Z}_2^u$. The target space, becomes a representation of the hash table, is denoted by $\mathbb{Z}_2^b$. As done many times before, we factor a random uniformly chosen linear transformation $T \in LT(\mathbb{Z}_2^u, \mathbb{Z}_2^b)$ through the factor space $\mathbb{Z}_2^f$. Model 5.4 shows the existence of two linear functions $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ and $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$. In addition the last one is surjective and they are chosen uniformly among $LT(\mathbb{Z}_2^u, \mathbb{Z}_2^f)$ and $LTS(\mathbb{Z}_2^f, \mathbb{Z}_2^b)$. Definitions 5.28 and 5.29, Remarks 5.31, 5.33 and Remark 5.36 stating the basic probability estimate and Theorem 5.38 giving the basic estimate of $\mathbf{E}(\mathbf{lpsl})$ are from [24], too.

Definition 5.28 (Event $E_1(S,T,l)$)**.** *Let $l \in \mathbb{N}$, $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a linear transformation and $S \subset \mathbb{Z}_2^u$.* Event $E_1(S,T,l)$ *occurs if there is a subset of S containing at least l elements mapped by the function T on a singleton,*

$$E_1(S,T,l) \equiv \exists \vec{y} \in \mathbb{Z}_2^b : |T^{-1}(\vec{y}) \cap S| > l.$$

The event E_1 corresponds to the existence of a chain of length at least l. The second event is defined to simplify the estimate of probability of the event E_1. It may seem quite unnatural but it fits the scheme of Theorem 5.14 as shown later.

Definition 5.29 (Event $E_2(S,T_0,T_1)$)**.** *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f, T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be linear transformations with T_1 being surjective and $S \subset \mathbb{Z}_2^u$.* Event $E_2(S,T_0,T_1)$ *occurs if*

$$E_2(S,T_0,T_1) \equiv \exists \vec{y} \in \mathbb{Z}_2^b : T_1^{-1}(\vec{y}) \subseteq T_0(S).$$

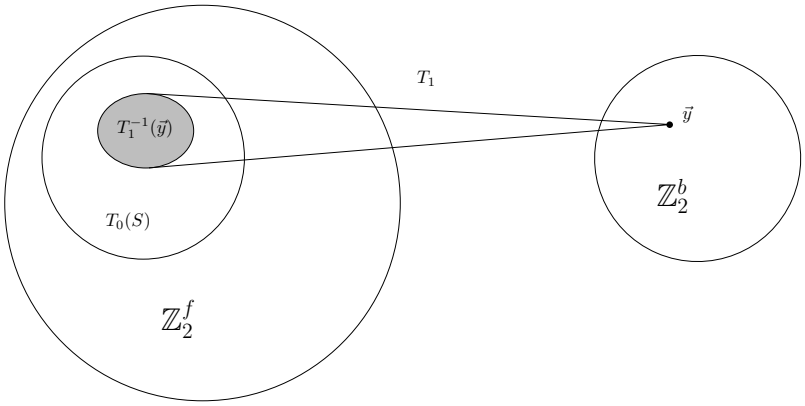


Figure 5.1: Occurrence of the event E_2.

Remember that when it is clear what we mean by S, T_0, T_1 and l we omit the parametrisation of the events and just use E_1 or E_2.

Now we will point an equivalent definition of the event E_2 showing why it may be used with Corollary 5.16.

Remark 5.30. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$, $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be linear transformations with T_1 being surjective and $S \subset \mathbb{Z}_2^u$. Then the event $E_2(S, T_0, T_1)$ occurs if and only if $T_1(\mathbb{Z}_2^f - T_0(S)) \neq \mathbb{Z}_2^b$. Equivalently*

$$(E_2(S, T_0, T_1) \equiv \exists \vec{y} : T_1^{-1}(\vec{y}) \subseteq T_0(S)) \Leftrightarrow (T_1(\mathbb{Z}_2^f - T_0(S)) \neq \mathbb{Z}_2^b).$$

Proof. To prove the direction from the left to the right assume that the event E_2 occurs. This happens if there is a vector $\vec{y} \in \mathbb{Z}_2^b$ such that $T_1^{-1}(\vec{y}) \subseteq T_0(S)$. Hence transformation T_1 can not map the set $\mathbb{Z}_2^f - T_0(S)$ onto the vector space $\mathbb{Z}_2^b$ since $\vec{y} \notin T_1(\mathbb{Z}_2^f - T^{-1}(\vec{y})) \supseteq T_1(\mathbb{Z}_2^f - T_0(S))$ or equivalently $\mathbb{Z}_2^b \neq T_1(\mathbb{Z}_2^f - T_0(S))$.

Now we show the reverse direction. If $\mathbb{Z}_2^b \neq T_1(\mathbb{Z}_2^f - T_0(S))$, then there is a vector $\vec{y} \in \mathbb{Z}_2^b$ such that $\vec{y} \notin T_1(\mathbb{Z}_2^f - T_0(S))$. Since T_1 is surjective we have that $T_1(\mathbb{Z}_2^f) = \mathbb{Z}_2^b$. Since there is no point in $\mathbb{Z}_2^f - T_0(S)$ mapped onto $\vec{y}$ it follows that the preimage of $\vec{y}$ must be contained in $T_0(S)$. Thus $T_1^{-1}(\vec{y}) \subseteq T_0(S)$.

□

Now, in the next remark, we use the previous equivalence to estimate the probability of the event E_2.

Remark 5.31. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$, $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be linear transformations with T_1 being surjective and $S \subset U$. Set $d = \frac{|\mathbb{Z}_2^f|}{b2^b}$. If $|S| \leq b2^b$ and $d > 1$, then*

$$\mathbf{Pr}\left(E_2(S, T_0, T_1))\right) \leq d^{-\log d - \log\log d}.$$

Proof. We apply Theorem 5.14 for the transformation T_1, the set $\mathbb{Z}_2^f - T_0(S)$, the target space $\mathbb{Z}_2^b$ and the inverse density $\mu = 1 - \frac{|\mathbb{Z}_2^f - T_0(S)|}{|\mathbb{Z}_2^f|}$ and we obtain

$$\mathbf{Pr}\left(T_1(\mathbb{Z}_2^f - T_0(S)) \neq \mathbb{Z}_2^b\right) \leq \mu^{f - b - \log b + \log\log\frac{1}{\mu}}.$$

We can estimate μ using only the value d as

$$\mu = 1 - \frac{|\mathbb{Z}_2^f - T_0(S)|}{|\mathbb{Z}_2^f|} = 1 - \frac{|\mathbb{Z}_2^f| - |T_0(S)|}{|\mathbb{Z}_2^f|} = \frac{|T_0(S)|}{|\mathbb{Z}_2^f|} \leq \frac{|S|}{|\mathbb{Z}_2^f|} \leq \frac{b2^b}{|\mathbb{Z}_2^f|} = \frac{1}{d} < 1.$$

To obtain the bound claimed by the remark rewrite the logarithm of the value d,

$$\log d = \log \frac{|\mathbb{Z}_2^f|}{b2^b} = \log |\mathbb{Z}_2^f| - \log(b2^b) = f - b - \log b.$$

In the following computation we use Corollary 5.8 to remove the inverse density μ. Since $E_2 \equiv T_1(\mathbb{Z}_2^f - T_0(S)) \neq \mathbb{Z}_2^b$ and $\mu \leq \frac{1}{d}$ we have the needed bound,

$$\begin{aligned}
\mathbf{Pr}\left(E_2\right) &\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)} \\
&= \mu^{\log d+\log\log\left(\frac{1}{\mu}\right)} \\
&\leq \left(\frac{1}{d}\right)^{\log d+\log\log d} \\
&= d^{-\log d-\log\log d}.
\end{aligned}$$

To meet the assumptions of Theorem 5.14, we must have that $\emptyset \neq \mathbb{Z}_2^f - T_0(S)$ and $\mathbb{Z}_2^f - T_0(S) \neq \mathbb{Z}_2^f$. Since the $S \neq \emptyset$, it certainly holds that $\mathbb{Z}_2^f - T_0(S) \neq \mathbb{Z}_2^f$. Because $d = \frac{|\mathbb{Z}_2^f|}{|S|} > 1$, it follows that $|\mathbb{Z}_2^f| > |S| \geq |T_0(S)|$ and the set $\mathbb{Z}_2^f - T_0(S)$ then can not be empty.

□

We show similar remarks fitting the situation when the size of the set S is proportional to that of the target space $\mathbb{Z}_2^b$. This corresponds to the situation when hashing only αm elements.

Statement 5.32. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$, $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be linear transformations with T_1 being surjective and $S \subset U$. Let $\alpha \in \mathbb{R}$, $\alpha > 0$ and assume that $|S| = \alpha 2^b$.*

1. *If $d = \frac{2^f}{\alpha b 2^b}$ and $d > 1$, then $\mathbf{Pr}\left(E_2(S, T_0, T_1))\right) \leq d^{-\log\alpha-\log d-\log\log d}$.*
2. *If $d = \frac{2^f}{\alpha 2^b}$ and $d > 1$, then $\mathbf{Pr}\left(E_2(S, T_0, T_1))\right) \leq d^{\log b-\log\alpha-\log d-\log\log d}$.*

Proof. We slightly changed the choice of the variable d and naturally moved to a different size $|S|$. The proof itself remains almost the same.

First, we show $\mu \leq \frac{1}{d} < 1$ in either case. Observe that

$$\mu = 1 - \frac{|\mathbb{Z}_2^f - T_0(S)|}{|\mathbb{Z}_2^f|} = \frac{|T_0(S)|}{|\mathbb{Z}_2^f|} \leq \frac{|S|}{|\mathbb{Z}_2^f|} = \frac{\alpha 2^b}{2^f}.$$

In the first case, $d = \frac{2^f}{\alpha b 2^b}$, we have

$$\mu \leq \frac{\alpha 2^b}{2^f} \leq \frac{\alpha b 2^b}{2^f} = \frac{1}{d} < 1.$$

In the second case, $d = \frac{2^f}{\alpha 2^b}$:

$$\mu \leq \frac{\alpha 2^b}{2^f} = \frac{1}{d} < 1.$$

The wanted estimate thus holds.

Secondly, we find the $\log d$ in either case. In the first one, $d = \frac{2^f}{\alpha b 2^b}$:

$$\log d = f - \log \alpha - b - \log b.$$

In the second one, $d = \frac{2^f}{\alpha 2^b}$, we have

$$\log d = f - \log \alpha - b.$$

From $\mu < 1$ and $|S| = \alpha 2^b > 0$ we have the assumption $\emptyset \neq \mathbb{Z}_2^f - T_0(S) \neq \mathbb{Z}_2^f$ of Theorem 5.14 met. Now we use Theorem 5.14 for the transformation T_1, the set $\mathbb{Z}_2^f - T_0(S)$ and the inverse density μ and obtain

$$\mathbf{Pr}\left(E_2\right) \leq \mu^{f-b-\log b-\log\log\left(\frac{1}{\mu}\right)}.$$

To prove the statement, we continue similarly as in the proof of Remark 5.31. The next estimates follow from Corollary 5.8 and the fact that $\mu \leq \frac{1}{d}$. In the first case we have that

$$\begin{aligned}
\mathbf{Pr}\left(E_2\right) &\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)} \\
&= \mu^{\log d+\log\alpha+\log\log\left(\frac{1}{\mu}\right)} \\
&\leq \left(\frac{1}{d}\right)^{\log d+\log\alpha+\log\log d} \\
&= d^{-\log\alpha-\log d-\log\log d}.
\end{aligned}$$

In the second case we have that

$$\begin{aligned}
\mathbf{Pr}\left(E_2\right) &\leq \mu^{f-b-\log b+\log\log\left(\frac{1}{\mu}\right)} \\
&= \mu^{\log d+\log\alpha-\log b+\log\log\left(\frac{1}{\mu}\right)} \\
&\leq \left(\frac{1}{d}\right)^{\log d+\log\alpha-\log b+\log\log d} \\
&= d^{\log b-\log\alpha-\log d-\log\log d}.
\end{aligned}$$

□

A similar remark for an estimate of the conditional probability of the event $E_2 \mid E_1$ now follows.

Remark 5.33. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ and $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be random uniformly chosen linear transformations with T_1 being surjective and $T = T_1 \circ T_0$. Let c_ϵ be a constant for which Theorem 5.15 is satisfied, $S \subset \mathbb{Z}_2^u$, $\epsilon \in (0,1)$ and $l \in \mathbb{N}$, $l \geq c_\epsilon(f-b)2^{f-b}$. Then*

$$\mathbf{Pr}\left(E_2(S,T_0,T_1) \mid E_1(S,T,l)\right) \geq 1-\epsilon.$$

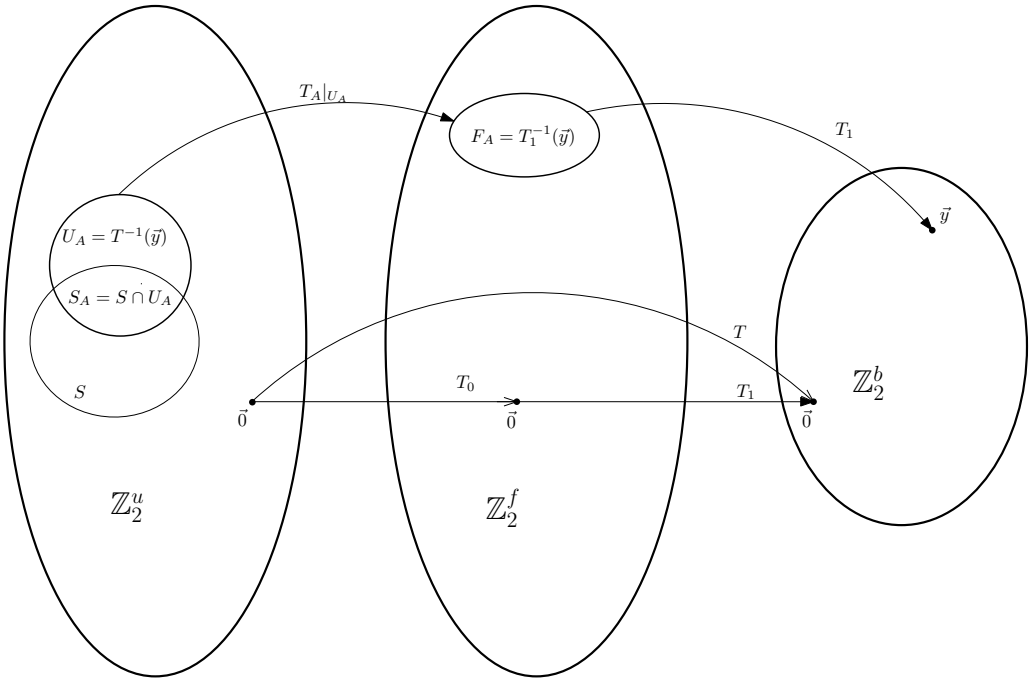


Figure 5.2: Image depicting the situation in the proof.

Proof. According to Model 5.4 we have that the linear transformation $T = T_1 \circ T_0$ is chosen uniformly.

Assume that the event E_1 occurs, then there is a vector $\vec{y} \in \mathbb{Z}_2^b : |T^{-1}(\vec{y}) \cap S| > l$. We fix the vector $\vec{y}$ and let $U_A = T^{-1}(\vec{y})$ and $F_A = T_1^{-1}(\vec{y})$. So there is a maximal subset $S_A \subseteq S$, $|S_A| > l$ mapped to $\vec{y}$. Notice that $S_A = U_A \cap S$. According to Lemma A.10 the sets U_A and F_A are affine vector subspaces and $|F_A| = 2^{f-b}$.

Since T_0 is a random uniformly chosen linear mapping from Model 5.5 it follows that its restriction $T_0|_{U_A}$ is also a random uniformly chosen affine linear transformation.

Now we use Corollary 5.16 for the source space U_A, the set $S_A \subseteq U_A$, the target space F_A and mapping $T_0|_{U_A}$ and obtain that

$$\mathbf{Pr}\left(T_0|_{U_A}(S_A) = F_A \mid E_1\right) \geq 1 - \epsilon.$$

If $F_A = T_1^{-1}(\vec{y}) \subseteq T_0(S)$, then the event E_2 certainly occurs. Stated in the language of probability:

$$\mathbf{Pr}\left(F_A \subseteq T_0(S) \mid E_1\right) \leq \mathbf{Pr}\left(E_2 \mid E_1\right).$$

From $F_A = T_0|_{U_A}(S) \subseteq T_0(S)$ it follows that:

$$\mathbf{Pr}\left(E_2 \mid E_1\right) \geq \mathbf{Pr}\left(F_A \subseteq T_0(S) \mid E_1\right) \geq \mathbf{Pr}\left(F_A = T_0|_{U_A}(S_A) \mid E_1\right) \geq 1 - \epsilon.$$

Thus the proof of Remark 5.33 finishes.

□

The next corollary puts Remark 5.33 and Lemma B.10 together.

Corollary 5.34. *Let $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ and $T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformations with T_1 being surjective. Let $\epsilon \in (0,1)$, $S \subset \mathbb{Z}_2^u$ and $l \in \mathbb{N}$, $l \geq c_\epsilon(f-b)2^{f-b}$. Then*

$$\mathbf{Pr}\left(E_1(S,T,l)\right) \leq \frac{1}{1-\epsilon}\mathbf{Pr}\left(E_2(S,T_0,T_1)\right).$$

Proof. The proof is a straightforward use of Remark 5.33 and Lemma B.10. They imply that

$$\mathbf{Pr}\left(E_1\right) \leq \frac{\mathbf{Pr}\left(E_2\right)}{\mathbf{Pr}\left(E_2 \mid E_1\right)} \leq \frac{1}{1-\epsilon}\mathbf{Pr}\left(E_2\right).$$

□

Definition 4.3 of the variable **lpsl** comes from the area of hashing and uses the notation of chains and their lengths. However we can refer to it now, too. Assume that the universe is equal to the vector space $\mathbb{Z}_2^u$ and the hash table is represented by the vector space $\mathbb{Z}_2^b$ and $S \subset U$. The randomness is brought by the uniform choice of a linear transformation $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ among $LT(\mathbb{Z}_2^u, \mathbb{Z}_2^b)$. Realise that the random variable $\mathbf{psl}(\vec{b}) = |T^{-1}(\vec{b}) \cap S|$ for a vector $\vec{b} \in \mathbb{Z}_2^b$. By setting $\mathbf{lpsl} = \max_{\vec{b} \in \mathbb{Z}_2^b} \mathbf{psl}(\vec{b})$ their meanings remain the same.

Lemma 5.35. *If $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ is a random uniformly chosen linear transformation, $S \subset \mathbb{Z}_2^u$ and $l \in \mathbb{N}$, then $E_1(S,T,l) \Leftrightarrow \mathbf{lpsl} > l$.*

Proof. The event $E_1(S,T,l)$ denotes the existence of a vector $\vec{y} \in \mathbb{Z}_2^b$ such that $|T^{-1}(\vec{y}) \cap S| > l$. Observe that such vector $\vec{y}$ exists if and only if the variable $\mathbf{lpsl} > l$ because

$$(\exists \vec{y} \in \mathbb{Z}_2^b : |T^{-1}(\vec{y}) \cap S| > l) \Leftrightarrow (\exists \vec{y} \in \mathbb{Z}_2^b : \mathbf{psl}(\vec{y}) > l) \Leftrightarrow (\mathbf{lpsl} > l).$$

□

We are able to bound the probability density function of the random variable **lpsl** when referring to its above definition.

Remark 5.36. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear map, $\epsilon \in (0,1)$ and $S \subset \mathbb{Z}_2^u$, $|S| \leq b2^b$. Then for every $r \geq 4$*

$$\mathbf{Pr}\left(\mathbf{lpsl} > 4c_\epsilon r b \log b\right) \leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right)-\log\log\left(\frac{r}{\log r}\right)}.$$

Proof. In the proof we conveniently use Corollary 5.34, Remark 5.31 and Lemma 5.35. We only have to choose their parameters. First we create a factor space $\mathbb{Z}_2^f$, its dimension is specified later. From Model 5.5 and the random uniform and independent selection of two linear transformations $T_0 : \mathbb{Z}_2^u \to \mathbb{Z}_2^f$ and surjective

$T_1 : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ it follows that the linear mapping $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ such that $T = T_1 \circ T_0$ is chosen uniformly as well. Fix the mappings T_0, T_1 and T.

Now set

$$\begin{aligned} f &= \lfloor b + \log b + \log r - \log\log r + 1 \rfloor, \\ l &= 4c_\epsilon r b \log b, \\ d &= \frac{2^f}{b2^b} \geq \frac{b2^b}{b2^b} \cdot \frac{r}{\log r} = \frac{r}{\log r} \geq 2. \end{aligned}$$

The choice of f implies that $|\mathbb{Z}_2^f| \geq \mathbb{Z}_2^b|$ because

$$f \geq b + \log b + \log r - \log\log r \geq b.$$

Hence a surjective function T_1 exists and may be fixed. Notice that the choices meet all the assumptions, $d > 1$, of Remark 5.31. To verify the condition $l \geq c_\epsilon (f-b) 2^{f-b}$ of Remark 5.36 we first show that

$$2^{f-b} \leq 2^{b+\log b+\log r-\log\log r+1-b} = \frac{2rb}{\log r}.$$

From the fact, $f - b \leq \log\left(\frac{2rb}{\log r}\right) \leq \log b \log r$, we have that the assumption holds since

$$\begin{aligned} c_\epsilon (f-b) 2^{f-b} &\leq c_\epsilon \frac{2rb}{\log r} \log\left(\frac{2rb}{\log r}\right) \\ &\leq 2c_\epsilon b \frac{r}{\log r} \log b \log r \\ &\leq 4c_\epsilon r b \log b \\ &= l. \end{aligned}$$

From Corollary 5.34, Remark 5.31 and Corollary 5.9 it follows that

$$\begin{aligned} \mathbf{Pr}\left(E_1\right) &\leq \frac{1}{1-\epsilon} \mathbf{Pr}\left(E_2\right) \\ &\leq \frac{1}{1-\epsilon} d^{-\log d - \log\log d} \\ &\leq \frac{1}{1-\epsilon} \left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}. \end{aligned}$$

According to Lemma 5.35 the event E_1 occurs if and only if $\mathbf{lpsl} > l$. The proof is completed by writing down the facts obtained so far

$$\begin{aligned} \mathbf{Pr}\left(\mathbf{lpsl} > l\right) &= \mathbf{Pr}\left(\mathbf{lpsl} > 4c_\epsilon r b \log b\right) \\ &= \mathbf{Pr}\left(E_1(S, T, 4c_\epsilon r b \log b)\right) \\ &\leq \frac{1}{1-\epsilon} \left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}. \end{aligned}$$

□

We modify the previous remark for the set S having $|S| = \alpha 2^b$.

Remark 5.37. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear mapping, $\epsilon \in (0,1)$, $r \geq 4$ and $\alpha \in (0.5, \frac{\log r}{2})$. If S is a subset of $\mathbb{Z}_2^u$ such that $|S| = \alpha 2^b$, then*

$$\mathbf{Pr}\left(\mathbf{lpsl} > 4c_\epsilon \alpha r b \log b\right) \leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)} \text{ and}$$

$$\mathbf{Pr}\left(\mathbf{lpsl} > 2\alpha c_\epsilon r\right) \leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}.$$

Proof. We do not repeat the whole proof of Remark 5.36 since the approach remains the same. Like in the previous proof, fix the linear mappings T_0, surjective T_1 with $T = T_1 \circ T_0$ and let $\mathbb{Z}_2^f$ be the factor space. We just show the choices that prove the remark. The choices for the first claim are indexed with 1 and in the second case we use the index 2. When we refer to a chosen variable without an index, then the statement, in which it appears, must be valid for either choice. Now perform the choices by setting

$$\begin{aligned}
f_1 &= \lfloor b + \log b + \log r - \log\log r + \log\alpha + 1 \rfloor, \\
f_2 &= \lfloor b + \log r - \log\log r + \log\alpha + 1 \rfloor, \\
d_1 &= \frac{2^{f_1}}{\alpha b 2^b}, \\
d_2 &= \frac{2^{f_2}}{\alpha 2^b}, \\
l_1 &= 4c_\epsilon \alpha r b \log b, \\
l_2 &= 2\alpha c_\epsilon r.
\end{aligned}$$

So that T_1 exists, we have to verify that $|\mathbb{Z}_2^f| \geq |\mathbb{Z}_2^b|$. It is enough to have $f \geq b$. The inequality $f \geq b$ follows from the fact $\log r - \log\log r + \log\alpha \geq 0$ and from our choice of f since

$$\begin{aligned}
f_1 &\geq b + \log b + \log r - \log\log r + \log\alpha \geq b, \\
f_2 &\geq b + \log r - \log\log r + \log\alpha \geq b.
\end{aligned}$$

Secondly, we show that the assumptions of Statement 5.32 are met for both choices of d. Precisely we need $d > 1$ and this is satisfied because

$$\begin{aligned}
d_1 &= \frac{2^{f_1}}{\alpha b 2^b} \geq \frac{r}{\log r} \geq 2, \\
d_2 &= \frac{2^{f_2}}{\alpha 2^b} \geq \frac{r}{\log r} \geq 2.
\end{aligned}$$

Naturally, we want to meet the condition placed on the variable l of Corollary 5.34. Recall the assumption $\alpha < \frac{\log r}{2}$, which is equivalent to $\log\frac{2\alpha}{\log r} < 0$, that is used in either case. Let us discuss the first case:

$$f_1 - b \leq \log b + \log r - \log\log r + \log\alpha + 1 \leq \log b + \log r \leq \log b \log r.$$

Thus for the value of the variable l_1 we have that

$$\begin{aligned} c_\epsilon(f_1 - b)2^{f_1 - b} &< c_\epsilon \left(\frac{2\alpha br}{\log r}\right)(\log b \log r) \\ &\leq 4c_\epsilon \alpha r b \log b \\ &= l_1. \end{aligned}$$

The validity in the second case follows from

$$\begin{aligned} c_\epsilon(f_2 - b)2^{f_2 - b} &\leq c_\epsilon \left(\frac{2\alpha r}{\log r}\right) \log \left(\frac{2\alpha r}{\log r}\right) \\ &\leq 2c_\epsilon \alpha \frac{r}{\log r} \log r \\ &= 2c_\epsilon \alpha r = l_2. \end{aligned}$$

Thus we are able to refer to Corollary 5.34. Now use Lemma 5.35, Corollary 5.34, Remark 5.31 and Corollary 5.9 in either case. In the first one we have

$$\begin{aligned} \mathbf{Pr}\left(\mathbf{lpsl} > 4c_\epsilon \alpha r b \log b\right) &= \mathbf{Pr}\left(\mathbf{lpsl} > l_1\right) \\ &= \mathbf{Pr}\left(E_1(S, T, l_1)\right) \\ &\leq \frac{1}{1-\epsilon}\mathbf{Pr}\left(E_2\right) \\ &\leq \frac{1}{1-\epsilon} d^{-\log\alpha - \log d - \log\log d} \\ &= \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}. \end{aligned}$$

And in the second case we have

$$\begin{aligned} \mathbf{Pr}\left(\mathbf{lpsl} > 2\alpha c_\epsilon r\right) &= \mathbf{Pr}\left(\mathbf{lpsl} > l_2\right) \\ &= \mathbf{Pr}\left(E_1(S, T, l_2)\right) \\ &\leq \frac{1}{1-\epsilon}\mathbf{Pr}\left(E_2\right) \\ &\leq \frac{1}{1-\epsilon} d^{\log b - \log\alpha - \log d - \log\log d} \\ &\leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}. \end{aligned}$$

□

5.6 The Expected Value of the Variable lpsl

In this section we estimate the expected value of the variable **lpsl**, $\mathbf{E}\left(\mathbf{lpsl}\right)$. We exploit bounds from Section 5.5 which give its cumulative probability density function. We are interested in this expected value because it corresponds to the length of the longest chain when using the system of linear transformations as a universal class.

Theorem 5.38. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformation and $\epsilon \in (0,1)$. If S is a subset of the vector space $\mathbb{Z}_2^u$ such that $|S| \leq b2^b$, then*

$$\mathbf{E}(\mathbf{lpsl}) \leq 4c_\epsilon(4 + I_\epsilon)b \log b.$$

The value I_ϵ is defined as:

$$I_\epsilon = \int_4^\infty \frac{1}{1-\epsilon} \left(\frac{r}{\log r} \right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)} dr. \tag{5.39}$$

Proof. First set $K_\epsilon = 4c_\epsilon b \log b$. In order to prove the theorem we use the probability distribution of the variable **lpsl** from Remark 5.36. For every $r \geq 4$ it states that

$$\mathbf{Pr}(\mathbf{lpsl} \geq rK_\epsilon) \leq \frac{1}{1-\epsilon} \left(\frac{1}{\log r} \right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}.$$

Lemma B.24 implies

$$\begin{aligned}
\mathbf{E}(\mathbf{lpsl}) &= \int_0^\infty \mathbf{Pr}(\mathbf{lpsl} > l)\, dl \\
&\leq 4K_\epsilon + \int_{4K_\epsilon}^\infty \mathbf{Pr}(\mathbf{lpsl} > l)\, dl \\
&= 4K_\epsilon + K_\epsilon \int_4^\infty \mathbf{Pr}(\mathbf{lpsl} > rK_\epsilon)\, dr \\
&\leq 4K_\epsilon + K_\epsilon \int_4^\infty \frac{1}{1-\epsilon} \left(\frac{r}{\log r} \right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)} dr \\
&= K_\epsilon(4 + I_\epsilon) \in O(K_\epsilon) = O(b \log b).
\end{aligned}$$

The fact that the integral I_ϵ is convergent for every $\epsilon \in (0,1)$ is shown later in Lemma 5.45. □

Multiplicative constant $4c_\epsilon(4 + I_\epsilon)$ plays an important role for a practical use of the result. For example when choosing ϵ equal to $\frac{1}{2}$ the value of the constant c_ϵ equals 4^{17} when using the original estimate. Our next goal is to show a better estimate when assuming that $\alpha \leq 1$ and using our results.

Theorem 5.40. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformation, $\epsilon \in (0,1)$ and $\alpha \in [0.5, 1]$. If S is a subset of the vector space $\mathbb{Z}_2^u$ such that $|S| = \alpha 2^b$, then*

$$\mathbf{E}(\mathbf{lpsl}) \leq 4\alpha c_\epsilon(4 + I_\epsilon)b \log b.$$

The value I_ϵ is defined in Equality 5.39.

Proof. Set $K_\epsilon = 4c_\epsilon b \log b$. When estimating $\mathbf{Pr}(\mathbf{lpsl} > r\alpha K_\epsilon)$ we use Remark 5.37. Its assumptions are satisfied since we use it for $r \geq 4$ and $\alpha \leq 1 \leq \frac{\log r}{2}$. It states that

$$\mathbf{Pr}(\mathbf{lpsl} > r\alpha K_\epsilon) \leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}.$$

Since $\alpha \leq 1$, then $\log\alpha \leq 0$ and thus we conclude that

$$\mathbf{Pr}(\mathbf{lpsl} > r\alpha K_\epsilon) \leq \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}.$$

Now we compute the expected value using Lemma B.24 as

$$\begin{aligned}
\mathbf{E}(\mathbf{lpsl}) &= \int_0^\infty \mathbf{Pr}(\mathbf{lpsl} > l)\, dl \\
&\leq 4\alpha K_\epsilon + \int_{4\alpha K_\epsilon}^\infty \mathbf{Pr}(\mathbf{lpsl} > l)\, dl \\
&= 4\alpha K_\epsilon + \alpha K_\epsilon \int_4^\infty \mathbf{Pr}(\mathbf{lpsl} > r\alpha K_\epsilon)\, dr \\
&\leq 4\alpha K_\epsilon + \alpha K_\epsilon \int_4^\infty \frac{1}{1-\epsilon}\left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)} dr \\
&= \alpha K_\epsilon(4 + I_\epsilon) \in O(\alpha K_\epsilon) = O(\alpha b \log b).
\end{aligned}$$

□

5.7 The Achieved Bound

The improved estimate of c_ϵ combined with the last claim of Remark 5.37 are used together to show a tighter bound on $\mathbf{E}(\mathbf{lpsl})$.

Theorem 5.41. *Let $b \in \mathbb{N}$, $b \geq 4$, $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformation, $\epsilon \in (0,1)$ and $\alpha \in [0.5, 1]$. If S is a subset of the vector space $\mathbb{Z}_2^u$ such that $|S| = \alpha 2^b$, then*

$$\mathbf{E}(\mathbf{lpsl}) \leq \frac{4c_\epsilon\alpha}{1-\epsilon} b\log b + 2c_\epsilon\alpha\left(\frac{J_{\alpha,b} - 4}{1-\epsilon} + 4\right).$$

The value $J_{\alpha,b}$ is defined as

$$J_{\alpha,b} = \int_{2b\log b}^\infty \left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)} dr. \tag{5.42}$$

Proof. From Remark 5.37, Lemma B.24 and Lemma 5.46 it follows that

$$\begin{aligned}
\mathbf{E}\left(\mathbf{lpsl}\right) &= \int_0^\infty \mathbf{Pr}\left(\mathbf{lpsl} > l\right) dl \\
&\leq 8c_\epsilon\alpha + \int_{8c_\epsilon\alpha}^\infty \mathbf{Pr}\left(\mathbf{lpsl} > l\right) dl \\
&= 8c_\epsilon\alpha + 2c_\epsilon\alpha \int_4^\infty \mathbf{Pr}\left(\mathbf{lpsl} > 2c_\epsilon\alpha r\right) dr \\
&= 2c_\epsilon\alpha \left(4 + \frac{1}{1-\epsilon}\int_4^\infty \min\left(1, \left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}\right) dr\right) \\
&\leq 2c_\epsilon\alpha \left(4 + \frac{1}{1-\epsilon}\left(2b\log b - 4 + J_{\alpha,b}\right)\right).
\end{aligned}$$

Let the value $J'_{\alpha,b}$ denote the integral,

$$J'_{\alpha,b} = \int_4^\infty \min\left(1, \left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}\right) dr. \tag{5.43}$$

Later, in Lemma 5.46 we estimate the integral $J'_{\alpha,b}$ by $2b\log b - 4 + J_{\alpha,b}$, where The whole bound on the expected value, $\mathbf{E}\left(\mathbf{lpsl}\right)$, then looks like:

$$\mathbf{E}\left(\mathbf{lpsl}\right) \leq \frac{4c_\epsilon\alpha}{1-\epsilon} b\log b + 2c_\epsilon\alpha\left(\frac{J_{\alpha,b}-4}{1-\epsilon} + 4\right).$$

In advance, let us note that from Lemma 5.46 it follows that $J_{\alpha,b} \leq 3.36$. The bound thus exists for every $\alpha \in [0.5, 1]$. □

Corollary 5.44. *Let $b \in \mathbb{N}$, $b \geq 4$, $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformation and $\alpha \in [0.5, 1]$. If S is a subset of the vector space $\mathbb{Z}_2^u$ such that $|S| = \alpha 2^b$, then*

$$\mathbf{E}\left(\mathbf{lpsl}\right) \leq 538\alpha b\log b + 44.$$

Proof. The best estimate on the expected value $\mathbf{E}\left(\mathbf{lpsl}\right)$ is achieved by the parametrisation technique using the estimate of c_ϵ from Statement 5.23. For the choice of $\epsilon = 0.8$ and by settings parameters $k = 2.26$, $l = 2$ of Statement 5.23 we get the required bound. □

5.8 Minimising the Integrals

This integral I_ϵ is a part of the multiplicative constant $4c_\epsilon(4 + I_\epsilon)$. The next lemma states a way how to estimate it and compute the value of the multiplicative constant so that it is minimal.

In the original proof of Theorem 5.38 in [3] only the special case $I_{\frac{1}{2}}$ is considered. We moved to I_ϵ because choosing other values for $\epsilon \in (0,1)$ may bring an improvement for the value of the multiplicative constant. Indeed, if we select different value of ϵ and use better estimates for the constant c_ϵ, then we really obtain a smaller value of the multiplicative constant and a tighter bound on $\mathbf{E}\,(\mathbf{lpsl})$.

Lemma 5.45. *The integral I_ϵ from Equality 5.39 converges for every $\epsilon \in (0,1)$ and*

$$I_\epsilon \leq \frac{4.8}{1-\epsilon}.$$

Proof. We recall that the integral I_ϵ equals

$$I_\epsilon = \frac{1}{1-\epsilon} \int_4^\infty \left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right)-\log\log\left(\frac{r}{\log r}\right)}.$$

Evaluation of the integral I_ϵ is split into two parts. We compare the integrand of I_ϵ to a function $f : \mathbb{R} \to \mathbb{R}$ which has a convergent improper integral $\int_4^\infty f(x)dx$. The function f chosen here is $x^{-1.5}$.

The function f majors the integrand only for $r \geq 16$. Therefore we bound the value of the integral I_ϵ in the interval $[4,16]$ by its upper Riemann sum.

For $r = 16$, the function f equals $f(r) = r^{-1.5} = \frac{1}{64}$. The value of the integrand equals

$$\left(\frac{16}{\log 16}\right)^{-\log\left(\frac{16}{\log 16}\right)-\log\log\left(\frac{16}{\log 16}\right)} = 4^{-2-1} = \frac{1}{64}.$$

By combining our estimates we obtain that

$$\begin{aligned} I_\epsilon &\leq \frac{1}{1-\epsilon}\left(\sum_{r=4}^{15}\left(\frac{r}{\log r}\right)^{-\log\left(\frac{r}{\log r}\right)-\log\log\left(\frac{r}{\log r}\right)} + \int_{16}^\infty \frac{1}{r^{1.5}}dr\right) \\ &\leq \frac{1}{1-\epsilon}\left(4.3 + \frac{1}{2}\right) = \frac{4.8}{1-\epsilon}. \end{aligned}$$

The estimate of the Riemann sum can be computed numerically. □

Lemma 5.46. *Let $b \in \mathbb{N}$, $b \geq 4$, $\alpha \in [0.5,1]$, $J'_{\alpha,b}$ be the integral defined in Equality 5.43 and $J_{\alpha,b}$ be the integral defined in Equality 5.42. Then*

$$J'_{\alpha,b} \leq 2b\log b - 4 + J_{\alpha,b} \text{ and}$$

$$J_{\alpha,b} \leq 3.36.$$

Proof. We estimate the integral

$$J'_{\alpha,b} = \int_4^\infty \min\left(1, \left(\frac{r}{\log r}\right)^{\log b - \log\alpha - \log\left(\frac{r}{\log r}\right) - \log\log\left(\frac{r}{\log r}\right)}\right) dr$$

in a similar way to the previous one.

However, the situation is slightly complicated. First we must realise that if the exponent is negative, the integrand is certainly less than one. We show that if $r > 2b\log b$, then the exponent is negative.

First we show that $r > 2b\log b$ implies $\log\left(\frac{r}{\log r}\right) \geq \log b$ since

$$\begin{aligned}\frac{r}{\log r} &\geq \frac{2b\log b}{1+\log b+\log\log b}\\ &= \frac{2b}{1+\frac{1}{\log b}+\frac{\log\log b}{\log b}}\\ &\geq \frac{2b}{2} = b.\end{aligned}$$

Since we assume $b \geq 4$, we have that $r > 2b\log b \geq 16$. So if $r > 2b\log b$, then for the remaining part of the exponent we have that

$$-\log\alpha - \log\log\left(\frac{r}{\log r}\right) < -\log 0.5 - \log\log\frac{16}{4} = 0.$$

Indeed, if $r > 2b\log b$, then the exponent is negative.

Now the integral $J'_{\alpha,b}$ is split into two parts according to the value of the variable r. If $r \in [4, 2b\log b]$, then we estimate the integrand by one. In the interval $[2b\log b, \infty)$ we use the value

$$J_{\alpha,b} = \int\limits_{2b\log b}^{\infty} \left(\frac{r}{\log r}\right)^{\log b-\log\alpha-\log\left(\frac{r}{\log r}\right)-\log\log\left(\frac{r}{\log r}\right)} dr.$$

So

$$J'_{\alpha,b} \leq 2b\log b - 4 + J_{\alpha,b}.$$

The integral $J_{\alpha,b}$ is determined in the similar way as the value of the integral I_ϵ in Lemma 5.45. The integrand is majored by the function $x^{-1.3}$ for $x \geq 2048$. In the interval $[16, 2048]$ we estimate the integral by its upper Riemann sum. We partition the interval uniformly with the norm equal to 0.1. For simplicity put $g(r) = \frac{r}{\log r}$ and estimate the integral $J_{\alpha,b}$ as

$$\begin{aligned}J_{\alpha,b} &= \int_{2b\log b}^{\infty} g(r)^{\log b-\log\alpha-\log g(r)-\log\log g(r)} dr\\ &\leq \int\limits_{16}^{\infty} g(r)^{1-\log\log g(r)} dr\\ &\leq \sum_{r\in\{16,16.1,\ldots,2048\}} \frac{g(r)^{1-\log\log g(r)}}{10} + \int\limits_{2048}^{\infty} r^{-1.3} dr\\ &\leq 3.01 + \frac{2048^{-0.3}}{0.3} \leq 3.36.\end{aligned}$$

□

Chapter 6

The Model of Universal Hashing

The hashing scheme we propose later in this chapter is a solution to the set representation problem. Solution of this problems usually provide some basic operations such as *Insert*, *Delete* and *Find*. These operations allow querying if an element is stored within the represented set, accessing the stored elements and inserting an element. Some schemes, e.g. double hashing, do not implement the element removal at all or the efficient implementation is not known.

The must important are the operations' running times. Various solutions to many algorithmic problems prefer different operations when representing sets. For instance some applications query the stored data rarely, e.g. log files. On the other hand, other applications of the set representation problem may store the data that is almost never changed – find operation is preferred. Therefore the running times of only selected operations are considered crucial.

In the case of a simple array we have $O(1)$ time for the find procedure provided that we know an element's index. But insertion or deletion may take $O(n)$ time. Better bounds for dynamic arrays can be obtained using the amortised complexity. Another example are balanced trees, they have running times typically bounded by the logarithmic time. As already mentioned, the right answer which data structure should be used lies in the estimated structure of the operations. The right choice may be an asymptotic improvement. Anyway, this does not change the fact that short running times are appreciated.

In this chapter we analyse the running times of the universal hashing. We start by mentioning the known facts. Then, we analyse the universal hashing using the system of linear transformations over vector spaces. Finally, we propose a model that guarantees the worst case complexity of the find operation.

6.1 Time Complexity of Universal Hashing

In this section we assume that the system of hash functions, which we use, is at least c-universal. The running time of the find operation is certainly proportional to the length of the chain of an element's bucket. The obvious worst case time is $O(n)$ where n is the number of elements hashed. The universal hashing gives a far better

expected estimate, $O(1)$.

Recall the definitions and notation from Chapter 2. The value n denotes the size of the represented set and m is the size of the table. The load factor of a hash table is denoted by $\alpha = \frac{n}{m}$.

Theorem 6.1. *Assume that we store a set S in a hash table using a c-universal class of functions H. Let the table's load factor α be upper bounded. Then the expected length of a chain is lower or equals $c\alpha$.*

Proof. We find the expected length of a chain containing arbitrary element $x \in U$. The expectation is taken over the uniform choice of a function from the universal family of functions. From the definition of the expected value we have that

$$\begin{aligned}
\mathbf{E}(\mathbf{psl}) &= \frac{\sum_{h \in H} psl(h(x), h)}{|H|} \\
&= \frac{\sum_{h \in H} \sum_{y \in S} I(h(x) = h(y))}{|H|} \\
&= \frac{\sum_{y \in S} \sum_{h \in H} I(h(x) = h(y))}{|H|} \\
&= \sum_{y \in S} \mathbf{Pr}(h(x) = h(y)) \\
&\leq \frac{cn}{m} = c\alpha.
\end{aligned}$$

□

Corollary 6.2. *If we hash using a c-universal class and α denotes the table's load factor, then the expected time of Find operation is $1 + c\alpha$.*

Proof. The running time of the find operation is proportional to the length of the chain in which the searched element belongs. In the worst case Find operation iterates the whole chain. We also add time for retrieving the element's hash value and for checking if the chain is not empty. These operations are usually performed in a constant time. From Theorem 6.1 it follows that the expected length of a chain is $c\alpha$. Since we have no assumptions on the distribution of the input, the expected running time is then bounded by $1 + c\alpha$. □

Corollary 6.3. *The expected time of Find operation in every model of universal hashing is $O(1)$ when the load factor is bounded.*

Proof. Follows from Corollary 6.2. □

6.2 Consequences of Trimming Long Chains

The model of hashing we propose guarantees the worst case bound on the length of the longest chain. Hence it bounds the running times of the operations. Knowledge

$\mathbf{E}(\mathbf{lpsl})$ enables us to state a bound on the length of a chain. If a chain, whose length is greater than our bound, is found, then we choose another function in our c-universal system and we rehash the represented set using the new function. This *bound* corresponds to the probability of the existence of a long chain. In fact, we set the *bound* according to this probability. Now let us examine consequences of such *limits* on models of universal hashing.

Following computations support and motivate our later ideas. By the Markov's inequality, we have that for every $k > 1$

$$\mathbf{Pr}(\mathbf{lpsl} > k\mathbf{E}(\mathbf{lpsl})) \leq \frac{1}{k}.$$

This fact means that less than half of all the universal functions create longest chains longer than $2\mathbf{E}(\mathbf{lpsl})$ for arbitrary stored set. For instance, in case of the system of linear transformations we may choose the *limit* as $2\mathbf{E}(\mathbf{lpsl}) \leq 1\ 076 \log m \log\log m + 88$. This *bound* follows from Corollary 5.44.

The expected length of the longest chain gives us a generic hint when the table should be rehashed if the worst case time for the find operation has to be guaranteed. The lower the value $\mathbf{E}(\mathbf{lpsl})$, or the tighter its estimate is, the better worst case *limit* is achieved.

Mentioned *bound* on the length of a chain, which is computed using the Markov's inequality and the expected value, can be further improved. If the probability density function of the random variable **lpsl** is known, then the density function may be used directly. Such *limit* l is associated with a probability $p \in (0, 1)$ that can be computed from the density function as $\mathbf{Pr}(\mathbf{lpsl} > l) \leq p$. And we can also go the other way, for the probability p we may find a minimal limit l with $\mathbf{Pr}(\mathbf{lpsl} > l) \leq p$.

In addition, for the system of linear transformations we already have the probability density function and we use it in Section 6.3, indeed. To sum up, the approach with the expected length and the Markov's inequality is more general but achieves greater limits. The approach with the probability density function gives better results. It is less general because if we know the probability density function of **lpsl**, then we are able to find $\mathbf{E}(\mathbf{lpsl})$.

Definition 6.4 (Chain length limit function, long chain, p-trimmed-system, trimming rate)**.** *Let H be a universal system of functions that map a universe U to a hash table B. Let m be the size of the hash table, $S \subset U$ be the stored set with $n = |S|$ and $\alpha = \frac{n}{m}$ be the table's load factor.*

Then function $l : \mathbb{N} \times \mathbb{R}_0^+ \to \mathbb{N}$ of variables m and α, $l(m, \alpha)$, is a chain length limit function*.*

We say that function $h \in H$ creates a long chain when hashing the set S *if there is a chain of length strictly greater than the limit value $l(m, \alpha)$.*

Moreover let $p \in (0, 1)$ be probability such that $\mathbf{Pr}(\mathbf{lpsl} > l(m, \alpha)) \leq p$, then the system

$$H_p^S = \{h \in H \mid h \text{ does not create a long chain when hashing the set } S\}$$

*is called a p-*trimmed system*. The probability p is called the* trimming rate*.*

The probability bound $p \in (0,1)$ of the existence of a long chain has important consequences for the model that limits its chains.

- At most $p|H|$ of all the functions in the original universal system H create long chains – longer than the prescribed limit $l(m,\alpha)$.
- The probability that the table needs to be rehashed, equivalently probability of selecting an inconvenient function, is lower than p provided the uniform choice of a hash function.
- During rehashing caused by an occurrence of a long chain, the probability of finding a suitable function is at least $1-p$ when assuming the uniform choice of a hash function.

Lemma 6.5. *If H_p^S is a p-trimmed system, then $|H_p^S| \geq (1-p)|H|$.*

Proof. Simply use the definition of H_p^S and that of the trimming rate p:

$$\begin{aligned} |H_p^S| &= |\{h \in H \mid \ h \text{ does not create a long chain}\}| \\ &= \mathbf{Pr}\left(\mathbf{lpsl} \leq l(m,\alpha)\right)|H| \\ &= \left(1 - \mathbf{Pr}\left(\mathbf{lpsl} > l(m,\alpha)\right)\right)|H| \\ &\geq (1-p)|H|. \end{aligned}$$

□

Regarding that every function is chosen uniformly and the unsuitable ones are discarded, we still perform the uniform selection of a hash function. The choice is restricted to the functions that do not create long chains; to the class H_p^S. Note that the restriction to the functions of the original universal system H comes from an information about the stored set.

Previous remarks are quite interesting. So now, we ask, if it is possible to use the family H_p^S as a universal one.

Theorem 6.6. *Let H be a c-universal system of hash functions, U be a universe and B be a hash table. Let $p \in (0,1)$ be the trimming rate and set $m = |B|$. Then for every $S \subset U$ the system of functions H_p^S is $\frac{c}{1-p}$-universal. Equivalently:*

$$\mathbf{Pr}\left(h(x) = h(y) \text{ for } h \in H_p^S\right) \leq \frac{1}{1-p}\mathbf{Pr}\left(h(x) = h(y) \text{ for } h \in H\right).$$

Proof. From Lemma 6.5 and from the assumption of c-universality of the original

system it follows that

$$\begin{aligned}
&\mathbf{Pr}\left(h(x) = h(y) \text{ for } h \in H_p^S\right) \\
&\quad = \frac{|\{h \in H \mid h(x) = h(y) \wedge \ h \textit{ does not create long chains}\}|}{|\{h \in H \mid \ h \textit{ does not create long chains}\}|} \\
&\quad \leq \frac{|\{h \in H \mid h(x) = h(y) \wedge \ h \textit{ does not create long chains}\}|}{(1-p)|H|} \\
&\quad \leq \frac{|\{h \in H \mid h(x) = h(y)\}|}{(1-p)|H|} \\
&\quad = \frac{1}{1-p}\mathbf{Pr}\left(h(x) = h(y) \text{ for } h \in H\right) \\
&\quad \leq \frac{c}{(1-p)m}.
\end{aligned}$$

Hence the system H_p^S is $\frac{c}{1-p}$-universal. □

Similar statements hold for the strongly universal systems. The probability of a collision in the system H_p^S is always $\frac{1}{1-p}$ times higher than the probability of a collision in the original system H.

Next statement summarises results for the systems of linear transformations.

Corollary 6.7. *For every trimming rate $0 < p < 1$ the p-trimmed system of linear transformations, $LT(U,B)_p^S$, is $\frac{1}{1-p}$-universal.*

Proof. System of linear transformations is 1-universal as seen in Remark 3.13. The fact then follows from Theorem 6.6. □

Every chain length limit function $l(m,\alpha)$ comes with an associated trimming rate, probability of the event $\mathbf{lpsl} > l(m,\alpha)$. This probability not only determines the probability of failure for a single function but it also determines the expected number of trials to find a suitable function, as stated in Lemma 6.8.

Lemma 6.8. *Let l be a chain length limit function and $p \in (0,1)$ be the trimming rate such that $\mathbf{Pr}\left(\mathbf{lpsl} > l(m,\alpha)\right) \leq p$. Then the expected number of trials to find a function, which does not create a long chain, is at most $\frac{1}{1-p}$ and the variance of this number is bounded by $\frac{p}{(1-p)^2}$.*

Proof. The probability of k independent unsuccessful searches of a function with bounded chains is at most p^k and thus the distribution of the first successful attempt is bounded by the geometric distribution. For an estimate of the expected value and the variance of the first successful attempt we need the following facts, that may be

found in [29],

$$\sum_{i=0}^{\infty} p^i = \frac{1}{(1-p)},$$
$$\sum_{i=0}^{\infty} ip^i = \frac{p}{(1-p)^2},$$
$$\sum_{i=0}^{\infty} i^2p^i = \frac{p(1+p)}{(1-p)^3}.$$

The expected time of success is then given by:

$$\sum_{i=0}^{\infty}(i+1)p^i(1-p) = (1-p)\sum_{i=0}^{\infty}p^i + (1-p)\sum_{i=0}^{\infty}ip^i = \frac{1-p}{1-p} + \frac{(1-p)p}{(1-p)^2} = \frac{1}{(1-p)}.$$

Now we estimate the variance:

$$\begin{aligned}
&\sum_{i=0}^{\infty}\left(i+1-\frac{1}{(1-p)}\right)^2 p^i(1-p) \\
&\quad= \sum_{i=0}^{\infty}(i+1)^2 p^i(1-p) - \sum_{i=0}^{\infty}2(i+1)p^i + \sum_{i=0}^{\infty}\frac{p^i}{(1-p)} \\
&\quad= \frac{1-p}{p}\left(\sum_{i=0}^{\infty}i^2p^i\right) - \frac{2}{(1-p)^2} + \frac{1}{(1-p)^2} \\
&\quad= \frac{1-p}{p}\frac{p(1+p)}{(1-p)^3} - \frac{1}{(1-p)^2} \\
&\quad= \frac{1+p}{(1-p)^2} - \frac{1}{(1-p)^2} \\
&\quad= \frac{p}{(1-p)^2}.
\end{aligned}$$

□

So the schema to obtain a chain length limit function is as follows. For a prescribed probability of failure – trimming rate p we find a minimal chain length limit function $l(m,\alpha)$ such that $\mathbf{Pr}\left(\mathbf{lpsl} > l(m,\alpha)\right) \leq p$. The probability p is chosen according to the expected number of trials required to find a function which does not create a long chain. For example in our model, we choose two trails and thus $p = 0.5$.

The lower the trimming rate p is the greater values of $l(m,\alpha)$ are obtained in order to meet $\mathbf{Pr}\left(\mathbf{lpsl} > l(m,\alpha)\right) \leq p$. In addition, from Corollary 6.7 it follows that the smaller the trimming rate p is, the better expected results are obtained. And Theorem 6.1 implies that the expected chain length still remains constant provided that the load factor α is bounded. So the small values of p give good expected results

and low number of trails required to obtain a function. On the other hand choosing a low trimming rate p gives only a poor worst case warranty.

The most interesting and the most important idea of trimming is that every p-trimmed system is an adaptation of the original class H to the stored set S.

6.3 Chain Length Limit

From now we concentrate our effort to obtain the tightest bound on the chain length for a given trimming rate $p \in (0,1)$. The corresponding bound is determined from the density function shown in Remark 5.37. This limit is used later in our model in Section 6.4.

In Theorem 6.9 the set S_e is not directly the stored set. Instead, we use the set S_e that comes from the analysis of our model shown in Theorems 6.18 and 6.20. On the other hand every set S_e is a subset of the stored set S. The size of the set S_e is bounded according to the requirements of our analysis.

Theorem 6.9. *Let $T : \mathbb{Z}_2^u \to \mathbb{Z}_2^b$ be a random uniformly chosen linear transformation, $m = 2^b$, $\alpha' \in \{1, 1.5\}$, $S_e \subset \mathbb{Z}_2^u$ such that $|S_e| \leq \alpha' m$ and $p \in (0,1)$ be the trimming rate.*

Then there is a chain length limit function $l(m) = a \log m \log\log m + b \log m$ for some $a, b \in \mathbb{R}$ depending only on α' and p. For the chain length limit function $l(m)$ we have $\mathbf{Pr}(\mathbf{lpsl} > l(m)) \leq p$.

- *When $\alpha' = 1.5$ and $p = 0.5$, set $l(m) = 57.29 \log m \log\log m$.*
- *When $\alpha' = 1$ and $p = 0.5$, set $l(m) = 47.63 \log m \log\log m$.*

Proof. It is enough to prove the statement for $|S_e| = \alpha' m$. For a smaller set S_e, with $|S_e| < \alpha' m$, it follows that $\mathbf{Pr}(\mathbf{lpsl} > l(m)) \leq p$, since it may be extended to a set that has exactly $\alpha' m$ elements and confirms to the bound. Hence the statement holds for every S_e such that $|S_e| \leq \alpha' m$ provided that it holds for every S_e with $|S_e| = \alpha' m$.

Now assume that $|S_e| = \alpha' m$ and set $f(x) = x^{\log b - \log \alpha' - \log x - \log\log x}$. From Corollary 5.9 it follows that the function $f(x)$ is decreasing in the interval $[2, \infty)$.

First, we introduce a parameter $\epsilon \in (0,1)$ which is used later in the proof to minimise the value of the limit function. If $r \geq 4$, $\alpha' \in \left(0.5, \frac{r}{\log f}\right)$ and $\epsilon \in (0,1)$, then by Remark 5.37 we have

$$\mathbf{Pr}(\mathbf{lpsl} > 2\alpha' c_\epsilon r) \leq \frac{1}{1-\epsilon} f\left(\frac{r}{\log r}\right).$$

We find the minimal value of r such that $\frac{1}{1-\epsilon} f\left(\frac{r}{\log r}\right) \leq p$. By setting the value of the variable r we also obtain the chain limit confirming to the prescribed trimming rate p.

Our next step is to define a lower bound d, $d \geq 2$, of the expression $\frac{r}{\log r}$. Since $d \leq \frac{r}{\log r}$ we have that $f(d) \geq f\left(\frac{r}{\log r}\right)$ because the function f is decreasing in the interval $[2, \infty)$. Whenever we have a value of the variable d such that $f(d) \leq (1-\epsilon)p$, then $f\left(\frac{r}{\log r}\right) \leq (1-\epsilon)p$, too. If we manage to find the minimal value of r from $d \leq \frac{r}{\log r}$, then we set the chain limit to $2\alpha' c_\epsilon r$ and the trimming rate p is achieved as well.

First, we show a way how to estimate the value of the variable r for a given $d \geq 2$.

Claim 6.10. *If $d \geq 2$ and $r = 2d \log d$, then $d \leq \frac{r}{\log r}$ and $r \geq 4$.*

Proof. Since we selected d as a lower bound for $\frac{r}{\log r}$ we have to find minimal r from the inequality $\frac{r}{\log r} \geq d$. Observe that for every $d \geq 2$ we have that $\log d \geq 1 + \log\log d$ and hence

$$\frac{\log d + \log d}{1 + \log d + \log\log d} \geq 1.$$

Putting $r = 2d \log d$ satisfies the inequality since

$$\frac{r}{\log r} = \frac{2d \log d}{1 + \log d + \log\log d} = \frac{d(\log d + \log d)}{1 + \log d + \log\log d} \geq d.$$

The value of r is thus $r = 2d \log d \geq 4$. □

The probability estimate of the event $\mathbf{lpsl} > 2c_\epsilon \alpha' r$ requires that $r \geq 4$. For the choice of the value r from Claim 6.10 it follows that $r \geq 4$ when $d \geq 2$. Because we choose $d \geq 2$, we no longer pay attention to the assumption of Remark 5.37. The remaining one $\alpha' < \frac{\log r}{2}$ may cause a problem. Its validity must be verified at the end, immediately after we state the exact choice of r.

To finish the proof set the lower bound $d = jb$ for a positive constant j. Instead of finding exact value of d, it is sufficient to find a minimal value of j. The simplification is motivated by the fact that the order of the asymptotic growth of $\mathbf{E}(\mathbf{lpsl})$ is $b \log b$. In Claim 6.11 we show that putting $d = jb$ respects this asymptotic growth. So now we just need to find the multiplicative constant as small as possible.

Claim 6.11. *Let j be a positive constant such that $d = jb \geq 2$, then the chain limit rule we propose has the form*

$$4c_\epsilon \alpha' jb(\log b + \log j).$$

Proof. We choose the value of r from Claim 6.10 as

$$r = 2d \log d = 2jb(\log b + \log j).$$

Hence our chain limit can be finally rewritten as:

$$4c_\epsilon \alpha' r = 4c_\epsilon \alpha' jb(\log b + \log j).$$

□

Claim 6.12. *To find the minimal value of d, $d \geq 2$ satisfying $f(d) \leq (1-\epsilon)p$ use the inequality*

$$(jb)^{-\log\alpha'-\log j-\log\log(jb)} \leq (1-\epsilon)p. \tag{6.13}$$

Proof. From remark Remark 5.37 we obtain the following inequality that allows us to find the minimal suitable value of d:

$$f(d) = d^{\log b-\log\alpha'-\log d-\log\log d} \leq (1-\epsilon)p.$$

We substitute jb into d to get the required result:

$$d^{\log b-\log\alpha'-\log d-\log\log d} = (jb)^{-\log\alpha'-\log j-\log\log(jb)} \leq (1-\epsilon)p.$$

□

Recall that we use the hash table $B = \mathbb{Z}_2^b$ and refer to $m = 2^b$ as to its size. Inequality 6.13 may have various interpretations.

- For fixed $0 < p, \epsilon < 1$ and a positive constant j we get a lower bound on m when our estimate becomes valid. Realise, that we can obtain arbitrarily low multiplicative constant. This follows from the fact that we are allowed to choose values for the constant j. Such estimates are valid only for large numbers of stored elements, since we need $d = jb \geq 2$ and $n \geq \frac{m}{2} \geq 2^{\frac{2}{j}-1}$.
- Or we can find the parameters ϵ and j such that multiplicative constant $4c_\epsilon\alpha' j$ is the smallest possible for the trimming rate p and the size of the hash table m. This statement is used to find the required estimate for $m \geq 4\,096$.

We use Inequality 6.13 to obtain the chain limit. The limit is computed for tables consisting of at least 4 096 buckets and the probability bound is set to 0.5. These choices were not random. When we used the formula for the first time, we gained the multiplicative constants in the order of tens. Estimates with the multiplicative constant in the order of tens start beating the most basic linear estimate, $\mathbf{lpsl} \leq \alpha' m$, when hashing thousands of elements.

Program optimising the value of the multiplicative constant only minimises the value $4c_\epsilon\alpha' j$ and does not pay any attention to the other constant. After the minimal value is retrieved, together with the values of the parameters, the value of the constant, $4c_\epsilon j \log jb$, is determined. To find the value of the constant c_ϵ according to Inequality 5.24 of Statement 5.23, additional parameters k and l are required. We use Algorithm 5 to solve this optimisation problem.

Algorithm 4 Calculate the multiplicative constant for parameters $p, m, \alpha', \epsilon, k, l$.

$c_\epsilon \leftarrow$ constant c_ϵ computed with parameters k, ϵ and l
$r \leftarrow (1-\epsilon)p$ {Right side, inevitably achieved bound.}
$j \leftarrow 1$

{Lower the value of j so that the right side is skipped.}
$l \leftarrow (j \log m)^{-\log(\alpha')-\log(j)-\log(\log(j \log m))}$
while $l < r$ **and** $j > 0$ **do**
  $j \leftarrow j - STEP$;
  $l \leftarrow (j \log m)^{-\log(\alpha')-\log(j)-\log(\log(j \log m))}$
end while

$j \leftarrow j + STEP$
return $4c_\epsilon \alpha' j$

Algorithm 5 Calculate the smallest limit for $p = 0.5$, $m \geq 4\ 096$ and prescribed α'.

$c \leftarrow \infty$
for $k \in [2, 4]$ with $STEP$ **do**
  for $l \in [1.3, 3]$ with $STEP$ **do**
    for $\epsilon \in [0.85, 0.99]$ with $STEP$ **do**
      if $c >$ multiplicative constant for $p = 0.5, m = 4\ 096, \alpha', \epsilon, k, l$ **then**
        $c \leftarrow$ computed multiplicative constant
      end if
    end for
  end for
end for

return c

The form of the chain length limit function follows from Claim 6.11 and Algorithm 5, which is able to compute the minimal constant a and the corresponding constant b. □

The best result achieved for $\alpha' = 1.5$, $m = 2^b \geq 4\ 096$ is for $\epsilon = 0.96$, $j = 0.74$ and equals $4c_\epsilon \alpha' j = 57.29$. The same approach gives multiplicative constant 47.63 for $\alpha' = 1$. The assumption $\alpha' < \frac{r}{2}$ holds since $\alpha' \leq 1.5 < 6 < jb \leq d \log d = \frac{r}{2}$.

Now compare this limit with the linear estimate, **lpsl** $\leq \alpha' m$, for the size of the hash table $m \geq 4\ 096$ and the number of stored elements $n \geq \alpha' m = 0.5 \cdot 4\ 096 = 2\ 048$. Linear estimate on the length of the longest chain equals $n = 2\ 048$. Our estimate equals approximately $57.29 \log m \log \log m \leq 57.29 \cdot 12 \cdot 2.27 \leq 1\ 555$ which is already far better.

These limits may be improved since we neglected the part $4c_\epsilon \alpha' j \log j \log m$ which is negative.

6.4 The Proposed Model

The model we propose is a slight modification of the simplest model of universal hashing scheme with separated chains. We distinguish the two cases – when Delete operation is not allowed or when the stored elements can be removed.

- **Universal class.** We use the system of linear transformations as the universal family of functions. The universe $U = \mathbb{Z}_2^u$ and the target space, the hash table, is referred to as $B = \mathbb{Z}_2^b$. We may imagine the situation as hashing u-bit binary numbers to the space of b-bit binary numbers. We refer to the size of the hash table as to $m = |B| = 2^b$.

- **Load factor rule.** The load factor of the table is kept in the predefined interval. If the load factor is outside the interval, whole table is rehashed into a greater or smaller table. New size is chosen so that the load factor was as near 0.5 as possible. Load factor is maintained in the interval $[0.5, 1]$ for the scheme without the delete operation. We need the interval $[0.25, 1]$ if the delete operation is allowed.

- **Chain limit rule.** When there is a chain longer than the limit value $l(m)$, then the hash table is rehashed. The value $l(m)$ is chosen according to Theorem 6.9 with the trimming rate $p = 0.5$. The chain length limit function of our model does not depend on the table's load factor so we omit the parameter α and use just $l(m)$.

  The exact chain length limit function comes from Theorem 6.9. If Delete operation is forbidden, then set the limit value $l(m) = 47.63 \log m \log\log m$ and use $\alpha' = 1$. When Delete operation is allowed, the chain length limit function $l(m) = 57.29 \log m \log\log m$ and $\alpha' = 1.5$.

Estimates for the chain limit rule are valid for the tables consisting of at least 4 096 slots. There are two ways how to deal with this problem. First, we may set the initial size of the table to 4 096 buckets. If the size of the table is 4 096 buckets, then the lower bound of the load factor rule is not applied and we allow the table's load factor to be in the interval $[0, 1)$.

Since not every hash table grows to a size of 4 096 buckets, this overhead becomes unacceptable. The other option lies in turning off the chain limit rule when the table has less than 4 096 buckets.

6.5 Time Complexity of Computing a Hash Value

Since our model is based on the system of linear transformations, what is quite unusual, we have to deal with the time complexity of computing an element's hash value. We ask if the time required to compute the hash value is still constant. When we bound the size of the universe $|U|$, certainly it is. But this time may be worse, when compared to the time required by linear system 3.7.

Linear system uses a constant number of arithmetic operations. If we do not bound the size of the universe $|U|$, we may compute them in $O(\log^2 |U|)$ time.

Our system, system of linear transformations, represents every transformation by a binary matrix $M \in \mathbb{Z}_2^{b \times u}$. Every element $\vec{x} \in U$ is a vector consisting of u bits. To obtain its hash the matrix-vector multiplication $M\vec{x}$ is performed. We can certainly do it in $O(ub) = O(\log |U| \log |B|)$ time.

If the size of the universe is bounded, this time is constant. Despite the better asymptotic bound, in practise, the time is worse than that of linear system. Assume, that we represent elements by a word determined by the computer's architecture. In this case linear system needs just three operations, one multiplication, one addition and a modulo. Computing a matrix-vector multiplication can be optimised, but is not so fast.

In addition, when hashing is applied, elements usually fit in the word of the underlying architecture. Therefore the arithmetic operations become constant and this is our case, too. To deal with the longer running times of computing a hash value, sometimes, it is possible to cache once computed hash value within the element. This solution is a trade-off that improves the time complexity but consumes additional memory.

6.6 Algorithms

In Section 6.4 we propose a model of universal hashing without any exact definition. Precise algorithms, showing how the operations work, are required. Despite the fact that the algorithms are very similar to those shown in Chapter 2, now they are described exactly.

First, let us describe the hash table's variables and notation. Hash table contains variables storing its size $Size$, the number of represented elements $Count$ and array T is the hash table itself. Function $Limit$ denotes the chain length limit function and $Hash$ is the current universal function – a linear transformation represented by a binary matrix. Every bucket $T[i]$ contains two variables $T[i].Size$, the number of elements inside the slot, and $T[i].Chain$, the linked list of the elements in the bucket.

Initialisation. We use Algorithm 5 from the proof of Theorem 6.9 to compute the bound for $\alpha' \in \{1, 1.5\}$, according to the status of Delete operation, and for $p = 0.5$. Theorem 6.9 states that the limit function is in the form $a \log m \log \log m + b \log m$ for $a, b \in \mathbb{R}$ depending only on α' and p. Thus we need to store just two real numbers a and b to represent the chain length limit function. Let us note that Theorem 6.9 gives the chain length limit function for a different parametrisation, too. However, in such case the amortised analysis must be changed.

Initialisation creates a new empty table with the prescribed size. It also chooses a universal function by the random initialisation of the bits in the matrix $Hash$. The values of bits are chosen randomly each bit having the same probability 0.5. If the delete operation is allowed $\alpha_{min} = 0.25$, $\alpha_{max} = 1$ and $\alpha' = 1.5$, if it is not, then $\alpha_{min} = 0.5$, $\alpha_{max} = 1$ and $\alpha' = 1$.

Algorithm 6 Initialisation of the hash table

Require: $p \in (0, 1)$, default 0.5
Require: $m \in \mathbb{N}$, default 4 096

Ensure: the random uniform choice of a linear transformation $Hash$

initialise the value α' according to the delete operation
use Algorithm 5 to compute the bound $Limit(m)$ for the prescribed p and α'
initialise the variables α_{min} and α_{max}
create the table T of size m
$Size \leftarrow m$
$Count \leftarrow 0$
choose the hash function – random binary matrix $Hash$

Algorithm 7 Rehash operation

Require: $m \in \mathbb{N}$, default $Size$ {New size of the hash table.}

repeat
  $T'.Initialise(m)$
  turn off the chain limit rule in T'
  turn off the load factor rule in T'
  for all element x stored in the hash table **do**
    $T'.Insert(x)$
  end for
until chain limit rule is satisfied in T'

swap T and T'
turn on the chain limit rule in T'
turn on the load factor rule in T'
free T

Algorithm 8 Find operation

Require: $x \in U$
$i \leftarrow h(x)$
if T[i].Chain.Contains(x) **then**
  return true {Successful find.}
else
  return false {Unsuccessful find.}
end if

Algorithm 9 Insert operation

Require: $x \in U$

```
i ← h(x)
if T[i].Chain.Insert(x) then
    {Set the number of stored elements.}
    Count ← Count + 1
    T[i].Size ← T[i].Size + 1

    Rehash ← false
    {The load factor rule may be violated.}
    if Count > Size * α_max then
        NewSize ← 2 * Size
        Rehash ← true
    else
        NewSize ← Size
    end if

    {The chain limit rule may be violated.}
    if T[i].Size > Limit(Size) then
        Rehash ← true
    end if

    {Rehash the table if needed, new function is chosen.}
    if Rehash then
        Rehash(NewSize)
    end if

    return true {Successful insert.}
else
    return false {Unsuccessful insert.}
end if
```

Algorithm 10 Delete operation

Require: $x \in U$
  $i \leftarrow h(x)$
  if T[i].Chain.Remove(x) **then**
    $Count \leftarrow Count - 1$
    $T[i].Size \leftarrow T[i].Size - 1$

    if $Count < Size * \alpha_{min}$ **then**
      Rehash($Size/2$)
    end if

    return true {Successful delete.}
  else
    return false {Unsuccessful delete.}
  end if

Rehash operation To enumerate the stored elements in Rehash operation, Algorithm 7, we iterate every chain of the table. A common optimisation is to place all the stored elements into a linked list. This allows faster enumeration but causes a space overhead.

Whenever a load factor rule or the chain limit rule is violated, the table is rehashed using a new function chosen in initialisation. Both Initialisation and Rehash operation, Algorithms 6 and 7, take an argument m specifying the new size, so that it is possible to rehash the table into a larger or smaller one.

Find, Insert and Delete operations Find is very straightforward. Let us notice that the linked list manipulation operations return **true** in the successful case, if an element is found, deleted or inserted, and otherwise return **false**. Insert and Delete operations are slightly complicated compared to the original ones because of the both rules that are required to hold.

6.7 Potential Method

In Section 6.8 we estimate the expected amortised time complexity using the potential method. So let us explain it first. Assume that we have a data structure and a sequence of operations $\{o_i\}_{i=1}^n$ performed on it. We want to estimate the running time of the sequence. Of course, we can use the worst case time for each operation but this may be very misleading.

Consider a simple array. The elements are placed at the array's end. If there is a free position, we store the element. When the array is full, then we double its size and store the element. Clearly, if we store n elements inside the array, then the worst case time required to insert another one is $O(n)$. We ask, what time is required to store n elements in such an array. The estimate using the worst case time for each operation gives the result $O(n^2) = O(\sum_{i=1}^n i)$. Amortised analysis gives a

far better result, $O(n)$. This result can be explained by the fact that the fast inserts accumulate a potential that is used later by a following slow insert.

Now we describe the potential method – a method how to estimate the amortised complexity. The potential of the data structure and an operation's amortised cost are tools which distribute the running time of a sequence among the operations more evenly. Assume that we have a data structure at the initial state s_0 and we perform a sequence of $n \in \mathbb{N}$ operations $\{o_i\}_{i=1}^n$. Every operation o_i changes the state of the data structure from s_{i-1} to s_i, $i \in \{1, \ldots, n\}$.

Every state has a real-valued potential given by the potential function p. Hence we obtain a sequence of potentials p_i for every $i \in \{0, \ldots, n\}$. The potential p_0 is the initial potential and is often chosen as zero. So the operations not only change the data structure, they change its potential, too. We define $a_i = t_i + p_i - p_{i-1}$ as the amortised cost of the i^{th} operation where t_i is its running time.

Definition 6.14 (Amortised complexity)**.** *Assume that we perform an operation of a data structure. Let t be the time required to perform the operation. Let p_b be the potential of the data structure before performing the operation and p_a be the potential after. Then the* amortised complexity *of the operation*

$$a = t + p_a - p_b.$$

The time taken by the operations in a sequence o, $T_o = \sum_{i=1}^n t_i$, may be estimated by the amortised time of the sequence, $A_o = \sum_{i=1}^n a_i$, as

$$T_o = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} (a_i - p_i + p_{i-1}) = A_o + p_0 - p_n.$$

Let us show some facts regarding the potential functions and the amortised complexity of a sequence of operations.

Claim 6.15. *Assume that we estimate the amortised complexity by a potential function p. Let $o = \{o_i\}_{i=1}^n$ be the performed sequence of operations, p_0 be the starting potential and p_n be the potential after performing the last operation.*

(1) If $p \geq 0$, then $T_o \leq A_o + p_0$.

(2) If $p \leq 0$, then $T_o \leq A_o - p_n$.

(3) If $p_0 = 0$, then $T_o = A_o - p_n$.

For randomised algorithms we may take the expected time consumed by an operation.

Definition 6.16 (Expected amortised complexity)**.** *Assume that an operation of a randomised data structure is performed and the operation runs in time t. Let p_b be the potential before performing the operation and p_a be the potential after. Then the* expected amortised complexity *of the operation is defined as*

$$a = \mathbf{E}\left(t + p_a - p_b\right).$$

A description of the potential method may be found in [23] and [25].

6.8 Expected Amortised Complexity

Expected amortised time complexity of the introduced scheme is analysed in the next two theorems in either of the two cases – when Delete operation is allowed or not.

Let us discuss the situation of Lemma 6.17. We are given a sequence of sets $S_1 \subseteq \cdots \subseteq S_k \subseteq S_e$ that should be represented in a hash table. We start with the initial hash function h_0. Set S_1 causes violation of the chain limit rule for function h_0. In order to enforce the rule, we select random functions $h_1, h_2, \ldots$ until we find a suitable function for the set S_1, denote it h_{i_1}. Later, after some inserts, we obtain a set S_2 and the function h_{i_1} is no longer suitable. We continue by selecting functions $h_{i_1+1}, \ldots, h_{i_2}$ with h_{i_2} being suitable for the set S_2. The chain length limit function is chosen for the set S_e and for the trimming rate p from Theorem 6.9. We need to find the expected number of trials, the number of selected functions, needed to enforce the chain limit rule for the sets $S_1, \ldots, S_k$. The set S_e is a set that comes from the analysis, we use the same set S_e is in Theorem 6.9.

Lemma 6.17. *Let $\mathbb{Z}_2^u$ be the universe, $\mathbb{Z}_2^b$ be the hash table with the size $m = 2^b$, $p \in (0,1)$ be the trimming rate and $\alpha' \in \{1, 1.5\}$. Let the chain length limit function be chosen according to Theorem 6.9 for α' and p.*

Let $S_1 \subseteq \cdots \subseteq S_k$ be a sequence of represented sets and $S_e \subset \mathbb{Z}_2^u$ be a set such that $|S_e| \leq \alpha' m$ and $S_k \subseteq S_e$. Let $h_0, h_1, \ldots, h_l$ be a sequence of random uniformly chosen linear transformations selected to enforce the chain limit rule for the sequence of sets. Assume that $0 = i_0 < \cdots < i_k = l$ is the sequence such that

(1) the functions $h_{i_j}, h_{i_j+1}, \ldots, h_{i_{j+1}-1}$ create a long chain for the set S_{j+1} for every $j \in \{0, \ldots, k-1\}$,

(2) the function h_{i_j} does not create a long chain for the set S_j, $j \in \{1, \ldots, k\}$.

Then $\mathbf{E}(l) = \frac{1}{1-p}$.

Proof. First, observe that if a function h is suitable for the set S_e, then it is suitable for every set $S_1, \ldots, S_k$.

By Lemma 6.8 the expected number of trials needed to represent the set S_e without a long chain is $\frac{1}{1-p}$.

The sequence of functions $h_1, \ldots, h_l$ is random and the selection of every function is uniform. The chain length limit function is chosen according to Theorem 6.9 – it fits for the set S_e. Hence if we do not consider the sets $S_1, \ldots, S_k$, we have that $\mathbf{E}(l) = \frac{1}{1-p}$.

If a function h_a, $1 \leq a < l$ creates a long chain for a set S_b, $i_{b-1} \leq a < i_b$, then it certainly creates a long chain for the set S_e. Thus if we fail with the function h_a for the set S_b, then we fail with the same function for the set S_e, too. In this situation we continue by choosing functions $h_{a+1}, h_{a+2}, \ldots$. The sequence of functions $h_1, \ldots, h_l$ for the sequence of sets, remains the same for the single set S_e, provided that the random choice is the same. So the functions selected for the sequence of sets may

be selected when rehashing the set S_e, too. Hence we do not need to consider the sequence of sets when choosing a right function for the set S_e.

Thus $\mathbf{E}(l) = \frac{1}{1-p}$ as observed before. □

Theorem 6.18. *Consider hashing with forbidden Delete operation. Then the operations Find and Insert have the expected amortised time complexity $O(1)$. Moreover, if the size of a hash table is m, then the find operation runs in $O(\log m \log\log m)$ time in the worst case.*

Proof. For the expected amortised complexity analysis we use the potential method. Let the expression $\alpha m - m$ denote the potential of the scheme. This is the negative value of the number of the remaining successful insert operations which would make the table's load factor reach one. Whenever a successful insertion is performed we check if

- the prolonged chain does not violate the chain limit rule or
- the load factor is not greater than 1.

If either of the two conditions is violated the whole table is rehashed. We have four cases to analyse.

- *Find operation* or an *unsuccessful Insertion operation* is performed. From Theorem 6.1 and Corollary 6.7 we know that it takes $O(1)$ expected time only. The potential does not change. Since the chains are bounded by $l(m)$, its worst case running time is $O(\log m \log\log m)$.
- *Insert operation* is performed and the *Rehash operation* is *not necessary*. Then from Theorem 6.1 the operation takes $O(1)$ time in the expected case. The potential is increased by one.
- *Rehash operation* after an *Insert operation* is required because the *load factor* exceeds one. The rehash itself takes the $O(m)$ time and the size of the table is doubled. Thus the resulting potential is $-m$. The potential before the operation equals 0. So the operation took $O(1)$ expected amortised time.
- Suppose that a Rehash operation after an *Insert* is needed because of the *violation of the chain limit rule*. We seek for a new function satisfying the rule without resizing the table. For convenience we define a sub-sequence of operations called a *cycle*.

  Definition 6.19. Cycles *create a partitioning of the original sequence of operations. Each* cycle *is a sub-sequence consisting of the operations between two immediate inserts which cause the violation of the load factor rule. The first insert causing the violation is included in the cycle and the second one belongs to the next cycle.*

We refer to the current cycle as to the cycle which contains the analysed operation. Now we find the number of rehash operations that are caused by the chain limit rule violation and occur in a single cycle. Let S_e be the set represented at the end of the current cycle. Then by Lemma 6.17 we need just $\frac{1}{1-p}$ trials in the current cycle.

Now we compute the expected amortised complexity of the insert operations causing the violation of the chain limit rule in a single cycle. For every table that consists of m slots at the beginning of the cycle, there are exactly $0.5m$ insert operations raising the load factor from 0.5 to 1. The expected time spent by fixing the chain limit rule in a cycle is, by Lemma 6.17, $\frac{1}{(1-p)}O(m)$. We can divide this amount of time along the cycle of $0.5m$ inserts. This distribution raises the expected amortised time for every insert operation only by a constant. The potential is incremented by one. The expected time of Insert operation without Rehash operation is $O(1)$.

We have one issue to care about. In the last case we distributed the time evenly along a complete cycle. If the number of inserts is not a power of two, we may possibly distribute a long time along a non-complete cycle. With this distribution we can not obtain a constant amortised time for the insert operation. However, we can distribute the time spent by fixing the chain limit rule from the last incomplete cycle along all the insert operations performed so far.

Thus the expected amortised time complexity of every analysed operation is $O(1)$. □

One can find a better potential function proving the previous result more formally. We use such an explicitly expressed potential function in the next theorem. We assume that Delete operation is allowed.

Theorem 6.20. *If the initial hash table is empty and Delete operation is allowed, then the expected amortised time complexity of every operation is constant. Moreover, if the size of a hash table is m, then the find operation runs in $O(\log m \log\log m)$ time in the worst case.*

Proof. In this proof we have two types of the operation cycles. We need to distinguish between the work required to enforce the load factor rule and the time spent by keeping the chain limit rule. In the analysis with forbidden Delete operation the situation is simpler and these cycles are the same. And recall the difference, the chain length limit function is computed for $\alpha' = 1.5$. The load factor α is now in the interval $[0.25, 1]$.

We deal with the amortised time of Find and unsuccessful Insert or Delete operations in advance. Their expected running time is proportional to the expected chain length. From Theorem 6.1 it follows that this value is constant. Since chains are bounded by $l(m)$ we have that the worst case time of Find operation is $O(\log m \log\log m)$. These operations do not change the potential[1]. Our analysis is thus simplified by omitting finds and unsuccessful delete and insert operations.

[1]The potential is defined later in the proof.

Let the sequence $o = \{o_i\}_{i=1}^{n}$ denote the performed operations, the i^{th} operation $o_i \in \{Insert, Delete\}$. The following two definitions make the analysis more exact and clear.

Definition 6.21 (α-cycle)**.** *The* α-cycle *consists of the operations between the two immediate operations that cause the violation of the load factor rule. Every* α*-cycle contains the second operation causing the violation and the first one is included in the previous* α*-cycle. The operation* o_1 *is contained in the first* α*-cycle.*

First, notice that for this definition it is not important if the upper or lower bound of the load factor is the cause of the violation. When a rehash operation is executed, the table size is chosen so that the load factor α was as near 0.5 as possible.

The next definition of the α-cycle is intended for the analysis of the time spent by fixing the violations of the load factor rule.

Definition 6.22 (l-cycle)**.** *The* l-cycles *are the partitioning of the sequence* o *such that every l-cycle ends*

- *after an operation causing the violation of the load factor rule or*
- *when we performed* $0.5m$ *insertions from the beginning of the l-cycle and the load factor did not exceed the value of* 1*.*

The l-cycle allows us to analyse the work caused by the chain limit rule. Both l-cycles and α-cycles divide the sequence o. Notice that if an α-cycle ends at the position i, the corresponding l-cycle also ends at the same position.

The analysis now takes the i^{th} operation for every $i = 1, \ldots, n$. We show that the expected amortised time complexity of the opertion o_i is $O(1)$ independently on its type. The potential now consists of the two parts, p_1 and p_2. Let $e = \frac{1}{(1-p)}$ denote the expected number of rehash operations, the expected number of trials, when finding a suitable function.

Above all we want every simple insertion and deletion to take $O(1)$ time only. Thus the potential consists of the part $p_1 = 4ei_\alpha + 8ed_\alpha$ where i_α is the number inserts and d_α is the number of delete operations performed so far in the current α-cycle.

Next, we need to distribute the work caused by the chain limit rule. The second part of the potential $p_2 = 2ei_l + (ce - r)m$. The value i_l denotes the number of insertions performed so far in the current l-cycle. The variable r denotes the number of performed Rehash operations, which are caused by the chain limit rule violation, from the initial state. The variable c denotes the number of started l-cycles from the beginning so far. The overall potential $p = p_1 + p_2$.

The analysis of Delete operation is simpler and comes first. When a deletion is performed we have to discuss the two cases:

- Simple successful deletion is expected to take $O(1)$ time. We search in chains that have constant expected length. The potential is increased by $8e$ since the number of deletions in p_1 gets increased by one.

- The load factor α violates the lower bound of the load factor rule. Realise that there are at least $0.25m$ deletions performed in the current α-cycle, let us explain why. At the beginning of the cycle the table has the size of m slots and the load factor equals 0.5. At the end of the cycle the load factor decreased to 0.25. Such a descent may be caused by at least $0.25m$ successful delete operations. Let us discuss the potential change. First, the potential difference of the part p_1 is less or equals $-2em$ since i_α and d_α get zeroed. The second part p_2 gets increased by at most em since a new l-cycle is started and the value i_l is zeroed. Rehashing of the table takes $O(em)$ expected time. Hence the expected amortised cost of the operation is $O(1)$.

The analysis of the first two cases of Insert operation remains similar to the case with forbidden Delete:

- Suppose no Rehash operation after an Insert is performed. The searched chain has the constant expected length and the potential is increased by $4e + 2e$.

- If the load factor exceeds the upper bound, then there are at least $0.5m$ insertions in the current α-cycle. It follows that the potential p_2 is certainly greater or equals $2em$. After performing the operation a new l-cycle is started. The potential p_2 is raised by em because the variable c gets incremented by one. But it is also lowered by at least $2em$ because i_l is set to zero. The rehash operation is expected to take $O(em)$ time. Regarding the fact that the potential p_1 only decreases we expect $O(1)$ amortised time for Insert operation violating the load factor rule.

- Insert operation is the last one in the l-cycle and the chain limit rule is not violated. Then there are $0.5m$ insertions in the current l-cycle, equivalently $i_l = 0.5m$ and $p_2 = em + (ce - r)m$. Since a new l-cycle is started, the i_l term is set to zero and the value c is incremented by one. Therefore there is no potential change in the part p_2. The part p_1 only decreases.

- Analysis of Insert operation, which violates the chain limit rule, remains. Time spent by Rehash operation equals $O(m)$ times the number of fails when finding a suitable function. The potential p_2 is decreased by the same value, since the variable r gets incremented by the number of fails. The potential increase in the part p_1 and p_2 equals $6e$. Without Rehash operation the expected time of Insert is $O(1)$. The expected amortised time complexity of the analysed operation is thus constant provided that $\mathbf{E}\left((ce - r)m\right) = 0$.

  First, we show why the expected number of fails in one l-cycle equals e. Let S be the set stored after performing the analysed operation. Let S_e be the set created from the set S by the remaining inserts of the current l-cycle. Because the table's load factor is maintained lower than 1 and there are at most $0.5m$ inserts in every l-cycle, we have that $|S_e| \leq 1.5m$ and $S \subseteq S_e$. From Lemma 6.17 we have that $e = \frac{1}{1-p}$ is the expected number trials needed to find a suitable function for every l-cycle.

Second, we show that $\mathbf{E}\left((ce-r)m\right)=0$. Define the random variable C_i equal to the number of rehash operations required to fix the chain limit rule in the i^{th} l-cycle for $i=1,\ldots,c$. From the previous observation it follows that $\mathbf{E}\left(C_i\right)=e$. Clearly $r=\sum_{i=1}^{c}C_i$. From Lemma B.18 it follows that

$$\mathbf{E}\left((ce-r)m\right)\leq m\left(ce-\mathbf{E}\left(\sum_{i=1}^{c}C_i\right)\right)=m(ce-c\mathbf{E}\left(C_1\right))=mc(e-e)=0.$$

Since $\mathbf{E}\left((ce-r)m\right)=0$ and $p_0=0$, the expected potential is always non-negative. Hence $T_o=A_o+p_0-p_n\leq A_o$ and our analysis is thus correct.

We showed that the expected amortised complexity is constant for every operation. □

Let us note that the fact $\left|\frac{(ce-r)m}{n}\right|\xrightarrow{n\to\infty}0$ indicates that it might be possible to show that the amortised complexity is constant, without any expectation. Since universal hashing is a randomised algorithm, such result would be certainly remarkable. However, if it was really true, there would be still many troubles showing the result.

So, one may ask if using the Law of Large Numbers, Theorem B.23, can not help. Indeed, from Lemma 6.8 we have that $\mathbf{Var}\left(C_i\right)$ is finite and it may be applied:

$$\begin{aligned}\left|\frac{(ce-r)m}{n}\right|&\leq m\left|\frac{ce-\sum_{i=1}^{c}C_i}{c}\right|\\&\leq m\left|\frac{c\mathbf{E}\left(C_1\right)}{c}-\frac{\sum_{i=1}^{c}C_i}{c}\right|\\&=m\left|\mathbf{E}\left(C_1\right)-\frac{\sum_{i=1}^{c}C_i}{c}\right|.\end{aligned}$$

Clearly, the convergence in probability stated by Theorem B.23 holds:

$$\left|\mathbf{E}\left(X_1\right)-\frac{\sum_{i=1}^{c}X_i}{c}\right|\xrightarrow{c\to\infty}0.$$

But in the case of $m\left|\mathbf{E}\left(C_1\right)-\frac{\sum_{i=1}^{c}C_i}{c}\right|$ we have to be careful. There is an infinite class of sequences $\{o_i\}_{i=1}^{n}$ for which $m\in\Omega(c)$. Thus for these sequences, if $c\to\infty$, then $m\to\infty$ as well.

In order to obtain convergence in probability we have that

$$\begin{aligned}\mathbf{Pr}\left(m\left|\mathbf{E}\left(C_1\right)-\frac{\sum_{i=1}^{c}C_i}{c}\right|\geq\epsilon\right)&=\mathbf{Pr}\left(\left|\mathbf{E}\left(C_1\right)-\frac{\sum_{i=1}^{c}C_i}{c}\right|\geq\frac{\epsilon}{m}\right)\\&\leq\frac{\mathbf{Var}\left(C_i\right)m^2}{m\epsilon^2}=\frac{m\mathbf{Var}\left(C_i\right)}{\epsilon^2}.\end{aligned}$$

This is a problem, since when $m\to\infty$, we can not expect that the probability converges to zero for a fixed ϵ.

For the class of sequences satisfying $m \in \Omega(c)$ the inequalities we used for estimate of $\left|\frac{(ce-r)m}{n}\right|$ mean only a multiplicative factor. Thus the estimate is tight and we see that the bound obtained from the Chebyshev's inequality is not sufficient. Its asymptotic rate is not high enough and the factor m appears. However, because we know the probability distribution of C_i, maybe we could exploit it and be more accurate. This problem remains open.

Chapter 7

Conclusion

In this work we present a model of universal hashing which preserves the constant expected length of a chain. The running time of the find operation is then $O(1)$, too. The model uses the system of linear transformations and exploits its remarkable properties shown in [3]. In addition, these results are substantially improved so that they allow the construction of the model. We are also able to show that the expected amortised running time of the insert and delete operations is constant. Not only that, our model bounds the worst case running time of the find operation in $O(\log m \log \log m)$.

Because the model is based on the system of linear transformations, the time to compute the hash value of an element is worsened, despite its asymptotic behaviour. The solution that we propose is to store once computed hash values within the object. This optimisation takes the advantage of the warranties provided by the model, if the find operation is dominant.

7.1 Future Work

There are many ways how to improve the model. For example, we can go the way of the tighter estimates. Although, it may be very interesting to describe the behaviour of double hashing when it is used with a universal class of functions. Combined with the system of linear transformations it may be possible to obtain a similar worst case bound without violating the expected running times.

Our next option is to use ideas of perfect hashing. Every chain may be represented by a hash table allowed to have a small load factor. Whenever the elements in the bucket can not be accessed in a constant time, then it might be possible to rehash only the small table instead of the large one. This approach may bring another speedup.

Another brilliant idea comes from the area of load balancing. We can hash by two functions simultaneously and a newly stored element is placed into a smaller bucket. The find operation has to search in both buckets associated with an element. However, as stated in [28], the expected worst case time for classic hashing is then substantially better and the expected complexity is preserved. The question is

whether using two universal functions may help.

Unfortunately, the thesis does not show the experimental results. A high quality benchmark of the model is required. The benchmark needs to be performed with and without the mentioned optimisation – caching of the hash values. This should show the influence of the universal system on the running times. The benchmark also has to show when the warranty is needed in dependence on the operations composition and the input distribution. If we try inputs created by a random number generator, then they are uniformly distributed. So the obtained chains are short even for classic hashing. On the other hand such good inputs are seldom present. The real world inputs, which are inconvenient for classic hashing, should be another output of the benchmark. On the other hand we should point out when the classic hashing or the other implementations outperform our model. When the chains are longer and the find operation is the most frequent one, then it is convenient to have a worst case warranty especially for large number of stored elements. The question is, how frequent the find operation has to be when compared to the modifying ones.

Of course, usage of simpler classes brings faster times of computing the hash codes. Linear classes are quite similar to each other. Maybe we could find a correspondence allowing the results found for the class of linear transformations to be brought into a faster class.

Models providing a reasonable worst case warranty with a good expected time complexity may be a suitable solution for various set representation problems. Current models of hashing may provide such qualities when enriched by simple rules. Also the approach of relaxation, shown in [19] for red black trees, may be helpful when achieving similar results. Data structures providing strong warranties are usually slower in the average case. Their relaxation worsens the warranties but improves the average case. Relaxation may be considered a reverse approach compared to ours.

Bibliography

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[2] Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Developments from a June 1996 seminar on Online algorithms*, pages 13–51, London, UK, 1998. Springer-Verlag.

[3] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Linear hash functions. *J. ACM*, 46(5):667–683, 1999.

[4] C.R. Aragon and R.G. Seidel. Randomized search trees. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 540 –545, oct-1 nov 1989.

[5] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Inf.*, 1:173–189, 1972.

[6] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[7] Pedro Celis, Per-Ake Larson, and J. Ian Munro. Robin hood hashing. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:281–288, 1985.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms, second edition, 2001.

[9] Luc Devroye and Pat Morin. Cuckoo hashing: further analysis. *Inf. Process. Lett.*, 86(4):215–219, 2003.

[10] Luc Devroye, Pat Morin, and Alfredo Viola. On worst case robin-hood hashing, 2004.

[11] M. Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 6–19, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[12] Martin Dietzfelbinger and Ulf Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *SODA '09: Proceedings of the twentieth Annual*

ACM-SIAM Symposium on Discrete Algorithms, pages 795–804, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[13] Martin Dietzfelbinger and Ulf Schellbach. Weaknesses of cuckoo hashing with a simple universal hash class: The case of large universes. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM*, volume 5404 of *Lecture Notes in Computer Science*, pages 217–228. Springer, 2009.

[14] Michael Drmota. An analytic approach to the height of binary search trees. *Algorithmica*, 29(1):89–119, 2001.

[15] W. Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1.* Wiley, New York, NY, third edition, 1968.

[16] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[17] E.M. Landis G.M. Adelson-Velskii. An algorithm for the organization of information. *Sov. Math. Dokl. 3*, pages 1259–1262, 1962.

[18] Leonidas J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *FOCS*, pages 8–21, 1978.

[19] Sabine Hanke, Thomas Ottmann, and Eljas Soisalon-Soininen. Relaxed balanced red-black trees. In *CIAC '97: Proceedings of the Third Italian Conference on Algorithms and Complexity*, pages 193–204, London, UK, 1997. Springer-Verlag.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[21] Maurice Herlihy, Nir Shavit, and Moran Tzafrir. Hopscotch hashing. In *DISC*, pages 350–364, 2008.

[22] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[23] Alena Koubková and Václav Koubek. *Datové struktury.* MATFYZPRESS, Prague, Czech Republic, 2010.

[24] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer, 1984.

[25] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox.* Springer, 2008.

[26] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2001.

[27] Peter B. Miltersen. Universal hashing, 1998.

[28] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[29] Lennart Råde and Bertil Westergren. *Mathematics handbook for science and engineering*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 1995.

[30] Bruce Reed. The height of a random binary search tree. *J. ACM*, 50(3):306–332, 2003.

[31] Sheldon M. Ross. *Probability Models for Computer Science*. Academic Press, Inc., Orlando, FL, USA, 2001.

[32] Shai Rubin, David Bernstein, and Michael Rodeh. Virtual cache line: A new technique to improve cache exploitation for recursive data structures. In *In Proceedings of the 8th International Conference on Compiler Construction*, pages 259–273. Springer, 1999.

[33] A. Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *SFCS '89: Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 20–25, Washington, DC, USA, 1989. IEEE Computer Society.

[34] Mark N. Wegman and J. Lawrence Carter. New classes and applications of hash functions. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 175–182, Washington, DC, USA, 1979. IEEE Computer Society.

Appendix A

Facts from Linear Algebra

Some definitions and facts, which are used in the work, are mentioned and proved in the appendix. Basic definitions of *vector space*, *linear transformation* and facts about solving linear equations can be found for example in [26].

Definition A.1. *Let V be a vector space and A be a subset of V. For a vector $\vec{v} \in V$ define the set $\vec{v} + A$ as*

$$\vec{v} + A = \{\vec{v} + \vec{a} \mid \vec{a} \in A\}.$$

Definition A.2. *Let V be a vector space and A, B be subsets of V. Define the set $A + B$ as*

$$A + B = \{\vec{a} + \vec{b} \mid \vec{a} \in A, \vec{b} \in B\}.$$

Definition A.3 (Affine subspace)**.** *Let V be a vector space, $V \leq U$ be its subspace and $\vec{a} \in U$. The set $A = \vec{a} + V$ is an* affine subspace *of the vector space U.*

Lemma A.4. *Let $A = \vec{a} + \vec{V}$ be an affine subspace of a vector space U determined by a subspace $V \leq U$ and a vector $\vec{a} \in U$. Then $A = \vec{b} + V$ for every $\vec{b} \in A$.*

Proof. Every vector $\vec{x} \in A$ may be written in the form $\vec{x} = \vec{a} + \vec{v_x}$ for a vector $\vec{v_x} \in V$. We need to rewrite it in the form $\vec{x} = \vec{b} + \vec{w_x}$ for a vector $\vec{w_x} \in V$. Since $\vec{b} \in A$ we also have that $\vec{b} = \vec{a} + \vec{v_b}$. Substituting $\vec{a} = \vec{b} - \vec{v_b}$ into $\vec{x} = \vec{a} + \vec{v_x}$ gives that $\vec{x} = \vec{a} + \vec{v_x} = \vec{b} - \vec{v_b} + \vec{v_x}$. Since V is a vector space we have that $\vec{w_x} = \vec{v_x} - \vec{v_b} \in V$. We just proved that $A \subset \vec{b} + V$. The reverse inclusion comes from the fact that $\vec{b} + V$ is an affine subspace and $a \in \vec{b} + V$ and. The proof is symmetric. □

Definition A.5 (Null-space of a matrix)**.** *Let T be a field, $A \in T^{m \times n}$ be a matrix with $n, m \in \mathbb{N}$. The set $\{\vec{x} \in T^n \mid A\vec{x} = \vec{0}\}$ is called the* null-space *of the matrix A.*

There is a one-to-one relationship between a matrix over a field T and a linear transformation between arithmetic vector spaces over the same field. For every vector $\vec{x} \in T^n$ let $[\vec{x}]_\beta$ denote the coordinates of the vector $\vec{x}$ relative to the basis β of T^n. Consider two vector spaces T^m and T^n and their two fixed ordered bases $\beta = \{\vec{b}_1, \ldots, \vec{b}_n\}$ of T^n and γ of T^m. Let $\{\vec{c}_1, \ldots, \vec{c}_n\}$ be a subset of T^m. Then

every linear transformation $L : T^n \to T^m$ with $L(\vec{b}_i) = \vec{c}_i$ for every $i \in \{1, \ldots, n\}$ corresponds to a matrix $A \in T^{m \times n}$ such that

$$A = \begin{pmatrix} [\vec{c}_1]_\gamma & \ldots & [\vec{c}_n]_\gamma \end{pmatrix}.$$

For the matrix A we have that $A[\vec{x}]_\beta = [L(x)]_\gamma$ for every $\vec{x} \in T^n$. Every matrix A for two fixed bases thus gives a linear transformation, too.

Remark A.6. *Null-space of every matrix $A \in T^{m \times n}$ is a subspace of the vector space T^n.*

Proof. Proof is a straightforward verification of the three properties of vector subspaces. In the following assume that $\vec{x}, \vec{y} \in \mathcal{N}(A)$.

The zero vector is in $\mathcal{N}(A)$ because $A\vec{0} = \vec{0}$.

The sum $\vec{x} + \vec{y} \in \mathcal{N}(A)$,

$$A(\vec{x} + \vec{y}) = A\vec{x} + A\vec{y} = \vec{0} + \vec{0} = \vec{0}.$$

For every $t \in T$ we have that $t\vec{x} \in \mathcal{N}(A)$ because

$$At\vec{x} = tA\vec{x} = t\vec{0} = \vec{0}.$$

□

Definition A.7 (Orthogonal complement)**.** *Let V be a vector subspace of a vector space U and the operation $\langle \vec{u} \mid \vec{v} \rangle$ denote the scalar product of vectors $\vec{u}, \vec{v} \in V$.* Orthogonal complement *of subspace V, denoted by $V^\perp$, is defined as:*

$$V^\perp = \{\vec{u} \in U \mid \langle \vec{u} \mid \vec{v} \rangle = 0 \text{ for all } \vec{v} \in V\}.$$

Definition A.8 (Affine linear transformation)**.** *Let V, U be vector spaces, $V_0 \leq V$, $U_0 \leq U$ be their subspaces and $V_A = \vec{v} + V_0$ and $U_A = \vec{u} + U_0$ be affine subspaces of V and U with vectors $\vec{v} \in V$ and $\vec{u} \in U$. Function $T_A : V_A \to U_A$ is an* affine linear transformation *if there is a linear transformation $T_0 : V_0 \to U_0$ such that $T_A(\vec{x}) = \vec{u} + T_0(\vec{x} - \vec{v})$, $\vec{x} \in V_A$.*

Definition A.9 (Set of all affine linear maps)**.** *Let V, U be vector spaces and $V_0 \leq V$, $U_0 \leq U$ be their subspaces. Let $V_A = \vec{v} + V_0$ and $U_A = \vec{u} + U_0$ be affine subspaces of V and U respectively with vectors $\vec{v} \in V$ and $\vec{u} \in U$.* Set of all affine linear mappings *between affine spaces V_A and U_A, $LT_A(V_A, U_A)$, is defined as:*

$$LT_A(V_A, U_A) = \{T : V_A \to U_A \mid T \text{ is an affine linear transformation}\}.$$

Lemma A.10. *Let $T : \mathbb{Z}_2^f \to \mathbb{Z}_2^b$ for $f \geq b$ be an onto linear map and $\vec{y} \in \mathbb{Z}_2^b$. Then $T^{-1}(\vec{y})$ is an affine subspace of $\mathbb{Z}_2^f$ and $|T^{-1}(\vec{y})| = 2^{f-b}$.*

Proof. By Remark A.6, $T^{-}1(0)$ is a vector subspace of $\mathbb{Z}_2^f$. The set $T^{-1}(\vec{y})$ is an affine subspace of the vector space $\mathbb{Z}_2^f$ since for every vector $\vec{u} \in T^{-1}(\vec{y})$ we have that $T^{-1}(\vec{y}) = \vec{u} + T^{-1}(\vec{0})$. First we show, $\vec{u} + T^{-1}(\vec{0}) \subseteq T^{-1}(\vec{y})$:

$$\begin{aligned} \vec{v} \in T^{-1}(\vec{0}) &\Rightarrow T(\vec{u} + \vec{v}) = T(\vec{u}) + T(\vec{v}) = \vec{y} + \vec{0} = \vec{y} \\ &\Rightarrow \vec{u} + T^{-1}(\vec{0}) \subseteq T^{-1}(\vec{y}). \end{aligned}$$

The reverse inclusion, $T^{-1}(\vec{y}) \subseteq \vec{u} + T^{-1}(\vec{0})$, holds as well since

$$\begin{aligned} \vec{v} \in T^{-1}(\vec{y}) &\Rightarrow T(\vec{v} - \vec{u}) = T(\vec{v}) - T(\vec{u}) = \vec{y} - \vec{y} = \vec{0} \\ &\Rightarrow \vec{v} - \vec{u} \in T^{-1}(\vec{0}) \\ &\Rightarrow T^{-1}(\vec{y}) \subseteq \vec{u} + T^{-1}(\vec{0}). \end{aligned}$$

Now we use this fact to prove the second statement of the lemma, $|T^{-1}(\vec{y})| = 2^{f-b}$. Fix arbitrary vector $\vec{u} \in T^{-1}(\vec{y})$. From Lemma A.4 then follows $T^{-1}(\vec{y}) = \vec{u} + T^{-1}(\vec{0})$. In addition, the equality describes a one-to-one map between $T^{-1}(\vec{0})$ and $T^{-1}(\vec{y})$. The consequence of is that all sets $T^{-1}(\vec{y})$ for every $y \in \mathbb{Z}_2^b$ have the same size, exactly $|T^{-1}(\vec{y})| = 2^{f-b}$. □

Lemma A.11. *Let $A \in \mathbb{Z}_2^{m \times n}$ be a matrix such that* $\operatorname{rank}(A) = m$ *and $m \leq n$. Then the system of linear equations $A\vec{x} = \vec{y}$ has 2^{n-m} solutions for every vector $\vec{y} \in \mathbb{Z}_2^m$.*

Proof. This is a consequence of Lemma A.10. We apply it for the linear transformation given by the matrix A and vector $\vec{y} \in \mathbb{Z}_2^m$.

□

Appendix B

Facts Regarding Probability

In the appendix we summarise some facts regarding the probability theory. We show the basic definitions and prove the statements used in the work.

First we discuss the discrete probability and discrete random variables. Then we move to continuous random variables. Finally we show the Markov's and the Chebyshev's inequality.

Whenever we perform a *random experiment* we get a random outcome – a *random event* occurred. The probability theory models the uncertainty included in random experiments and deals with the probabilities of the random events. For example if the coin toss is the random experiment, then the set $\{\text{roll}, \text{die}\}$ contains all the possible outcomes. If the coin is fair, they share the same probability of occurrence.

Definition B.1 (Probability space, elementary event)**.** *We refer to the set Ω consisting of all the possible outcomes of a random experiment, $\Omega = \{\omega_1, \ldots, \omega_n\}$, as to the* probability space*. Its elements ω_i, $i \in \{1, \ldots, n\}$, are the* elementary events*. The probability of every elementary event ω_i, $i \in \{1, \ldots, n\}$, equals p_i, $0 \leq p_i \leq 1$, and is written as:*

$$\mathbf{Pr}(\omega_i) = p_i.$$

The sum of the probabilities of the elementary events must be equal to one:

$$\sum_{i=1}^{n} p_i = 1.$$

In general the probabilities of the elementary events need not to be the same. However, this assumption is frequently satisfied and then $p_i = \frac{1}{n}$ for $i = 1, \ldots, n$.

Definition B.2 (Event, probability)**.** *The set $A = \{a_1, \ldots, a_m\} \subseteq \Omega$, containing only elementary events, is called a* compound event*. The* probability of the compound event A *is*

$$\mathbf{Pr}(A) = \sum_{\omega \in A} \mathbf{Pr}(\omega).$$

If the probabilities of the elementary events are the same, then

$$\mathbf{Pr}(A) = \frac{|A|}{|\Omega|} = \frac{m}{n}.$$

Definition B.3 (Complementary event, the certain event, the impossible event)**.** *Let $A \subseteq \Omega$ be an event. The event $\Omega - A$ is the* complementary event *of the event A. The* certain event *is the probability space Ω and its complementary event $\emptyset$ is the* impossible event*.*

Corollary B.4 (Probability of the complementary event)**.** *If $A \subseteq \Omega$ is an event then the probability of the complementary event is*

$$\mathbf{Pr}(\Omega - A) = 1 - \mathbf{Pr}(A).$$

Proof. Follows from the definition of probability:

$$\begin{aligned} 1 = \mathbf{Pr}(\Omega) &= \mathbf{Pr}((\Omega - A) \cup A) \\ &= \sum_{\omega \in \Omega - A} \mathbf{Pr}(\omega) + \sum_{\omega \in A} \mathbf{Pr}(\omega) \\ &= \mathbf{Pr}(\Omega - A) + \mathbf{Pr}(A). \end{aligned}$$

□

We can compute the size of the set, consisting of elementary events, if we know the event's probability and the size of the probability space.

Lemma B.5. *Let $A \subseteq \Omega$ be an event such that $\mathbf{Pr}(A) = p$. If the probabilities of the elementary events are the same, then $|A| = p|\Omega|$.*

Proof. This statement is a direct use of the definition of the probability of the event A.

$$p = \mathbf{Pr}(A) = \frac{|A|}{|\Omega|}$$

□

Definition B.6 (Intersection)**.** *Let $A, B \subseteq \Omega$ be events. The* event that both A and B occur *is denoted by A, B and equals the event $A \cap B$. The probability of the event A, B is*

$$\mathbf{Pr}(A, B) = \mathbf{Pr}(A \cap B).$$

Definition B.7 (Disjoint events)**.** *Let $A, B \subseteq \Omega$ be events. Events A and B are* disjoint *if $A \cap B = \emptyset$. The corresponding compound probability then equals*

$$\mathbf{Pr}(A, B) = \mathbf{Pr}(\emptyset) = 0.$$

Definition B.8 (Independent events)**.** *Let $A, B \subseteq \Omega$ be events. Events A and B are* independent *if*

$$\mathbf{Pr}(A, B) = \mathbf{Pr}(A)\,\mathbf{Pr}(B).$$

The motivation of the following definition is a way how to compute the probability of event A inside the restricted probability space B.

Definition B.9 (Conditional probability)**.** *Let $A, B \subseteq \Omega$ be events and assume that $\mathbf{Pr}(B) \neq 0$. Then the* conditional probability *of event $A \mid B$ is defined as*

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A, B)}{\mathbf{Pr}(B)}.$$

Lemma B.10. *If $A, B \subseteq \Omega$ are non-disjoint events such that $\mathbf{Pr}(B) \neq 0$, then $\mathbf{Pr}(B) \leq \frac{\mathbf{Pr}(A)}{\mathbf{Pr}(A|B)}$.*

Proof. From our assumptions we have that the conditional probability of the event $A \mid B$ is defined and positive. To prove the lemma the definition of the conditional probability and the straightforward bound on the compound probability, $\mathbf{Pr}(A, B) \leq \mathbf{Pr}(A)$, is used

$$\mathbf{Pr}(B) = \frac{\mathbf{Pr}(A, B)}{\mathbf{Pr}(A \mid B)} \leq \frac{\mathbf{Pr}(A)}{\mathbf{Pr}(A \mid B)}.$$

□

Remark B.11. *Let $A, B \subseteq \Omega$ be events with $B \neq \emptyset$. The events are independent if and only if $\mathbf{Pr}(A \mid B) = \mathbf{Pr}(A)$.*

Proof. If we assume the independence of the two events we have

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A, B)}{\mathbf{Pr}(B)} = \frac{\mathbf{Pr}(A)\,\mathbf{Pr}(B)}{\mathbf{Pr}(B)} = \mathbf{Pr}(A).$$

The reverse implication holds since

$$\mathbf{Pr}(A, B) = \mathbf{Pr}(A \mid B)\,\mathbf{Pr}(B) = \mathbf{Pr}(A)\,\mathbf{Pr}(B).$$

□

Lemma B.12. *Properties of Probability*

(1) $\mathbf{Pr}(\emptyset) = 0$

(2) $\mathbf{Pr}(\Omega) = 1$

(3) $0 \leq \mathbf{Pr}(A) \leq 1$

(4) $\mathbf{Pr}(A \cup B) = \mathbf{Pr}(A) + \mathbf{Pr}(B) - \mathbf{Pr}(A \cap B)$

Theorem B.13 (The Law of Total Probability)**.** *Let $A \subseteq \Omega$ be an event and $\Omega_1, \ldots, \Omega_k$ be a decomposition of the probability space Ω such that it does not contain the impossible event. Then*

$$\mathbf{Pr}(A) = \sum_{i=1}^{k} \mathbf{Pr}(A \mid \Omega_i)\,\mathbf{Pr}(\Omega_i).$$

Proof. We perform a simple computation to decompose the event A. The decomposed event is then rewritten using the definition of the conditional probability.

$$\begin{aligned}\mathbf{Pr}(A) &= \mathbf{Pr}(A \cap \Omega) \\ &= \sum_{i=1}^{k} \mathbf{Pr}(A, \Omega_i) \\ &= \sum_{i=1}^{k} \mathbf{Pr}(A \mid \Omega_i)\, \mathbf{Pr}(\Omega_i)\end{aligned}$$

□

Definition B.14 (Discrete random variable)**.** *Let Ω be a probability space. Then* discrete random variable X *is a function $X : \Omega \to \mathbb{N}$. Probability of the event $X = x$ equals*

$$\mathbf{Pr}(X = x) = \sum_{\substack{\omega \in \Omega \\ X(\omega) = x}} \mathbf{Pr}(\omega).$$

In the continuous probability we usually assume that the vector space Ω is an uncountable set.

Definition B.15 (Continuous random variable, probability density function, cumulative probability density function)**.** Continuous random variable X *is a function $X : \Omega \to \mathbb{R}$.*

Function $F : \mathbb{R} \to [0, 1]$ is the cumulative probability density function *of the random variable X if the following conditions are satisfied*

(1) $\lim_{t \to -\infty} F(t) = 0$,

(2) $\lim_{t \to \infty} F(t) = 1$.

(3) The function F is non-decreasing.

(4) $F(t) = \mathbf{Pr}(X \leq t)$ for every $t \in \mathbb{R}$.

Function $f : \mathbb{R} \to \mathbb{R}^*$ such that $f(t) = F'(t)$ for every $t \in \mathbb{R}$ is the probability density function *of the random variable X. The probability density function f satisfies that*

(1) $\int_{-\infty}^{\infty} f(t)\, dt = 1$,

(2) $\int_{-\infty}^{t} f(x)\, dx = F(t)$.

More formal definitions of random variables use the Lebesgue measure and Borel sets to measure the probability $\mathbf{Pr}(X = x)$. However previous definitions are sufficient for our needs.

If X is a continuous random variable with the cumulative density function F then the probability of $X \in [a, b]$ equals

$$\mathbf{Pr}(a \leq X \leq b) = F(b) - F(a) = \int_a^b f(x)\, dx.$$

The motivation for the definition of the probability density function is the fact that it corresponds to the probability of X being equal to a singleton. We see that for every $\epsilon > 0$ value $\epsilon f(b)$ approximates $\mathbf{Pr}(b - \epsilon \leq X \leq b)$, if it is small enough. However for continuous variables the event $X = b$ is impossible unless F is in-continuous in b and thus $f(b) = \infty$.

After observing the previous fact we can rewrite $F(t)$ to the form:

$$F(t) = \mathbf{Pr}(X \leq t) = \int_{-\infty}^{t} \mathbf{Pr}(X = t)\, dt.$$

Definition B.16 (Expected value)**.** *Let X be a discrete random variable. Its* expected value*, $\mathbf{E}(X)$, is defined as*

$$\mathbf{E}(X) = \sum_{x \in \mathbb{N}} x \mathbf{Pr}(X = x).$$

In the continuous case, assume that X is a continuous random variable, then its expected value*, $\mathbf{E}(X)$, is defined as*

$$\mathbf{E}(X) = \int_{-\infty}^{\infty} x \mathbf{Pr}(X = x)\, dx.$$

Definition B.17 (Variance)**.** *Let X be a random variable. Its variance, $\mathbf{Var}(X)$, is defined as*

$$\mathbf{Var}(X) = \mathbf{E}\left((X - \mathbf{E}(X))^2\right).$$

We only show the basic properties of the expected value and variance without the proof.

Lemma B.18. *Assume that X and Y are random variables and $a, b, c \in \mathbb{R}$. Then*

(1) $\mathbf{E}(aX + bY + c) = a\mathbf{E}(X) + b\mathbf{E}(Y) + c$

(2) $\mathbf{Var}(aX + bY + c) = a^2\mathbf{Var}(X) + b^2\mathbf{Var}(Y)$

Definition B.19 (Expected value of a function of a random variable)**.** *Let X be a continuous random variable and $g : \mathbb{R} \to \mathbb{R}$ be a function. The* expected value *of the random variable $Y = g(X)$ is defined as*

$$\mathbf{E}(Y) = \int_{-\infty}^{\infty} y\mathbf{Pr}(Y = y)\, dy = \int_{-\infty}^{\infty} g(x)\mathbf{Pr}(X = x)\, dx.$$

Lemma B.20. *Let X, Y be random variables and $t \in \mathbb{R}$. For every $z \in [0, 1]$ set $\omega_t(z) = \{y \in \mathbb{R} \mid z = \mathbf{Pr}(X = t \mid Y = y)\}$. Let Z_t be a random variable bounded in the interval $[0, 1]$ such that*

$$\mathbf{Pr}(Z_t = z) = \int_{\omega_t(z)} \mathbf{Pr}(Y = y)\, dy.$$

Then $\mathbf{Pr}(X = t) = \mathbf{E}(Z_t)$.

Proof. By the Law of Total Probability we state the following fact:

$$\begin{aligned}
\mathbf{Pr}(X = t) &= \int_{-\infty}^{\infty} \mathbf{Pr}(X = t, Y = y)\, dy \\
&= \int_{-\infty}^{\infty} \mathbf{Pr}(X = t \mid Y = y)\, \mathbf{Pr}(Y = y)\, dy \\
&= \int_{0}^{1} z\mathbf{Pr}(Z_t = z)\, dz \\
&= \mathbf{E}(Z_t).
\end{aligned}$$

So the expected value of the variable Z equals the probability $\mathbf{Pr}(X = t)$. □

In Lemma 5.10 we refer to the modification of Lemma B.20 for $X \geq t$. In this modified version we assume that variable $Z = \mathbf{Pr}(X \geq t \mid Y = y)$. A straightforward inspection proves the consequence.

The following two theorems, the Markov's [31] and Chebyshev's [15] inequalities, are well known and we often refer to them in the work. Let us note that various improvements for the higher moments hold as well.

Theorem B.21 (Markov's inequality)**.** *Let X be a random variable and $t \in \mathbb{R}^+$. Then the probability of the event $X \geq t$ is bounded as*

$$\mathbf{Pr}(|X| \geq t) \leq \frac{\mathbf{E}(|X|)}{t}.$$

Proof. For the indicator $\mathbf{I}(|X| \geq t)$ we have that $t\mathbf{I}(|X| \geq t) \leq |X|$. If $|X| \geq t$, than $\mathbf{I}(|X| \geq t) = 1$ and it follows that $t\mathbf{I}(|X| \geq t) = t \leq |X|$. If $|X| < t$, then $0 = \mathbf{I}(|X| \geq t) \leq |X|$.

$$\begin{aligned}
\mathbf{E}(|X|) &= \int_0^\infty |X| \mathbf{Pr}(|X| = x)\, dx \\
&\geq \int_0^\infty t\mathbf{I}(|X| \geq t) \mathbf{Pr}(|X| = x)\, dx \\
&= t \int_0^\infty \mathbf{I}(|X| \geq t) \mathbf{Pr}(|X| = x)\, dx \\
&= t\mathbf{Pr}(|X| \geq t).
\end{aligned}$$

□

Theorem B.22 (Chebyshev's inequality)**.** *If X is a random variable, then*

$$\mathbf{Pr}(|X - \mathbf{E}(X)| \geq \epsilon) \leq \frac{\mathbf{Var}(X)}{\epsilon^2}.$$

Proof. First observe that the event $|X - \mathbf{E}(X)| \geq \epsilon$ is equivalent to the event $(X - \mathbf{E}(X))^2 \geq \epsilon^2$. The theorem follows from the Markov's inequality used for the latter event and from the definition of variance.

$$\begin{aligned}
\mathbf{Pr}(|X - \mathbf{E}(X)| \geq \epsilon) &= \mathbf{Pr}\left((X - \mathbf{E}(X))^2 \geq \epsilon^2\right) \\
&\leq \frac{\mathbf{E}\left((X - \mathbf{E}(X))^2\right)}{\epsilon^2} \\
&= \frac{\mathbf{Var}(X)}{\epsilon^2}.
\end{aligned}$$

□

From the Chebyshev's inequality it follows that the average converges to the expected value. This fact is commonly referred to as the Weak Law of Large Numbers.

Theorem B.23 (Weak Law of Large Numbers)**.** *Let $n \in \mathbb{N}$, $\epsilon > 0$ and $X_1, \ldots, X_n$ be independent identically distributed random variables. Then*

$$\mathbf{Pr}\left(\left|\frac{\sum_{i=1}^n X_i}{n} - \mathbf{E}(X_1)\right| \geq \epsilon\right) \leq \frac{\mathbf{Var}(X_1)}{n\epsilon^2}.$$

Proof. Define the average of $X_1, \ldots, X_n$ as the random variable $\bar{X} = \frac{\sum_{i=1}^n X_i}{n}$. From the properties of the expected value and variance stated in Lemma B.18 it follows that

$$\begin{aligned}
\mathbf{E}\left(\bar{X}\right) &= \mathbf{E}(X_1) \\
\mathbf{Var}\left(\bar{X}\right) &= \frac{\mathbf{Var}(X_1)}{n}.
\end{aligned}$$

The Chebyshev's inequality used for the random variable $\bar{X}$ yields the required result. □

Following lemma enables us to compute the expected value of a continuous random variable if we know its cumulative probability density function.

Lemma B.24. *Let X be a continuous random variable taking only non-negative values. Let $F : \mathbb{R}_0^+ \to [0,1]$ be its cumulative probability density function. Then*

$$\mathbf{E}(X) = \int_0^\infty 1 - F(x)\ dx$$

Proof. From the definition of the expected value and the cumulative probability density function we have that

$$\begin{aligned}
\mathbf{E}(X) &= \int_0^\infty t\mathbf{Pr}(X = t)\, dt \\
&= \int_0^\infty \int_0^t \mathbf{Pr}(X = t)\, dx\ dt \\
&= \int_0^\infty \int_x^\infty \mathbf{Pr}(X = t)\, dt\ dx \\
&= \int_0^\infty \mathbf{Pr}(X \geq x)\, dx \\
&= \int_0^\infty 1 - F(x)\ dx.
\end{aligned}$$

□