# Developing Mathematical Structures to Describe and Implement Binary Key and Lock Systems

Hillevi Nilsson
hillevinilsson98@gmail.com

under the direction of
Prof. Henrik Eriksson
Department of Computer Science and Communications
KTH Royal Institute of Technology

Research Academy for Young Scientists
July 13, 2016

**Abstract**

The aim of this paper is to develop a mathematical structure with which one can describe and construct any system of binary keys and locks. This will be achieved by first compressing an already existing so called Phillevi-diagram describing the relations between the locks and keys. The compressed version will then be embedded in a boolean lattice, giving the codes for the design of the locks and key. These codes are then translated into specified features constituting the actual keys and locks. The final product is a system of binary keys and locks that can easily be manufactured.

# Contents

# 1 Introduction

Keys and locks have been used for thousands of years and we still use them daily. There are different designs but they share the same objective; a lock keeps everyone out except for a few people possessing the right keys. Often there are several different locks and keys connected to each other in a certain system.

Apartment buildings often use a key and lock system (KAL system), similar to Example 1:

**Example 1.** *In a certain apartment building, seen in Figure 1, there are three different kinds of locks: on the front door, on the two apartment doors and on the door to the laundry room.*

*There are also three different types of keys for different people: the landlord owns a master key that gives access to all doors, the postman's key only opens the front door and the two tenants' keys give access to the front door, the laundry room and their own apartments.*

*The more locks a key can open, the more powerful it is. As the landlord has access to all the doors, his key is the most powerful one. The tenants' keys has access to some, but not all doors and is therefore less powerful than the landlord's. The postman's key can only open the front door which all the other keys also can open and is therefore the least powerful. The power structure of locks follows a similar system; the fewer keys that can open a lock, the more powerful that lock is.*

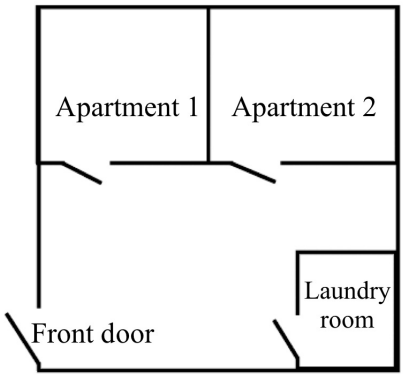



Figure 1: An apartment building with two apartments, one front door and a laundry room. The two tenants each have access to three of the four doors, the postman only to the front door and the landlord to all doors.

Some keys open more than one lock and consequently some locks can be opened by a greater number of keys. The keys used in this study have a design based on a binary system. Every key has several positions in a row that can either be a peak (↑) or a valley (↓), as seen in Figure 2.

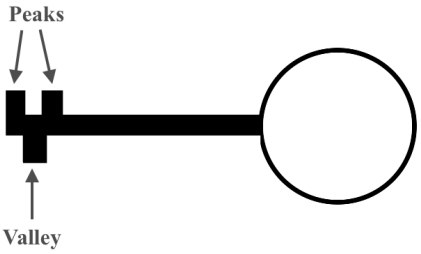



Figure 2: Each position of a key is either a peak or a valley.

While a key position has two options, each position on the locks has one of three different requirements; either it needs a peak to open ($NP$), it needs a valley to open ($NV$), or it accepts both ($AB$).

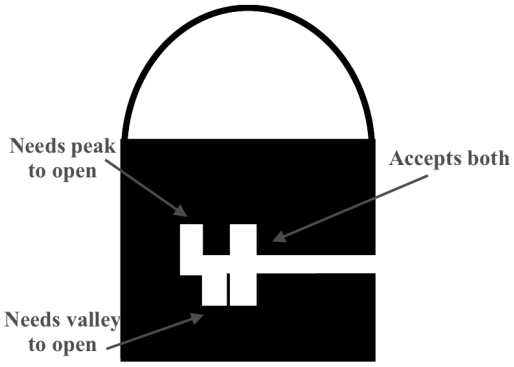



Figure 3: Each position of a key either requires a peak, requires a valley or accepts both.

Of course a lock can have several AB positions. A lock with more AB positions can be opened by more keys and the lock is therefore less powerful. The lock in Figure 3 has one AB position and can therefore be opened by two keys; the ones represented in Figures 2 and 4. The aim of this paper is to develop a mathematical structure which describes

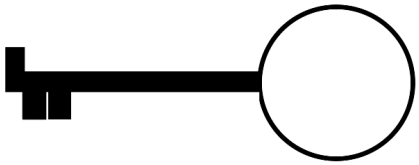


Figure 4: Another key that also opens the lock in Figure 3.

a system for such binary keys and locks as introduced above. When this structure is established, our next interest lies in investigating how a given system can be implemented: If there is a given power structure for keys and locks, how does one design actual locks and keys with peaks and valleys? How many available positions are needed for a specific system of keys and locks?

By constructing a general algorithm, large amounts of KAL systems can be produced. A system can be custom made and by keeping track of existing keys, there is no risk of using the same keys as in someone else's system. People will always have a need to lock things up and this would make it easier for every party: People can make their own KAL systems designed to meet their needs and those manufacturing the locks and keys can streamline their production if they have a system for quickly designing any possible KAL system.

# 2 Access Matrices and Phillevi Diagrams

We will follow the line of thought presented by P. Thiede in [1]. The KAL system is described with a binary matrix (a so called access matrix), as in Table 1. The matrix shows which locks every key opens.

Table 1: Access matrix describing whether the keys needed for Example 1 opens the lock (1) or not (0). The master key is represented by $K_1$, $K_2$ and $K_3$ are the tenants' keys and $K_4$ belongs to the post man. The laundry room is guarded by $L_1$, the apartments by $L_2$ and $L_3$ and the front door by $L_4$.

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|
| $K_1$ | 1 | 1 | 1 | 1 |
| $K_2$ | 1 | 1 | 0 | 1 |
| $K_3$ | 1 | 0 | 1 | 1 |
| $K_4$ | 0 | 0 | 0 | 1 |

The study also introduced the Phillevi diagram illustrated in Figure 5, to describe KAL systems [1].

**Definition 1.** *A* Phillevi diagram *consists of keys (•) and locks (∘) connected by lines, giving the diagram a power structure. Every node is either a key or a lock and its position in the diagram determines its power. A node is more powerful than another node if there is a descending chain from it to the other node.*

A line can never be drawn horizontally, which means that two nodes connected by a line are never equally powerful. The node above the other is always more powerful than the one below.

A key more powerful than a lock, opens that lock.

A key more powerful than another key opens all locks that the less powerful key opens.

A lock is more powerful than another lock if all keys that open it also opens the less powerful lock. A lock is more powerful than a key if all keys that open it are more powerful than that key.

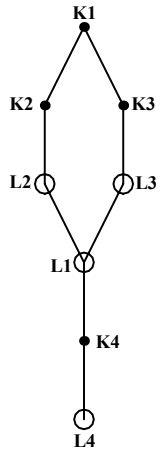



Figure 5: Example of a Phillevi diagram visualising the KAL system in Table 1 and Example 1.

The previous study that was earlier referred to, also proves that the Phillevi power structure represented in the Phillevi diagram is a partial order on the set of nodes and that it is a Hasse diagram for that partial order. We refer to [1] for this concept and for the following definition.

**Definition 2.** *A* partially ordered set *(or* poset*) is a set $P$ with a binary relation denoted $\leq$ that satisfies the following axioms.* [2]

1. For all $x \in P$, $x \leq x$ (reflexivity).
2. If $x \leq y$ and $y \leq x$, then $x = y$ (antisymmetry).
3. If $x \leq y$ and $y \leq z$, then $x \leq z$ (transitivity).

## 2.1 Compression of a Phillevi Diagram

We have seen that KAL systems can be described by an access matrix and more clearly with a Phillevi diagram. However, further improvements can be made by *compressing* the Phillevi diagram. The input we have is a Phillevi diagram with separate keys and locks as seen in Figure 6.

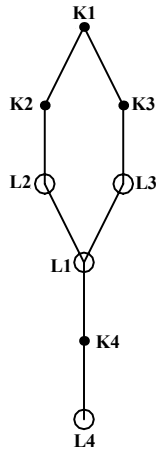



Figure 6: Phillevi diagram before compression.

Figure 6 can be compacted into a smaller Phillevi diagram if some of the keys and locks are compressed together, as seen in Figure 7. The reason for this is that keys and locks are often connected in pairs - and can therefore be designed together. For example, the key of a tenant is directly related to the lock of his apartment. No one else has a direct connection to that lock and therefore the lock and the key can be compressed into one key-lock (*klock*, $\odot$).

(a) Key and lock before compression. (b) Klock after compression.

Figure 7: Compression of a key and lock into a klock.

After compression the Phillevi diagram in Figure 6 takes the form of Figure 8.

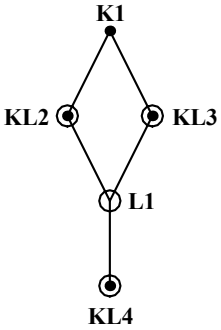



Figure 8: Phillevi diagram after compression.

The conditions for compressing a key and a lock into a klock are the following:

**Condition 1.** *The key needs to be right above the lock - the key and the lock are connected directly. The key needs to be above the lock, as compressing the two of them means that they get the the same node and are thus equally powerful. If a more powerful lock would be compressed with a less powerful key, the key would get access to that lock, breaking the original order. Therefore this only works for a more powerful key directly above a less powerful lock.*

**Condition 2.** *There can only be* one *key directly over the lock, otherwise the power structure is violated. If two keys open the same lock and the lock is compressed with one of them, the other key would become more powerful than the first key, which would result in a different power structure than before.*

If Conditions 1 and 2 are met, a key and a lock can be compressed together and make a klock. The klock does then have a direct connection to all the other keys and locks that the separate lock and key had, and the resulting compressed Phillevi diagram is equivalent to the original.

## 2.2 Compressed Phillevi Diagrams are Posets

We know that a Phillevi diagram is a poset [1]. It remains to show that this holds also for the compressed Phillevi diagram.

The axioms for a poset are the same as earlier presented in Definition 2:

1. For all $x \in P$, $x \leq x$ (reflexivity)
2. If $x \leq y$ and $y \leq x$, then $x = y$ (antisymmetry)
3. If $x \leq y$ and $y \leq z$, then $x \leq z$ (transitivity) [2]

**Proposition 1.** *The compressed Phillevi diagram where a key* k *and lock* l *are compressed into one node* kl *according to the Conditions 1 and 2 defines a poset.*

*The relations $x \leq y$ between the old, uncompressed nodes will remain.*

*Proof.* The Phillevi diagram defines the poset relation by $x \leq y$ iff there exists a path $x \leq x_1 \leq x_2 \leq x... \leq y$. Compression will only affect the path if some $x_i = l$. As the compression only takes place if $k$ is the only node directly above $l$ we must have $x_{i+1} = k$. The compression will lead to the direct link $l \leq k$ being replaced by the new node $kl$. The shortened chain proves that we still have $x \leq y$. Conversely, if $x \leq y$ after compression there is a chain $x \leq x_1 \leq x_2... \leq y$. If some $x_i = kl$, we can replace it with the direct link $l \leq k$, so there was a chain before compression and the relation $x \leq y$ was true also then. □

We now know that the compressed Phillevi diagram remains a poset. The next step is to describe the actual keys with the help of this system. This is easy if the Phillevi diagram can first be embedded in a boolean lattice.

## 2.3 Boolean Lattices

An example of a *boolean lattice* (for definition see [3]), is the power set $\mathcal{P}(S)$ of a set $S$, with $\subseteq$ as the binary relation replacing $\leq$.

This boolean lattice consists of nodes arranged in generations connected by lines. Every line represents a $\subseteq$ relation; i.e, the node below is a subset to the node above. The nodes in the same generation all contain the same number of elements. For every generation downwards, the nodes all contain one element less. At the top is the full set $S$ and at the bottom is the empty set $\varnothing$.

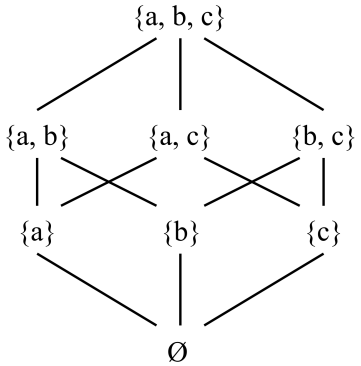



Figure 9: Example of a boolean lattice for the power set of a set with three elements.

Each set in the lattice is defined by which elements are missing compared to the full set. By representing presence of an element with a 1 and its absence with a 0, the boolean lattice of Figure 9 becomes the one in Figure 10. In the same way that nodes in the Phillevi diagram are more powerful than every connected node below, the same power structure can be seen in a boolean lattice. The more elements a node contains, the more powerful it is.

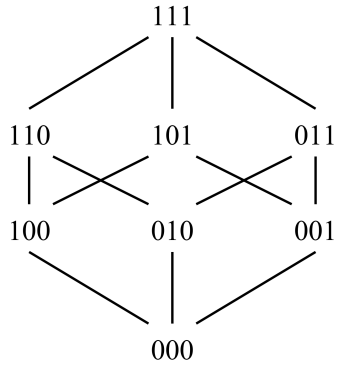



Figure 10: Boolean lattice from Figure 9 with sets replaced by series of ones and zeroes.

**Proposition 2.** *The structure of all boolean lattices follow the binomial distribution.*

*Proof.* Let us consider again the boolean lattice representing the power set $\mathcal{P}$ of a set $S$, where $S$ has $n$ elements and define $n$ as the *order* of the boolean lattice. All boolean lattices are ordered after their size - order zero is the smallest and the greater value, the greater the lattice. A boolean lattice of order $n$ contains a total number of $n$ possible elements. The top node (set) contains all $n$ elements. For every generation downwards in the boolean lattice, every node on the lower generation includes one element less than the nodes in the generation above, introducing a new zero in the binary representation. This is repeated for every generation until the empty set in the bottom generation is reached. The generations are numbered ($k$) starting with the top node called generation 0, down to the bottom node which is generation $n$. At any level $k$, exactly $k$ elements are included out of the total of $n$. There are $\binom{n}{k}$ ways to choose these, and thus $\binom{n}{k}$ sets in generation

$k$, which can be calculated by using the binomial distribution:

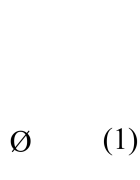
$$\binom{n}{k} = \frac{n!}{(n-k)!k!} \tag{1}$$

□

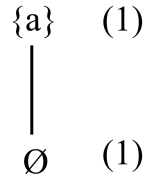
As a consequence all boolean lattices have a characteristic shape described by a row in Pascal's triangle (see Table 2). The first boolean lattice (order zero) consists of only one node, but the following ones are more complex. Every new lattice of higher order is described by the next row in Pascal's triangle.

Table 2: Pascal's triangle

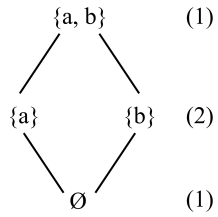
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 1 | | | | | | | | (0) |
| | | | | | 1 | | 1 | | | | | | | (1) |
| | | | | 1 | | 2 | | 1 | | | | | | (2) |
| | | | 1 | | 3 | | 3 | | 1 | | | | | (3) |
| | | 1 | | 4 | | 6 | | 4 | | 1 | | | | (4) |
| | 1 | | 5 | | 10 | | 10 | | 5 | | 1 | | | (5) |
| 1 | | 6 | | 15 | | 20 | | 15 | | 6 | | 1 | | (6) |
| | 7 | | 21 | | 35 | | 35 | | 21 | | 7 | | 1 | (7) |
| | | | | | | ... | | | | | | | | |



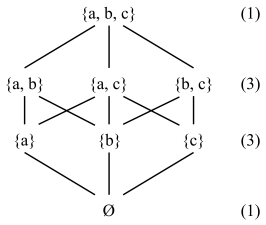

(a) Order 0




(b) Order 1




(c) Order 2




(d) Order 3

Figure 11: Boolean lattices of orders zero through three. Generations are labelled with number of nodes (cf. first four rows of Table 2).

The number of nodes in generation $k$ of a boolean lattice of order $n$ is the $k$:th number in the $n$:th row in Pascal's triangle. Thus we can always anticipate how every boolean lattice will be constructed. Given a boolean lattice of order $n$ we will know the following:

1. All boolean lattices are symmetric - nodes in the same generation have the same number of parents and children. Parents meaning the nodes above with lines connected down to our node and children meaning the nodes below which our node is connected to.

2. The top node always has $n$ children. In each new generation, nodes have one child less and the bottom node has no children. Nodes in generation $k$ thus have $n - k$ children.

3. The top node has no parents. In each new generation, nodes have one more parent and the bottom node has $n$ parents. Nodes in generation $k$ thus have $k$ parents.

## 2.4 Magnitude of a Phillevi Diagram

If we want to embed a Phillevi diagram in a boolean lattice, we need to know the minimum required size of the boolean lattice. This leads us to the following definition:

**Definition 3.** *The* magnitude *of a Phillevi diagram is defined as the minimum height of a boolean lattice in which the Phillevi diagram can be embedded.*

The magnitude can be found with the following procedure:
If there is no master key ($M_K$), a $M_K$ is added to the KAL system. Every key that earlier did not have a parent now has one from a direct connection to the master key. We know that the Phillevi diagram always contains exactly *one* master key (a key that opens all locks and is more powerful than all other keys). We also know that every connection between nodes is drawn in the Phillevi diagram. As $M_K$ is the most powerful key, we can be sure that there are no lines connecting a parent to $M_K$. If we begin at the $M_K$ and follow all lines downwards, all keys, locks and klocks will at some point be passed.

We start with $M_K$, following every line downwards and list all children that it is directly connected to, which constitutes the next generation. In th next generation the number of parents (they will by definition only come from the generation above, in this case the $M_K$) for every node is counted. Also the number of nodes on this generation is counted. The greatest number of parents and children one node in every generation has are noted. By continuing down to the next generation and repeating the steps $n$ times for diagrams with $n$ generations, the algorithm will always end with at least one node that has no children. Following this procedure we will know:

1. The number of generations, or the "height" $n$ of the Phillevi diagram.
2. The number of parents that entered the node with the most parents in every generation.
3. The greatest number of children for a node in a generation.
4. How many nodes each generation consists of.

These four steps will insure a complete charting of the Phillevi diagram.

## 2.5 Embedding a Poset In a Boolean Lattice

**Proposition 3.** *A Phillevi diagram can be embedded in a boolean lattice. If we scan every generation in the Phillevi diagram as described in section 2.4, we will be able to determine the size for the boolean lattice needed to embed a certain poset.*

*Proof.* We again start at generation zero with $M_K$ in the Phillevi diagram and investigate if it can fit the highest generation of the initiated boolean lattice. The remaining generations and their nodes are then examined to satisfy the following requirements:

1. The Phillevi diagram needs to have as many generations as the boolean lattice has, or less. If the Phillevi diagram is higher than the boolean lattice, all nodes can not be embedded in the boolean lattice.

2. There can never be a node with more parents or children than the corresponding generation in the boolean lattice.

3. There can always be fewer nodes in a generation in the Phillevi diagram than the boolean lattice but never the other way around, as in that case the diagram can not be embedded.

Starting with $M_K$ we continue to the next generation and repeat the steps in 2.5. If the diagram after $k$ generations no longer fits the boolean lattice that was earlier sufficient, the next order of boolean lattice is tried. A bigger boolean lattice always contains the ones that are smaller and therefore the first $k-1$ generations do not have to be tested again. This process is repeated for every generation until all nodes fit in the boolean lattice.

Do we know that the process will end? Yes, we do, as every partial order is embeddable in the boolean lattice of all subsets of the elements. □

## 2.6 Construction of Keys and Locks

Once in the boolean lattice, every node is treated as a klock. Therefore both a lock and a key can be made for every node. This makes it crucial to keep track of every key and lock to know which ones we really have to produce.

### 2.6.1 Construction of Keys

Assume we have embedded our keys and locks in a boolean lattice, meaning every node has been assigned a special code of ones and zeroes. To construct the keys, the ones and zeroes need to be translated into peaks (↑) and valleys (↓). The master key opens everything and therefore the less powerful a node is, the more it differs from the master key.

### 2.6.2 Master Key

From the boolean lattice, the form of every key can be determined. In the boolean lattice the key at the top (the master key) will consist of only ones. The physical key could then naïvely be given only peaks. The code for every key is determined by its position in the boolean lattice - the less powerful the key, the more zeroes its code contains. However, this would mean that if two different systems of keys uses the same size of a boolean lattice, they will use the same codes. This means that anyone who wants to forge a key and get access anywhere, can simply make a key with the right amount of positions (which should not be very difficult to find out) and fill it with peaks. With such a key one gets access to everything, which does not make sense for a safe key and lock system. Therefore, the keys will need to be translated into a new design.

The foundation of the naïve design is that the better any key resembles the master key, the more powerful it is. This will hold also for the new design; the only difference is that the master key will not consist of only peaks. Instead a sequence will be randomized and make the definition of the master key. The design of all the other keys will then follow this new master key. Every position on a key will be changed according to what the matching position says on the master key.

Table 3: Designing the keys from Example 1 with the help of codes extracted from the boolean lattice in Figure 10. A one means "the same as master key" (green), a zero means "the opposite to the master key" (red).

| Keys | Code | Naïve Design | General Design |
|---|---|---|---|
| Master key | 11 | ↑↑ | ↑↓ |
| Tenant 1 | 10 | ↑↓ | ↑↑ |
| Tenant 2 | 01 | ↓↑ | ↓↓ |
| Post man | 00 | ↓↓ | ↓↑ |

At this stage the total number of positions desired for the key is decided. If one wants the keys to have more positions than required that is not a problem. The superfluous positions are simply made the same for all keys in the system. It is possible to choose any positions on the key for those that are supposed to differ between keys and those being

the same for every key. If we want the previous example to have ten positions where only the two first positions change, it could look like Table 4:

Table 4: Four different keys where only the two first positions change. A ↑ means "the same as master key" (green), a ↓ means "the opposite to the master key" (red) and black means "the same on all keys".

| | |
|---|---|
| Master key | ↑↓↑↓↓↑↑↓↑↓ |
| Tenant 1 | ↑↑↑↓↓↑↑↓↑↓ |
| Tenant 2 | ↓↓↑↓↓↑↑↓↑↓ |
| Post man | ↓↑↑↓↓↑↑↓↑↓ |

The last part (↑↓↓↑↑↓↑↓) is the same for every key, but still makes the lock more difficult to pick. The crucial positions do not need to be in the beginning of the sequence, but can also be spread out. In the example presented in Table 4 the crucial positions might as well be chosen to be 3 and 9, not 1 and 2.

### 2.6.3 Construction of Locks

Recall that the locks have three different sorts of positions; needs peak to open (NP), needs valley to open (NV) or it accepts both (AB). As some locks are supposed to be opened by more than one key, AB positions need to exist. Otherwise only the one key that fits the lock exactly can open it (and that key can only open that one lock). The more powerful a lock is, the more positions will be specific and less AB positions exist.

From the boolean lattice we have already seen how the binary codes give the *keys* their appearance (in section 2.6.2). These same zeroes and ones also give the *locks* their design, but as we have three different positions in the locks the translation to actual positions on the locks will be different from the key construction.

Since we have redesigned the keys according to the master key, the same will be necessary for the locks. For keys, a one in the boolean lattice means "has the same as what is on the master key". When making the locks, a one will stand for "needs the same as what is on the master key". This can either be NV or NP, depending on what the master key has at its corresponding position. A zero in the boolean lattice will mean an

AB on the lock.

If we continue the example from section 2.6.2, this is how all the possible locks would have been designed:

Table 5: Designing the locks from Example 1 with the help of codes extracted from the boolean lattice in Figure 10. A one means "need the same as master key" (green), a zero means "accepts both" (red).

| Locks | Code | $M_K$ | Lock-design |
|---|---|---|---|
| Master key | 11 | $\uparrow\uparrow$ | NP NV |
| Apartment 1 | 10 | $\uparrow\downarrow$ | NP AB |
| Apartment 2 | 01 | $\downarrow\uparrow$ | AB NV |
| Front door | 00 | $\downarrow\downarrow$ | AB AB |

# 3 Conclusion

We have now undergone every step in the process of designing actual keys and locks from a access matrix and a Phillevi-diagram. It was proved that the Phillevi diagram can be embedded in a boolean lattice, which in turn gave us the codes for the keys and locks. These codes where then translated into actual designs for keys and locks. Together with the previous study [1] the result is a complete process for describing and implementing KAL systems, which then can be programmed making the whole process automated.

# 4 Future Research

With this system, one should be able to produce any kind of binary KAL system. However, this is limited to only binary keys and a possibility for the future would be to develop a system for more advanced keys. If the system is limited to a certain amount of possible keys and locks, where does that limit go? It would also be interesting to investigate how many different systems that could be produced if the size of the boolean lattice is decided beforehand. The system is binary and an interesting proceeding would be to make KAL systems for more complex keys and locks. Does a system with many different position

values make a safer system? If so, how unsafe is a binary system? How easy would it be for someone to pick one of these locks or to figure out the construction of the master key?

## Acknowledgements

I would like to thank my wonderful mentor Henrik Eriksson for his educational and fun guidance. Without him this would never have been possible and I am truly grateful for this opportunity. These two weeks would not have been the same without my co-worker Philip Thiede, who both made everything more enjoyable and laid the foundation for my study. My deepest thanks also go out to Rays and their partners AstraZeneca and Kjell och Märta Beijers stiftelse for making this whole experience possible. Thank you all.

# References


[1] Thiede P, *Representing Lock-and-Key Systems as Modified Hasse Diagrams* 2016 (in press)

[2] Stanley R.P, *Enumerative Combinatorics Volume 1.* 2nd edition. United States of America: Cambridge University Press; 2012.

[3] Rasiowa H, Sikorski R. *The Mathematics of Metamathematics.* Warzawa (Polen): Panstwowe Wydawnictwo Naukowe;1963