Lecture 1: Introduction

Welcome to the class on discrete math. Why do we offer such a course in our curriculum? The answer is easy. Many of the problems in software engineering, computer science, and information technology, are mathematical, and they need mathematical solutions. The trick of problem solving in these fields often requires us to take a step back, determine different ways of tackling a problem, and try one after the other until the problem is solved. The more we understand about the computational, mathematical and logical nature of those problems the better ideas we come up with to solve a problem and consequently the better our solutions are.

I suspect that this way of problem solving is not limited to problems in information technology. Biologists, astronomers, and economists actually need to tackle problems in very similar ways. Astronomers, for example, base lots of research on discovering information that is contained in databases. In order to retrieve information that must be able to describe their queries logically.

My personal goal for this class is that you don't only learn to understand and internalize the different mathematical definitions, such as injectivitiy, universal quantifiers or Abelian group. It is much more that you discover the excitement of solving problems, learn to apply some tools that I will show you in this class, how to learn more about your problem by experimenting with it.

Therefore let's get started. Here is a very small problem that we will study today to illustrate what you will learn in this class. Don't worry, I don't expect you to be able to provide all answers to the questions that I am going to ask, but it is a good start. Let's look at a little pebble game (that I found on Klaus Suttner's webpage). You have a certain number of pebbles, some white, some blue. Imagine that all pebbles form a line. You will traverse this line from left to right according to the two following rules:

1. If the current pebble is white, exchange it for a blue one, and skip the next two.
2. If the current pebble is blue, exchange it for a white one, and skip the next.

You apply the rules iteratively, until you hit the end of the line of pebbles. This defines one computational step.

Example 1 (One step)

So far so good. Let us reflect. I have given you a little puzzle, without asking you how to solve it. What do we know about it? This example, is limited in size, so, we can play with the number of pebbles. We can only play with the number of times we traverse the row of pebbles. We can also vary the coloring of the initial set of pebbles.

Example 2 (6 white pebbles, 30 steps)

Now we notice repetition. We note that every 9th line we encounter a sequence of 6 white pebbles. Interesting, there is some periodicy in here. But this was expected, right, we only have 6 pebbles. How many different *configurations* of 6 pebbles are there? There are $2^6 = 64$. One cycle is 8 steps long. And this is 2^3. And we will see that depending on the starting configuration, there are another "different" 8 cycles.

Example 3 (6 white pebbles, 8steps)

Do you notice anything about this pattern? Let's separate the columns into the odd columns and even columns.

Example 4 (6 white pebbles, 8steps, parity)
If we push these two columns together then we get the one from the previous example.

Odds

Evens

Now we notice, that the odds represent 3-bit binary numbers. Interesting. This means that every loop has a counter build in, completely automatically. All we have to do is project the even positions, and interpret the bits the right way to determine the running counter.

A quick reminder about binary numbers. Usually, we compute base 10. But in computer science, we tend to compute base 2, the reason being that in a physical processor, we can represent the two digits 0 and 1 as low and high voltage. We can convert any number from one number system to another. Here is a table with the first 2^4 numbers.

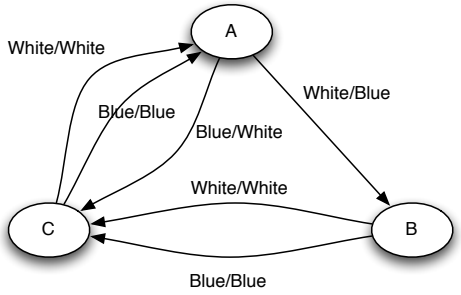
2^3	2^2	2^1	2^0	in decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Can you see the similarity between the odd columns and the binary numbers? Back to the example. We notice that the evens repeat themselves. The first four lines are like the second four lines. We don't know what that means, or where it comes from, but so be it. Is anyone ready for a conjecture at this point?

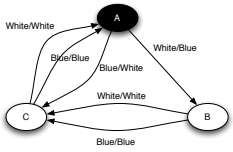
Let's start slow. Assume that we have n pebbles. How many different configurations of pebbles can we actually encounter? Every pebble can be either white or blue. There are only two possibilities. Two pebbles have four possible colorings. And then n pebbles have 2^n different colorings. Each coloring corresponds to a bit sequence of zeros and ones. Those things are also called *binary words*.

Furthermore, what does the flipping operation actually do? It takes a binary n bit word and maps it to the next n bit word.

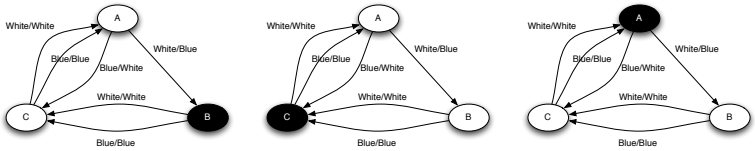
In addition, the length of the line pebbles remain invariant under the flipping operation. A line with n pebbles in, a line with n pebbles out. We say that the flipping operation is length preserving. How do we see that? To see this easily, we can write out the algorithm as an automaton.



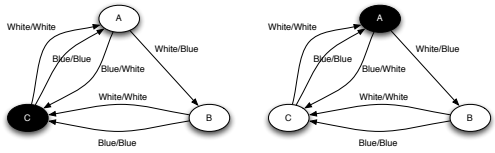
The nodes of this automaton, named A, B, and C are also called *states*. How can we possibly run that thing? Let's take as an example, the configuration of the pebbles ●●●●●, (line 3 in our example) above, and step trough. The labels at the edges of the automaton should be read as consume/produce. White/Blue means, consume a white pebble, produce a blue pebble. Initially, we are in step A.



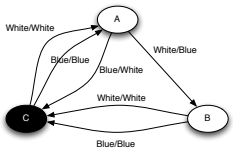
Let's look at the first three steps of the automaton when consuming the pebbles. The first three pebbles to consume are ●●●, white, white, and blue.



The first step consumes a white pebble and produces a blue, the second step consumes a white pebble and produces a white, and the third step consumes a blue pebble, and produces a white. We have already produced ●●● and we still need to consume ●●●. Let's consider the next two steps.



Next, we consume ● produce ● and consume ● produce ●. Altogether, we have produced already ●●●●●, and we still ned to consume the last pebble ●, and produce a ●. The automaton comes to a stand still at state C.



All is consumed, and ●●●●●● is produced.

Intuitively, it is clear that flipping preserves the length of our pebble configurations. Every step consumes a pebble and produces one. There is no way that extra pebbles are created during execution, and it is also clear that none are eaten up and destroyed. This argument is right and good, however, not formal enough yet to serve as a proof. A conjecture is a statement that we feel is true, but we haven't found a proof for it yet.

Conjecture 5 *The flipping operation is length preserving.*

Another interesting observation. The flipping operation is invertible. We understand what how the flipping operation is on concrete examples, which we could write as equations of the form

$$\begin{aligned} \text{flip}(\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \text{flip}(\bullet\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \text{flip}(\bullet\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \ldots \end{aligned}$$

If we abstract over it, it seems there is a mapping flip that maps *any* configuration of pebbles to some other configuration of pebbles. Do you think there is an inverse of flip? Let's write (as it is commonly done flip^{-1}) for that inverse, which on the three individual instances looks like:

$$\begin{aligned} \text{flip}^{-1}(\bullet\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \text{flip}^{-1}(\bullet\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \text{flip}^{-1}(\bullet\bullet\bullet\bullet\bullet\bullet) &= \bullet\bullet\bullet\bullet\bullet\bullet \\ \ldots \end{aligned}$$

Yes, it is, we could actually write out an automaton that computes that inverse. Just defining the automaton is unfortunately not enough. We need to

show that it is indeed the inverse. We could for example do this by considering the combination of flip and flip^{-1}, for example, is it true that flip^{-1}(flip(C)) $= C$ is true for any configuration C? We don't have the tools definitions yet to argue for this in the first lecture. For now, let's just state it as a conjecture.

Conjecture 6 *flip is invertible.*

And an even more complicated to prove conjecture that all cycles have the same length, that we know what it is depending on the number of pebbles we are considering, and that we can therefore compute the numbers of cycles.

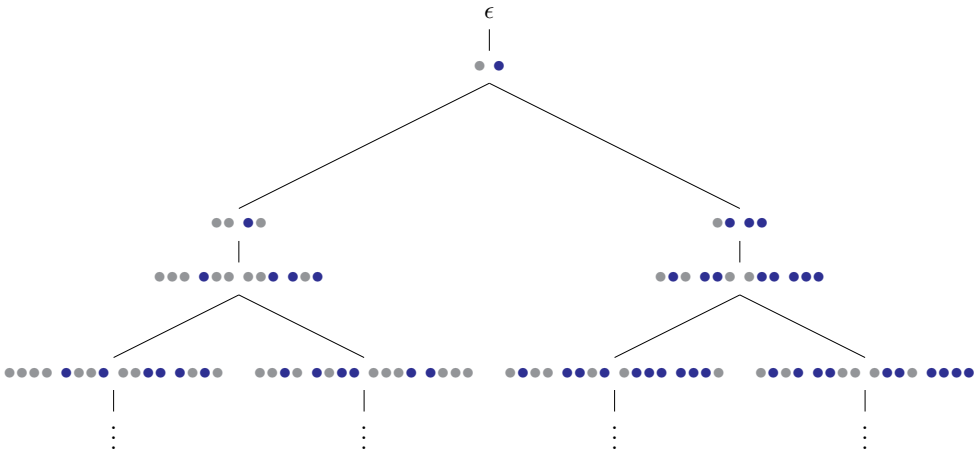
Conjecture 7 *There are* $2^{\lfloor k/2 \rfloor}$ *cycles on pebble configurations of length* k*. The length of each cycle is* $2^{\lceil k/2 \rceil}$*.*

We are not going to prove this conjecture here. Some conjectures are open for a long term, just like the four coloring theorem, Kepler's' conjecture, or Fermat's last theorem. All of these conjectures have an by other mathematicians accepted proof, which makes them theorems. Sometimes, we also use words like *lemma*, which is simply a theorem of lesser importance, or *corollary*, which usually is an *easy* to see consequence of a theorem or a lemma.

For our example, we can easily verify this. $k = 6$, means that $k/2$=3, and thus $\lfloor k/2 \rfloor = \lceil k/2 \rceil = 3$. Thus we have $2^3 = 8$ cycles, each of which is of length $2^3 = 8$ as well. Let's put it into a table, where we display word size on the vertical axis and cycle length on the horizontal.

w/c	1	2	4	8	16	32
0	1					
1		1				
2		2				
3			2			
4			4			
5				4		
6				8		
7					8	
8					16	
9						16
10						32

This table is rather interesting. We observe, for example, that for any cycle length there are only two rows that exhibit cycles of that length. Second, the numbers of powers of 2. On the other hand, this is not surprising. The conjecture above already tells us that the cycle lengths are powers of 2. Now, we can organize all this knowledge in form of a tree, which we call the cycle tree. We write ϵ for a the empty list of pebbles.



Also this tree has some interesting aspects to it that we may want to look into deeper. For example, its structure. Note that depending on the level a node is on, it either has one child node (below the original node, with one line attached to it) or two. A tree whose nodes have only one child are also called *lists* and those where all nodes have two children nodes are called *binary trees*. We will learn more about them in the lectures that follow.