Note 5

Recursion and induction

5.1 Examples of a recursively defined functions

In this section, we introduce the concept of a recursively defined function. The concept of a *recursion* in this context is simply to define a function or an expression using that function or expression itself for other input values. Let us start with an example:

Example 5.1

The *factorial* function $\texttt{fac} : \mathbb{N} \to \mathbb{N}$ is defined by $n \mapsto 1 \cdot 2 \cdots n$. Hence n is mapped to the product of the first n positive integers. It is also very common to write $n!$ instead of $\texttt{fac}(n)$. We have for example $\texttt{fac}(1) = 1$, $\texttt{fac}(2) = 1 \cdot 2 = 2$, $\texttt{fac}(3) = 1 \cdot 2 \cdot 3 = 6$, $\texttt{fac}(4) = 1 \cdot 2 \cdot 3 \cdot 4 = 24$, etcetera. Now note that, if we want to compute the next value, $\texttt{fac}(5)$, we can use that we already know what $\texttt{fac}(4)$ is. Indeed,

$$\texttt{fac}(5) = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = (1 \cdot 2 \cdot 3 \cdot 4) \cdot 5 = \texttt{fac}(4) \cdot 5 = 24 \cdot 5 = 120.$$

In general, if for some $n > 1$, we already have computed $\texttt{fac}(n-1)$, we can compute the value of $\texttt{fac}(n)$ using that $\texttt{fac}(n) = \texttt{fac}(n-1) \cdot n$. This leads to the following algorithmic description of the factorial function:

Algorithm 1 $\texttt{fac}(n)$

Input: $n \in \mathbb{Z}_{\geq 1}$.

1: **if** $n = 1$ **then**
2: **return** 1
3: **else**
4: **return** $\texttt{fac}(n-1) \cdot n$.

This algorithm simply uses itself to compute $\texttt{fac}(n)$. More precisely, if $n = 1$, it directly returns 1 as the value for $\texttt{fac}(1)$, as prescribed in line 2 of the algorithm. Graphically, we can illustrate this as follows:

$\texttt{fac}(1)$

$$\texttt{fac}(1) = 1$$

$\texttt{return}\ 1$

If $n = 2$, the algorithm will go to line 4 and attempt to return $\texttt{fac}(2-1) \cdot 2$. However, this requires that first the value of $\texttt{fac}(1)$ is computed. Hence the algorithm will then start over, but now for the value 1. We have already seen that the algorithm returns 1 in that case. Now that the algorithm has arrived at the conclusion that $\texttt{fac}(1) = 1$, it can revisit line 4 and compute that $\texttt{fac}(2) = \texttt{fac}(1) \cdot 2 = 1 \cdot 2 = 2$. Hence the algorithm returns 2. Graphically, the situation is:

$\texttt{fac}(2)$

$$\texttt{fac}(2) = \texttt{fac}(1) \cdot 2$$

$\texttt{fac}(1)$

$$\texttt{fac}(1) = 1$$

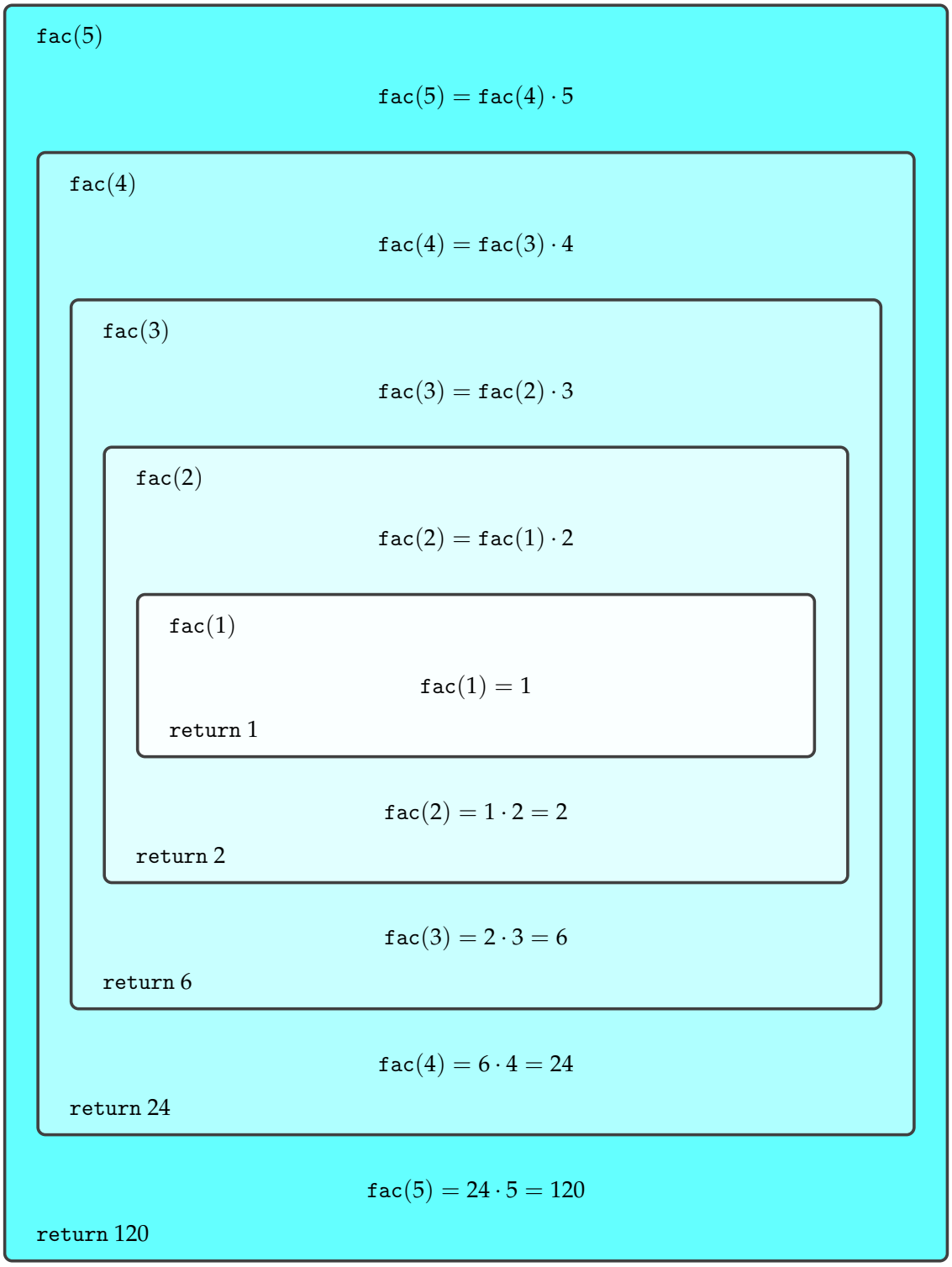
$\texttt{return}\ 1$

$$\texttt{fac}(2) = 1 \cdot 2 = 2$$

$\texttt{return}\ 2$

For larger values of n more "boxes inside other boxes" will appear, since the algorithm will need to use itself more often to compute its output for the smaller input values $n-1, n-$

$2, \ldots, 1$ before it can return its final output. For $n = 5$, the following graphical representation indicates what happens when this algorithm gets input value $n = 5$:



Since the algorithm uses itself while running (in algorithmic terms one often says that the algorithm *calls* itself), it is called a *recursive* algorithm. A recursive algorithm is simply an algorithm that might call itself for other input values in order to compute its final output value. Also in mathematics, recursions occur. In the context of this example, we have actually give a recursive definition of the factorial function:

$$\texttt{fac}(n) = \begin{cases} 1 & \text{if } n = 1, \\ \texttt{fac}(n-1) \cdot n & \text{if } n \geq 2. \end{cases} \tag{5-1}$$

What this example illustrates is the principle of a recursive definition: to define the values a function takes using that same function itself. Note by the way that it is also very common to define $0! = 1$, but that is another matter. Here is another example of a recursively defined function: Let $z \in \mathbb{C}$ be a complex number and define $f : \mathbb{N} \to \mathbb{C}$ recursively as:

$$f(n) = \begin{cases} z & \text{if } n = 1, \\ f(n-1) \cdot z & \text{if } n \geq 2. \end{cases} \tag{5-2}$$

Then $f(1) = z$, since this corresponds to the case $n = 1$ in the recursive definition. Further $f(2) = f(1) \cdot z$, since this is what the recursive definition gives for $n = 2$. Using that we already computed that $f(1) = z$, we may conclude that $f(2) = f(1) \cdot z = z \cdot z$. Finally using that $z \cdot z = z^2$, we see that $f(2) = f(1) \cdot z = z \cdot z = z^2$. Similarly, $f(3) = z^3$. Therefore it is perfectly reasonable to *define* the expression z^n for any natural number n recursively as $f(n)$. In previous chapters, we have used n-th powers of complex numbers several times. Now we have a more formal definition for it. In this light, it is also common to define $z^0 = 1$ and $z^{-n} = 1/z^n$ for any natural number n. This means that we now have defined very precisely what z^n means for any integer $n \in \mathbb{Z}$.

When attempting to define a function recursively, one should make sure afterwards that such a recursive description actually defines the function for all values from its domain. For the functions defined in equations (5-1) and (5-2) you can find a justification in Example 5.6, but feel free to skip that example on a first reading. For now, let us just show an example of a recursive description that does not work out. Let $g : \mathbb{N} \to \mathbb{C}$ be a function and suppose that

$$g(n) = \begin{cases} 1 & \text{if } n = 1, \\ g(n+1) & \text{if } n \geq 2. \end{cases}$$

By definition we see that $g(1) = 1$, but we do not have enough information to determine what $g(2)$ is. If we apply the recursive definition, we would just obtain that $g(2) =$

$g(3)$. Then attempting to compute $g(3)$, the recursion only yields that $g(3) = g(4)$. Continuing like this, we obtain that $g(2) = g(3) = g(4) = g(5) = \cdots$, but we never find out what $g(2)$ actually is.

As a final example of a recursive definition, we consider the famous Fibonacci numbers.

|||| Example 5.2

Let us now consider a recursive definition that looks slightly different. We are going to define recursively a function $F : \mathbb{N} \to \mathbb{N}$ whose values $F(1), F(2), F(3), F(4), \ldots$ are called the *Fibonacci numbers*:

$$F(n) = \begin{cases} 1 & \text{if } n = 1, \\ 1 & \text{if } n = 2, \\ F(n-1) + F(n-2) & \text{if } n \geq 3. \end{cases} \tag{5-3}$$

Let us see how this definition works in practice by computing the first Fibonacci numbers. First of all $F(1) = 1$, since if $n = 1$, the first line of equation (5-3) applies. If $n = 2$, the second line of equation (5-3) applies, so that $F(2) = 1$. For $n = 3$, the third line of equation (5-3) applies and we find that $F(3) = F(2) + F(1) = 1 + 1 = 2$. Similarly for $n = 4$, we find that $F(4) = F(3) + F(2) = 2 + 1 = 3$, using that we already have computed that $F(3) = 2$ before.

When dealing with a sequences of numbers, such as the Fibonacci numbers, it is quite common to change the notation a bit: instead of writing $F(n)$, one often writes F_n. In this notation we would get $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$ and so on. It turns out that it is possible to derive a closed formula expression for the Fibonacci numbers:

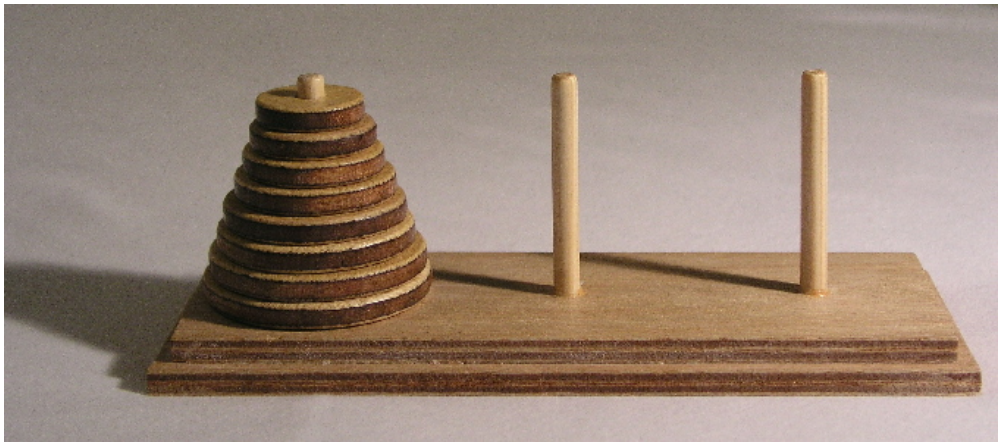
$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2} \right)^n . \tag{5-4}$$

We will come back to explaining how this expression comes about in a later chapter.

5.2 The towers of Hanoi

In this section, we further illustrate the usefulness of a recursive way of thinking when analyzing a puzzle called *the towers of Hanoi.* The towers of Hanoi is a puzzle on a board containing three upright sticks of equal lengths and sizes. Further there are various circular discs all of different diameter, each with a hole in the middle so they can be placed on a stick. In the starting position of the puzzle, all discs are stacked on the first stick. The disc with largest diameter is stacked first, the other discs in decreasing diameter size. The number of disks can vary. For an example with eight disks, see Figure 5.1.

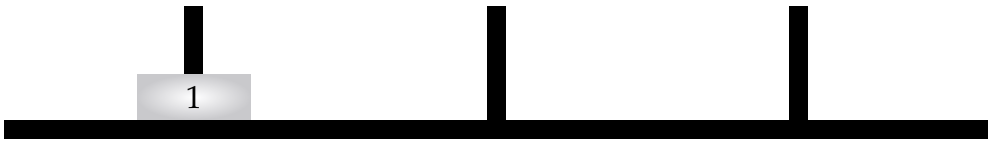
Figure 5.1: The tower of Hanoi with eight discs.



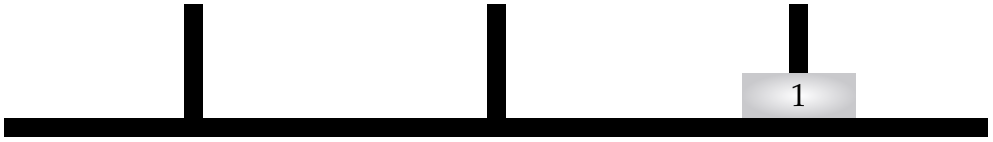
Now the goal of the puzzle is to move the stack of discs from the first to the third stick, stacked in the same way again from large to small. However, the challenge is that this has to be achieved following three rules:

- Only one disc may be moved at a time.
- Only a disc on top of a stack may be moved.
- A disc may only be placed on a larger disc.

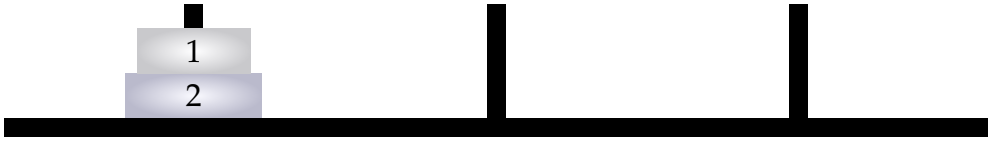
If there are only very few discs, it is not hard to solve the puzzle. If there are many discs, the game becomes more complicated and a priori it is not even clear if there always exists a solution. To get started, let us look at some examples with only a few discs. First of all, if there is only one disc, we can solve the puzzle in one move:



Move disc 1 from stick 1 to stick 3



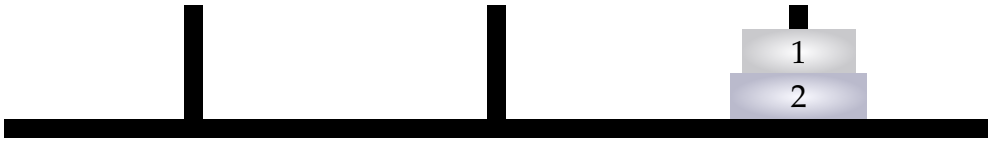
If there are two discs, the puzzle can be solved in three moves:



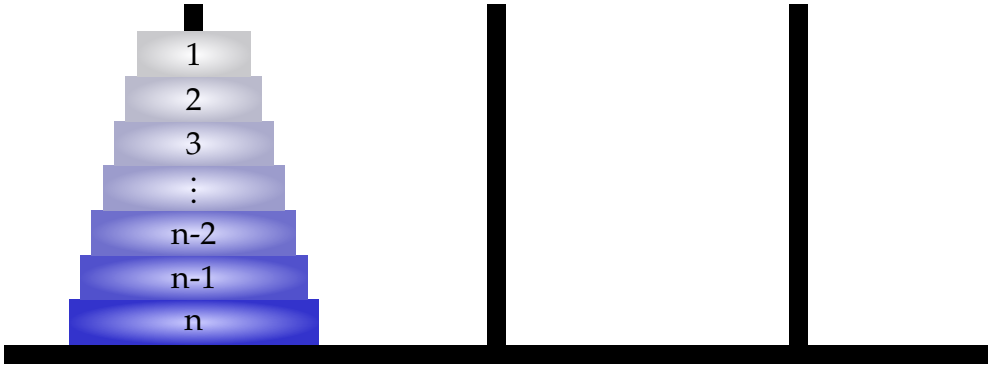
Move disc 1 from stick 1 to stick 2

Move disc 2 from stick 1 to stick 3

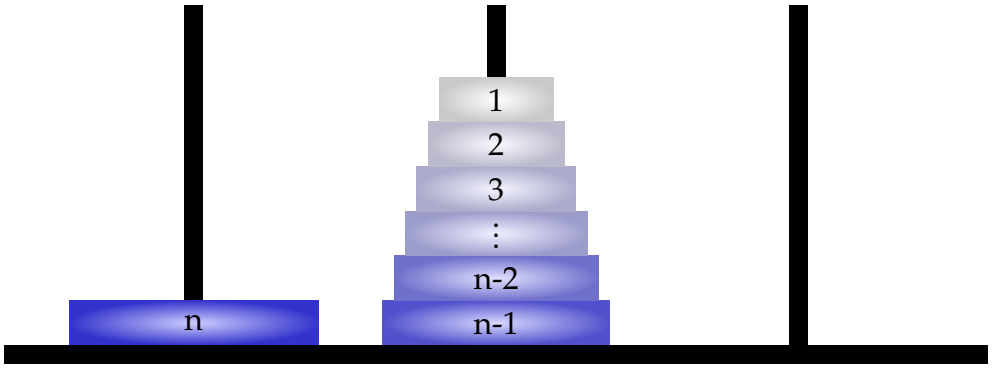
Move disc 1 from stick 2 to stick 3



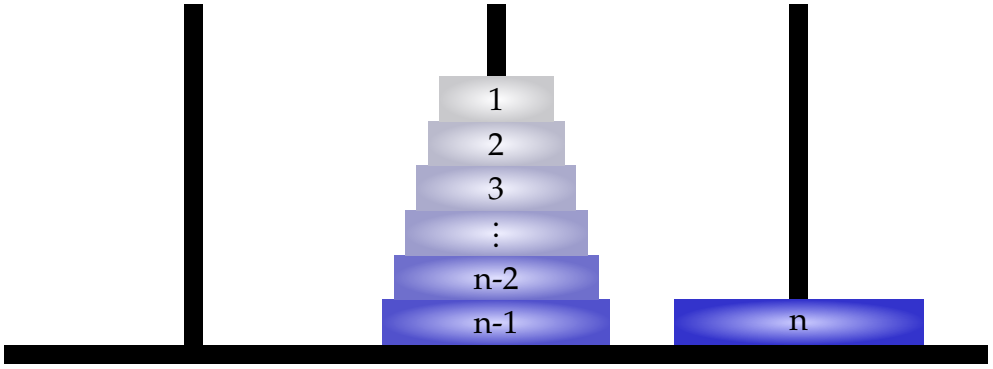
If there are three discs, it is still not so hard to solve the puzzle by some trial and error, but what if there are ten discs, or a hundred? To find a solution, let us try to think in a recursive way. We already know how to solve the puzzle for if there is only one disc (and also if there are two discs). Perhaps, just as for the factorial function, we can figure out what to do for a larger number of discs, say n discs, if we already would know what to do if there are less than n discs. Suppose therefore that $n \geq 2$ is a natural number and that we already know how to solve the puzzle if there are $n-1$ discs. This means that we know how to move a stack of $n-1$ discs from one stick to another stick. Then the following strategy works to move n discs:



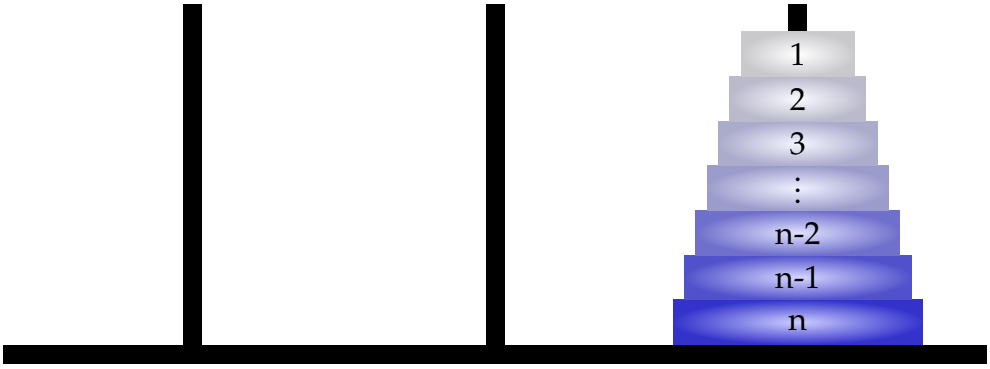
Using that we know how to move $n-1$ discs to another stick, move the stack of discs 1 to $n-1$ from stick 1 to stick 2.



Now move disc n from stick 1 to stick 3. This just takes one move.



Again using that we know how to move $n-1$ discs to another stick, move the stack of discs 1 to $n-1$ from stick 2 to stick 3.



This shows that the puzzle can be solved recursively! In particular, there is a solution for any number of discs.

5.3 The summation symbol Σ

If n is some natural number and $z_1, \ldots, z_n$ are complex numbers, then one can denote their sum by an expression like $z_1 + z_2 + \cdots + z_n$ or $z_1 + \cdots + z_n$. However, it is sometimes more convenient to have a more compact notation for this: $\sum_{k=1}^{n} z_k$. Using a recursive definition, we can be very precise:

$$\sum_{k=1}^{n} z_k = \begin{cases} z_1 & \text{if } n = 1, \\ \left(\sum_{k=1}^{n-1} z_k\right) + z_n & \text{if } n > 1. \end{cases} \tag{5-5}$$

Using this recursive definition, we obtain precisely what we wanted. One can simply use the definition and verify that indeed for small values of n one obtains:

n	$\sum_{k=1}^{n} z_k$
1	z_1
2	$z_1 + z_2$
3	$z_1 + z_2 + z_3$
4	$z_1 + z_2 + z_3 + z_4$

(5-6)

If $f : \mathbb{N} \to \mathbb{C}$ is a function, one similarly can replace the sum $f(1) + f(2) + \cdots + f(n)$ by the more compact expression $\sum_{k=1}^{n} f(k)$. Consider for example the expression $\sum_{k=1}^{n} k$, that is to say, the sum of the first n natural numbers. Similarly as in Table 5-6, we obtain

the following:

n	$\sum_{k=1}^{n} k$
1	1
2	$1+2=3$
3	$1+2+3=6$
4	$1+2+3+4=10$

(5-7)

Having this notation, will come in handy in various later chapters, but it is also heavily used in several other areas of mathematics and natural sciences.

The variable k in an expression like $\sum_{k=1}^{n} z_k$ is called the summation index. There is no reason to use the variable k as such and using another variable is completely fine. In particular one has for example $\sum_{k=1}^{n} z_k = \sum_{j=1}^{n} z_j$, since both summations amount to adding up the numbers $z_1, \ldots, z_n$. Also one may index the numbers that are to be added in a different way. In particular, if we want to add up the numbers $z_2, \ldots, z_{10}$, one can simply write $\sum_{k=2}^{10} z_k$.

5.4 Induction

In the previous section, we ended by solving the towers of Hanoi puzzle completely by approaching the problem in a recursive way. The number of moves our solution requires, can also be described recursively. If we denote by $T(n)$ the number of moves our strategy has for the puzzle with n discs, then we know that $T(1) = 1$ (the puzzle with only one disc can be solved in one move), but also that $T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1$ for $n \geq 2$ (our strategy involved moving a stack of $n-1$ discs twice and a single move of the nth disc). In other words, we have

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2 \cdot T(n-1) + 1 & \text{if } n \geq 2. \end{cases} \tag{5-8}$$

For instance $T(2) = 2 \cdot 1 + 1 = 3$, $T(3) = 2 \cdot 3 + 1 = 7$, and $T(4) = 2 \cdot 7 + 1 = 15$. It is striking that for these small values of n, the value of $T(n)$ is always one less than 2^n. Therefore one may "guess" that $T(n) = 2^n - 1$ for all natural number n. Let us test this conjecture, the word typically used instead of "guess", by computing $T(5)$. We have $T(5) = 2 \cdot T(4) + 1 = 2 \cdot 15 + 1 = 31$. This confirms our conjecture that $T(n) = 2^n - 1$ for $n = 5$. On the downside, all we know now is that the conjecture is true for all n in the set $\{1, 2, 3, 4, 5\}$. We could or course continue to verify our conjecture for more values of n by computing $T(6), T(7)$ and so on, but since there are infinitely many natural numbers, there is no way we can verify the formula $T(n) = 2^n - 1$ for *all* natural numbers n in

this way. Fortunately, there is a very intuitive property of the natural numbers that can help us out and which we state without proof:

|||| Theorem 5.3 Induction principle

Let S be a subset of the natural numbers and assume that S has the following two properties:

1. $1 \in S$,
2. if $n-1 \in S$ for some arbitrary natural number $n \geq 2$, then also $n \in S$.

In this case, we have $S = \mathbb{N}$.

The statement in this theorem is often called the *induction principle* or simply *induction*. Requirement 1. ($1 \in S$) is called the *base case of the induction*, while requirement 2. (if $n-1 \in S$ for some natural number n, then also $n \in S$) is called the *induction step*. The reason that in requirement 2., the natural number n has to be at least two, is that otherwise $n-1$ might not be a natural number. Indeed, if $n = 1$, then $n - 1 = 0$, but 0 is not in $\mathbb{N}$. Requirement 2. can be reformulated in propositional logic as follows.

2. for all $n \in \mathbb{N}_{\geq 2}$: $\quad n-1 \in S \quad \Rightarrow \quad n \in S$.

Verifying requirement 2., that is to say, verifying the induction step, is typically done by showing that $n \in S$ is true if we assume that $n-1 \in S$. When verifying the induction step $n-1 \in S \quad \Rightarrow \quad n \in S$, the assumption $n-1 \in S$ is called the *induction hypothesis*. The process of verifying the two requirements is typically called a *proof by induction* or, if the role of the variable n needs to be stressed, a *proof by induction on* n.

The induction principle is the key to understanding many statement in mathematics, but is also central in computer science, since there it can be used to show correctness of various algorithms, recursive definitions and computer programs.

In mathematics, it is convenient to use a reformulation of the induction principle, avoiding having to work with a subset $S \subseteq \mathbb{N}$. The reason is that this can be avoided using a nice consequence of Theorem 5.3. Such consequences are often called "corollaries" in mathematical texts and we will use the same terminology.

|||| Corollary 5.4

For each natural number n, let $P(n)$ be a logical proposition. Suppose that the following two statements are true:

1. $P(1)$,
2. for all $n \in \mathbb{N}_{\geq 2}$: $\quad P(n-1) \Rightarrow P(n)$.

Then $P(n)$ is true for all $n \in \mathbb{N}$.

Proof. In order to be able to use Theorem 5.3, we use a trick by defining $S = \{n \in \mathbb{N} \mid P(n)\}$. In other words, $n \in S$ by definition precisely if $P(n)$ is true. To be able to conclude that $P(n)$ is true for all natural number n, it is enough to show that $S = \mathbb{N}$. Indeed if there would exist some natural number m such that $P(m)$ is false, then by definition of S, we would have that $m \notin S$ and therefore that $S \neq \mathbb{N}$.

Now we use Theorem 5.3 to show that $S = \mathbb{N}$. The assumption that $P(1)$ is true, just means that $1 \in S$. The assumption that for all $n \in \mathbb{N}_{\geq 2}$: $\quad P(n-1) \Rightarrow P(n)$, means that whenever $n - 1 \in S$, also $n \in S$. Hence the two requirements from Theorem 5.3 are satisfied. Therefore by Theorem 5.3, we may conclude that $S = \mathbb{N}$. This, as remarked already, just means that $P(n)$ is true for all natural numbers n. □

As in Theorem 5.3, checking that $P(1)$ is valid is called the base case of the induction, while checking that for all $n \in \mathbb{N}_{\geq 2}$: $\quad P(n-1) \Rightarrow P(n)$, is called the induction step. While carrying out the induction step, the logical proposition $P(n-1)$ is called the induction hypothesis, similarly as before. Also, the statement in Corollary 5.4 as a whole is still called the induction principle. Hence to prove a claim of the form "$P(n)$ is true for all natural numbers n," we can follow the following strategy:

(i) Inform the reader that you are going to prove the claim that "$P(n)$ is true for all natural numbers n," using induction on n.

(ii) **Base case:** Check that $P(1)$ is valid.

(iii) **Induction step:** For an arbitrary natural number $n \geq 2$, assume that $P(n-1)$ is true and use this assumption (the induction hypothesis) to show that in that case also $P(n)$ is true. The challenge here is sometimes to figure out how to use the induction hypothesis $P(n-1)$ to one's advantage.

(iv) Once the previous items are finished, inform the reader that from the induction principle one can now conclude that $P(n)$ is valid for all natural numbers n.

Now, let use use this strategy to prove our conjecture that $T(n) = 2^n - 1$. In other words, let us prove the following:

Claim: Let $T : \mathbb{N} \to \mathbb{N}$ satisfy the recursion

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2 \cdot T(n-1) + 1 & \text{if } n \geq 2. \end{cases}$$

Then for all $n \in \mathbb{N}$ we have $T(n) = 2^n - 1$.

Proof. Let $P(n)$ be the statement $T(n) = 2^n - 1$. We will show the claim using induction on n.

Base case: We have $T(1) = 1$. Since $2^1 - 1 = 1$, we see that $T(1) = 2^1 - 1$. Hence $P(1)$ is valid.

Induction step: Let $n \geq 2$ be an arbitrary natural number. The induction hypothesis is $P(n-1)$, which in our case just means the equation $T(n-1) = 2^{n-1} - 1$. Assuming this, we should derive that $P(n)$ is valid. In other words, assuming that $T(n-1) = 2^{n-1} - 1$, we should derive that $T(n) = 2^n - 1$. From the recursive definition of $T(n)$, using that $n \geq 2$, we know that $T(n) = 2 \cdot T(n-1) + 1$. Combining this with the induction hypothesis, we see that

$$T(n) = 2 \cdot T(n-1) + 1 = 2 \cdot (2^{n-1} - 1) + 1 = 2 \cdot 2^{n-1} - 2 \cdot 1 + 1 = 2^n - 1.$$

This is exactly what we needed to show.

Now that we have carried out the base case of the induction as well as the induction step, we can conclude from the induction principle that the statement $T(n) = 2^n - 1$ is true for all natural numbers n. □

One can actually show that the strategy we found in Section 5.2 is the best possible. In other words, any solution of the puzzle with n discs will take at least $T(n)$ moves. We see that solving a ten disc version of the towers of Hanoi, already would take $2^{10} - 1 = 1023$ moves.

The best way to get the hang of proofs by induction is to look at several examples and then to try to do an inductive proof yourself. Let us therefore look at some more examples. Here is a famous one:

|||| Example 5.5

Let us denote by $S(n)$ the sum of the first n natural numbers. Informally, one often writes $1 + 2 + \cdots + n$ for this sum, while we can also use the summation sign and write $S(n) = \sum_{k=1}^{n} k$. As we saw in Table 5-7, we have for example $S(1) = 1$, $S(2) = 1+2 = 3$, $S(3) = 1+2+3 = 6$ and $S(4) = 1+2+3+4 = 10$. The claim is that the following equality holds for all natural number n:

$$S(n) = \frac{n \cdot (n+1)}{2}.$$

Note that $S(n)$ satisfies the following recursion:

$$S(n) = \begin{cases} 1 & \text{if } n = 1, \\ S(n-1) + n & \text{if } n \geq 2. \end{cases}$$

Indeed, we have already observed that $S(1) = 1$, while if $n \geq 2$, using equation (5-5), we obtain that

$$S(n) = 1 + \cdots + n = \sum_{k=1}^{n} k = \left(\sum_{k=1}^{n-1} k \right) + n = S(n-1) + n.$$

Now let us prove the following claim.

Claim: For $n \in \mathbb{N}$, let $S(n) = 1 + \cdots + n$, the sum of the first n natural numbers. Then $S(n) = \frac{n \cdot (n+1)}{2}$.

Proof. We prove the claim using induction on n.

Base case: If $n = 1$, then $S(1) = 1$, while $\frac{1 \cdot (1+1)}{2} = 1$. Hence the formula $S(n) = \frac{n \cdot (n+1)}{2}$ is valid for $n = 1$.

Induction step: Let $n \geq 2$ be an arbitrary natural number and assume as induction hypothesis that $S(n-1) = \frac{(n-1) \cdot (n-1+1)}{2}$. We can simplify the induction hypothesis slightly and say that it holds that $S(n-1) = \frac{(n-1) \cdot n}{2}$. Assuming the induction hypothesis and using that

$S(n) = S(n-1) + n$, we may conclude that

$$\begin{aligned} S(n) &= S(n-1) + n \\ &= \frac{(n-1)\cdot n}{2} + n \\ &= \frac{(n-1)\cdot n}{2} + \frac{2\cdot n}{2} \\ &= \frac{n^2 - n}{2} + \frac{2\cdot n}{2} \\ &= \frac{n^2 - n + 2\cdot n}{2} \\ &= \frac{n^2 + n}{2} \\ &= \frac{n\cdot(n+1)}{2}. \end{aligned}$$

This is exactly what we needed to show, completing the induction step.

Using the induction principle, we may conclude that the formula $S(n) = \frac{n\cdot(n+1)}{2}$ is valid for all natural numbers n. □

|||| Example 5.6

This example is of a more theoretical nature and can be skipped on a first reading. We want to make sure that the recursive definition we gave previously of the factorial function $\texttt{fac} : \mathbb{N} \to \mathbb{N}$ in equation (5-1), actually was correct from a mathematical point of view. The issue is that we never showed that `fac` is defined by its recursive description for *any* natural number n. In other words, when writing $\texttt{fac} : \mathbb{N} \to \mathbb{N}$, we implicitly say that the domain of the function is $\mathbb{N}$, but how do we know? What we need to do is to show that for any natural number n, the recursive description in equation (5-1) will give rise to the output value $\texttt{fac}(n)$ after finitely many steps.

Therefore, let $P(n)$ be the statement that $\texttt{fac}(n)$ can be computed in finitely many steps using equation (5-1) for any natural number n. We want to show that this statement $P(n)$ is true for all natural numbers. The base of the induction is taken care of by the observation that equation (5-1) immediately implies that $\texttt{fac}(1) = 1$. Now let $n \geq 2$ be an arbitrary natural number and assume as induction hypothesis that $\texttt{fac}(n-1)$ can be computed in finitely many steps using equation (5-1). Since $n \geq 2$, equation (5-1) implies that $\texttt{fac}(n) = \texttt{fac}(n-1)\cdot n$. Hence given $\texttt{fac}(n-1)$, all we need is one multiplication with n to compute $\texttt{fac}(n)$. Hence $\texttt{fac}(n)$ can be computed in finitely many steps, if $\texttt{fac}(n-1)$ can. This completes the induction step.

More generally a function $f : \mathbb{N} \to B$, from the natural numbers $\mathbb{N}$ to a given set B, can

be defined recursively as long as $f(1)$ is specified and for any $n \geq 2$, the value $f(n)$ can be computed from $f(n-1)$. The reason is that in such cases, a very similar reasoning as the one we just carried out for the factorial function, applies. In particular, equation (5-2) defines z^n for any natural number n.

5.5 A variant of induction

Many variants of induction exist. In this section, we would like to mention one of them: induction starting with a different base case. So far, the base case of our induction proofs always was the case $n = 1$ and after that we considered larger natural numbers n. In some cases however, a logical statement also makes sense for other values of n. Consider for example the statement:

$$\text{A polynomial } p(Z) \in \mathbb{C}[Z] \text{ of degree } n \text{ has at most } n \text{ roots in } \mathbb{C}.$$

This statement also makes sense for $n = 0$. Indeed, for $n = 0$ the statement is rather easy to verify: a polynomial $p(Z)$ of degree zero, is just a nonzero constant p_0. Indeed, the constant p_0 is nonzero precisely since in general the leading terms of a degree d polynomial is nonzero by Definition 4.1. But then $p(z) = p_0 \neq 0$ for all $z \in \mathbb{C}$, implying that the polynomial has no roots.

Conversely, there are statements that only become true for large enough values of n. Consider for example, the statement:

$$\text{There exist } n \text{ points in the plane } \mathbb{R}^2 \text{ that do not lie on a line.}$$

If $n = 1$, this is wrong, since there are many lines through any given point. Also if $n = 2$, this is wrong, since given any two points, the line connecting them will contain these points. However, for $n \geq 3$, the statement is true. Indeed, if $n \geq 3$, we can for example choose three of the points as the vertices of an equilateral triangle and the remaining $n - 3$ points arbitrarily.

Because of these kind of examples, it is convenient to have a slightly more flexible variant of induction. For a given integer $a \in \mathbb{Z}$, we denote by $\mathbb{Z}_{\geq a} = \{n \in \mathbb{Z} \mid n \geq a\}$. For example $\mathbb{Z}_{\geq -1} = \{-1, 0, 1, 2, \dots\}$. With this notation in place, we can formulate the following variant of induction, called *induction with base case b*:

Theorem 5.7

Let $b \in \mathbb{Z}$ be an integer and for each integer $n \geq b$, let $P(n)$ be a logical proposition. Suppose that the following two statements are true:

1. $P(b)$,
2. for all $n \in \mathbb{Z}_{\geq b+1}$: $P(n-1) \Rightarrow P(n)$.

Then $P(n)$ is true for all $n \in \mathbb{Z}_{\geq b}$.

Proof. Let us define the logical statement $Q(n)$ to be $P(n+b-1)$. Then $Q(n)$ is defined for any natural number n. Indeed if $n \geq 1$, then $n+b-1 \geq b$. Now we apply Corollary 5.4 to the logical statements $Q(n)$. The first requirement from Corollary 5.4 then is that $Q(1)$ should be valid. However, this is fine, since $Q(1) = P(b)$ and it is given that $P(b)$ is valid. The second requirement from Corollary 5.4 becomes that for all $n \in \mathbb{N}_{\geq 2}$: $Q(n-1) \Rightarrow Q(n)$. However, since $n \geq 2$, we have $n+b-1 \geq b+1$ and therefore $n+b-1 \in \mathbb{Z}_{\geq b+1}$. Since $Q(n-1) = P(n+b-2)$ and $Q(n) = P(n+b-1)$ and the implication $P(n+b-2) \Rightarrow P(n+b-1)$ is valid (we know that $n+b-1 \in \mathbb{Z}_{\geq b+1}$), we see that the implication $Q(n-1) \Rightarrow Q(n)$ is valid. Hence the second requirement for the logical statements $Q(n)$ when applying Corollary 5.4 is also met. Hence the corollary implies that $Q(n)$ is valid for all natural numbers n. Since $Q(n) = P(n+b-1)$, this means that $P(n+b-1)$ is valid for all natural number n. In particular $P(1+b-1) = P(b)$ is valid, $P(2+b-1) = P(b+1)$ is valid, and so on. This amount to the statement that $P(n)$ is valid for all integers $n \geq b$, which is what we wanted to show. □

Note that if we choose $b = 1$, we recover Corollary 5.4. The overall structure of a proof with induction with base case b is the same as for the usual induction. One still has a base case and an induction step. Let us consider an example of a proof by induction of this type.

Example 5.8

Consider the inequality $n + 10 \leq n^2 - n$. Since a polynomial of degree two like $n^2 - n$ grows faster than a degree one polynomial like $n + 10$, one should expect that if n becomes large enough this inequality is true. Now let us denote by $P(n)$ the statement that $n + 10 \leq n^2 - n$. In this case, we can define $P(n)$ for any integer n. The statement $P(4)$ for example is the inequality $4 + 10 \leq 4^2 - 4$. This is false since in fact $14 = 4 + 10 > 4^2 - 4 = 12$. On the other

hand, for $n = 5$, the statement $P(5)$ is true, since $15 = 5 + 10 \leq 5^2 - 5 = 20$. We claim that $P(n)$ is true for any $n \in \mathbb{Z}_{\geq 5}$ and give a proof by induction using Theorem 5.7 with $b = 5$:

Base case: We have already verified that $P(5)$ is valid, so the base case is done.

Induction step: Let $n \geq 6$ be an arbitrary natural number and assume as induction hypothesis that $P(n-1)$ is valid. In particular, this means that we may assume that $(n-1) + 10 \leq (n-1)^2 - (n-1)$. Using this assumption, we should deduce that $P(n)$ is valid. Let us first rewrite the induction hypothesis in a more convenient form. We have $(n-1) + 10 = n + 9$, while $(n-1)^2 - (n-1) = n^2 - 2n + 1 - n + 1 = n^2 - 3n + 2$. Hence the induction hypothesis amounts to assuming that the inequality $n + 9 \leq n^2 - 3n + 2$ is valid. But then we can deduce:

$$\begin{aligned} n + 10 &= (n + 9) + 1 \\ &\leq (n^2 - 3n + 2) + 1 \\ &= n^2 - 3n + 3 \\ &= n^2 - n - 2n + 3 \\ &\leq n^2 - n. \end{aligned}$$

The final inequality holds, since $-2n + 3 \leq 0$ for any $n \geq 6$ (in fact even for any $n \geq 2$). We conclude that if $P(n-1)$ is true, then $n + 10 \leq n^2 - n$, that is to say $P(n)$, is true as well. This is what we needed to show, thus completing the induction step.

Using induction with base case 5, we may conclude that the inequality $n + 10 \leq n^2 - n$ is valid for all $n \in \mathbb{Z}_{\geq 5}$.