# Introduction to Mathematics in Computer Science

Book of the Mathematics Preparatory Course

UNIVERSITÄT DES SAARLANDES

SIC Saarland Informatics Campus

## Preface

Welcome! This book is a part of the Mathematics Preparatory Course at Saarland University.

If you are reading these lines, you are most likely about to start a new chapter of your life—your studies in a computer-science-related field at Saarland University. Again, welcome here!

Whether you are studying cybersecurity, bio- or business informatics, data science and artificial intelligence, media informatics, "plain" computer science, or another related field—in your studies, you will see, and need, a lot of math and logic. After all, at its core, computer science *is* a particular(ly weird and wonderful) field of mathematics. Formally, the basic lectures will only require the knowledge you already bring from school. However, our experience has shown that many students struggle with the more formal, mathematical aspects of their studies. Our Preparatory Course can (and will) help smooth that journey. Whether you are here to brush up on some aspects of math you learned previously or want a small head start into the new mathematical world you are about to enter—our Mathematics Preparatory Course is here for you, and this book will be your companion throughout.

In the next chapters (and weeks), you will (re-)discover the foundations of logic, learn how to prove things, and get a new perspective on sets and relations.

## Acknowledgements

This book only exists because countless people put their heart and soul into it.

It was primarily written by **Jonathan Baumann**, **Christian Hagemeier**, **Johannes Hostert**, **Nils Husung**, **Iona Kuhn**, **Robert Pietsch**, and **Niklas Schneider**.

Elina Celik, Marc Hermes, Janine Lohse, Benjamin Peters, Tim Rohde, Lisa-Marie Rolli, Gregory Stock, Marcel Ullrich, and Oliver Valta helped shape this book by fighting through early drafts and by providing additional valuable input. **Felix Freiberger** additionally provided extremely valuable advice and kept the writers motivated throughout the entire process.

The Mathematics Preparatory Course would not be possible in its current shape without the invaluable work put in by **Christian and Julia Eisentraut** over many years. They also wrote the German-language book that served as a precursor to, and inspiration for, the book you are holding in your hands now.

Finally, the whole team of the Mathematics Preparatory Course would like to express their gratitude towards the whole department of computer science and especially our executive producer **Erich Reindel** for their continued and invaluable support—financial and otherwise—throughout the existence of this course.

## License

## Beware, reader!

Although this book was thoroughly checked by for bugs, mistakes, typos, errors, inconsistencies, historical inaccuracies, nonsense, and other design flaws, it may still contain several of those. Of course, all of these are entirely the fault of the primary authors. If you find a mistake, or if you have additional feedback, please tell us at book@vorkurs.cs.uni-saarland.de.

Book built on October 2, 2023 from commit `b04a38de64fb32a3e029c3a2166ba733dc5c1b3c`

*This page is intentionally left blank.*

# Contents

# 1 | Formal Languages

What is a valid arithmetic expression?

This is something you probably never had to think much about, and it may seem very trivial. Off the top of your head, you could come up with a large number of examples from "$2 + 2$" to "$1337 \cdot 42 + 1234 \cdot (8765 + 23)$."

However, could you come up with a concise set of rules that describes every arithmetic expression? And can you do it in such a way that it disallows "$+ 42 +$" or "$1 + - \cdot 3$," which are not meaningful? How do we even give meaning to an arithmetic expression?

Of course, you have some very good intuitions about this, and you might think that this is enough. However, we would very much like to avoid working with intuitions, as they are unreliable especially when introducing new concepts to someone. To illustrate this, consider the following example in natural language, which relies on intuitions: "Call me a cab, please." Your intuition probably tells you that you're being asked to make a phone call. However, if you say this to someone who does not know English very well (and does not yet have the same intuition—maybe they don't know the word "cab"), they might just start to refer to you as "a cab" now.

Clearly, we would like to avoid such problems in mathematics, so we would like to describe everything as precise as possible. To do so, we will introduce formal languages, which are a fairly abstract concept that allows us to model a wide variety of things. At the most basic level, a formal language is simply a collection of expressions that adhere to the rules of that language.

We will continue with the example of arithmetic expressions, but you will encounter lots of other uses in your studies. For example, formal languages are also used to describe programming languages.

This topic is split into two main areas: **syntax** is concerned with the rules that describe what exactly is part of our language and what is not. **Semantics**, on the other hand, deals with how we can give meaning to expressions.

**Chapter Goals**

In this chapter you will learn:

- What a BNF is and how we can use it to specify the syntax of a language.
- What mathematicians call a tree and why we need precedence rules.
- How to give a meaning to expressions of a language by the means of operational and denotational semantics.

## 1.1 Syntax

The syntax of a language describes which expressions are **well-formed**, i.e. considered to be part of this language. Let's consider arithmetic expressions on natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$.[1] We also allow the use of variables. From this brief description, you as an educated person know that "$42 + 1337$" is part of our language. You would also agree that "$3 -$ " is not a valid expression and hence not part of our language. But how would you explain this to someone who does not know arithmetic expressions yet?

To make our intuition of arithmetic expressions precise, we could define them as follows:

- Every natural number $n \in \mathbb{N}$ is an arithmetic expression.
- Every variable $x, y, z, \ldots$ is an arithmetic expression.
- If $\varphi$ and $\psi$ are arithmetic expressions, then $\varphi + \psi$ is an arithmetic expression.
- If $\varphi$ and $\psi$ are arithmetic expressions, then $\varphi - \psi$ is an arithmetic expression.
- If $\varphi$ is an arithmetic expression, then $-\varphi$ is an arithmetic expression.
- If $\varphi$ and $\psi$ are arithmetic expressions, then $\varphi \cdot \psi$ is an arithmetic expression.

Of course, we could add more operators, but for now, we restrict ourselves to addition, subtraction, negation and multiplication.

**Checkpoint 1.1:** Arithmetic Expressions

Is "$4 \div 2$" an arithmetic expression? What about "$42.5 - 0.5$" and "$x + a$"?

This approach is not bad; however, it is quite lengthy and uses natural language with all its pitfalls. But before we have a look at a more elegant presentation, let us dwell on this definition for a bit longer. What we saw is called an **inductive definition**. We have six *cases*, of which the first two are the *base cases* and the last four are *inductive cases*. The difference is that in the base cases, we can directly construct an arithmetic expression, while in the induction cases, we need to already have constructed one or two arithmetic expressions. We refer to these expressions using so-called **meta-variables**, denoted by Greek letters $\varphi$ (phi), $\psi$ (psi), $\rho$ (rho), … Note that meta-variables are not part of our language itself, we only use them at the meta-level to talk *about* our language. That is in contrast to the variables $x, y, z, \ldots$, which denote arithmetic expressions.

With the definition, we wanted to explicitly spell out what an arithmetic expression is. And indeed, we clearly point out how we can construct them. But can we actually say that "$3 -$ " is *not* an arithmetic expression? If we really wanted to be precise, then we would need to say that nothing else than what is derivable from our six rules is an arithmetic expression. But actually, this is what an inductive definition is about: it defines the set (in our case the language) that is induced by some rules. So we typically do not spell out the assertion and just take it for granted.

[1] Computer scientists typically consider 0 to be part of the natural numbers, and so do we throughout this course. In your math lectures, however, the natural numbers will probably start with 1.

### 1.1.1 Backus–Naur Form

John Backus, a programming language designer at IBM, faced the same problem of lengthy descriptions when working on IAL, an old programming language that was called ALGOL later on. This led him to invent a formal notation. The community first called this notation “Backus normal form”; however, Donald Knuth pointed out that it is not a normal form in the classical sense. Since some contributions were also made by Peter Naur, the notation finally received its name “**Backus–Naur form**,” which is commonly abbreviated as **BNF**. In this course, we don’t use the original notation, but a slightly different variant. In this style, the definition of our arithmetic expressions looks like this:

**Definition 1.2** (Arithmetic Expressions)**.**

$$\mathcal{E} \ni \varphi, \psi ::= n \mid x \mid \varphi + \psi \mid \varphi - \psi \mid -\varphi \mid \varphi \cdot \psi \qquad n \in \mathbb{N}, x \textit{ is a variable}$$

Let us walk through the definition step by step. What we are defining is the language $\mathcal{E}$. The meta-variables $\varphi$ and $\psi$ should denote expressions of this language. We use the inverted element-of symbol $\ni$ here to express this. On the right-hand side of $::=$, we have our six cases again. As $n$ and $x$ are just placeholders, we remark what they stand for on the right.

**Checkpoint 1.3:** ab-BNF

What language does the BNF $\mathcal{L} \ni \varphi ::= a\ \varphi \mid b$ describe?

There is an important restriction on meta-variables, namely that we can use them only once in each case. That is, we cannot have a BNF like $\mathcal{L} \ni \varphi ::= a \mid b \mid \varphi \circ \varphi$. The case $\varphi \circ \varphi$ would impose that the left-hand and the right-hand side of the $\circ$ operator are the same. Our BNFs correspond to what is called **context-free grammars**. The common programming languages (and even many natural languages) can be described using context-free grammars. But there are also languages for which one cannot come up with a BNF, for example the language where every expression has the structure $\varphi\varphi$. Here, $\varphi$ is an arbitrary sequence of characters (and the set of characters must contain at least two elements). Being non-context-free also applies to natural languages such as Swiss-German.[2] There are other more powerful grammar formalisms, however it is typically harder to parse them, i.e. turn the expression into a syntax tree.

**In Other Words:** Meta

When talking about a (formal) language, we need some language to talk in. This language is called the **metalanguage**. Typically, this metalanguage is a natural language like English or German, but we can also use formal languages like a BNF. Like the **object language** (the language we are studying), the metalanguage can have variables. You already know them, we call them meta-variables. Later on, we also define the semantics of a language in terms of the metalanguage. Always keep in mind that we have both an object language and a metalanguage and strictly separate the two!

[2] https://en.wikipedia.org/wiki/Cross-serial_dependencies

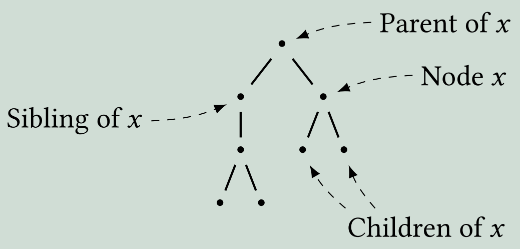
### 1.1.2 Syntax Trees

You might now wonder why we do not have any parentheses in our language. According to the BNF, we can derive $1$ and $2 + 3$. But what if we compose these two expressions with a $-$? Do we obtain $1 - 2 + 3$? From our mathematical intuition, it should rather be $1 - (2 + 3)$. And indeed, the latter would be the correct answer. The point is that the expressions of any language defined by a BNF are tree-like objects.

Before we continue with our example, let us have a look at what trees are, in the mathematical sense of course. **Trees** are quite an important mathematical concept and in computer science, there are many data structures making use of them. Although there are a few similarities between our trees and trees in nature, they are more related to family trees. For talking about trees, we use vocabulary of both areas.

A tree consists of **nodes** (•) and **edges** (—) between them. In the form of trees we consider, a tree has separate levels, just like a family tree. Given a node $x$, there may be a single node $y$ on the level above with an edge to $x$. The node $y$ is called the **parent** of $x$. Conversely, $x$ is a **child** of $y$. Each node may have one or no parent and arbitrarily many children. Nodes that share the same parent are called **siblings**. We may draw a tree like this:

**Example 1.4:** A Tree



Like in a family tree, there are **ancestors**—the node's parent, the parent's parent, the parent of the parent's parent, and so on—as well as **descendants**—the node's children, the children's children, and so on. Oftentimes, we need to speak of the ancestors or descendants including the node itself. Since we are lazy about writing, we usually define the ancestors and descendants such that they include the node itself. We remark that this is controversial, so you may find contexts in which this is not the case. But in this course, a node is an ancestor and a child of itself.

There are some special nodes: a node without a parent is called a **root** and a node without children is called a **leaf**. So like in nature, our trees are branching from the root towards the leafs. However, our graphical representation is upside down compared to nature.

All nodes which are not leafs also have a special name, they are called **inner nodes**. Our trees have exactly one root. That is why the trees of our form are called **rooted trees**.[3]

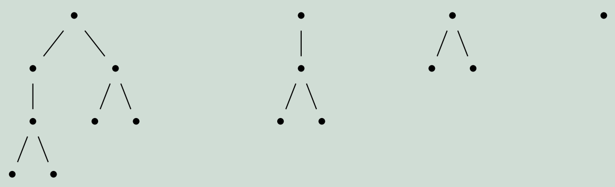
There are a few more concepts related to trees, but we just introduce one more term: a **subtree** starting at a node $s$ is the tree containing all the descendants of $s$.

Now, let's return to our example expression. In Figure 1.7 it is depicted as a **syntax tree**. Observe that instead of the "uninformative" •-nodes, we have information attached to the nodes. Concretely,

[3]In general, a tree is just a graph without cycles. But without a unique root, the concepts of children and parents do not work.

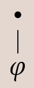
**Example 1.5:** Subtrees

The tree presented in Example 1.4 has 4 distinct subtrees:



**Checkpoint 1.6:** The Language of Trees

Give a BNF for binary trees, trees with at most two children per node. You may use constructs like the one on the right.



we annotate the nodes with the corresponding case of the BNF. In general, trees with annotations are called **labeled trees**, and syntax trees are a special kind thereof.

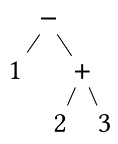


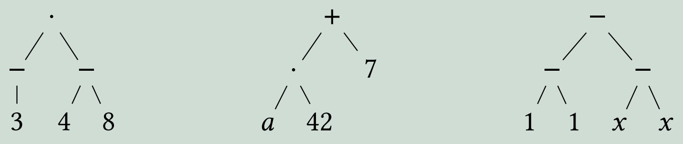
Figure 1.7: Syntax tree corresponding to $(1 - (2 + 3))$

Note that all the terms related to trees from above also apply here: the base cases of our inductive definition (variables and numbers) correspond to leaves and the induction cases match the inner nodes.

If we always had to draw such a tree to denote an arithmetic expression, we would have ended up in a big mess. Fortunately, there is a shorter way: one writes it in a linear fashion and parenthesizes every subtree. Typically one omits the parentheses around leaves. Using this convention, the example expression can be written as $(1 - (2 + 3))$. We call such an expression **fully parenthesized**.

**Example 1.8:** Fully Parenthesized Expressions and Their Syntax Trees

$((-3) \cdot (4 - 8))$  $((a \cdot 42) + 7)$  $((1 - 1) - (x - x))$



There is a one-to-one correspondence between syntax-trees and fully parenthesized expressions. Especially the aspect that each linearized expression has a unique syntax tree is important—otherwise we might have to guess what the expression really "means." We have to keep this uniqueness-property when we make our life even easier with so-called precedence rules next up.

But before we continue, one final remark: parentheses are still *not* part of our language. They are just used as an external means to denote a syntax tree in a linear fashion.

**Checkpoint 1.9:** Syntactic Equality

Do $((1 + 1) + 1)$ and $(1 + (1 + 1))$ have the same syntax tree? Are these expressions equal?

### 1.1.3 Precedence Rules

Having to write fully parenthesized is still not desirable. We would rather want to elide unnecessary parentheses. For this, we use so-called **precedence rules**. There are two important aspects we specify: how strong an operator binds as well as the operator's associativity. From what you have learned in school you know that $2+3\cdot 4$ is implicitly parenthesized as $2+(3\cdot 4)$. We say that $\cdot$ binds stronger than $+$, or $\cdot$ has higher precedence than $+$. You also know, that $2-1-1$ is implicitly parenthesized as $(2-1)-1$. In general, if we have multiple operators of the same precedence level and group them from the left, we say that these operators are **left-associative**. So $-$ is a left-associative operator. Conversely, if we group multiple operators of the same precedence level from the right, these are **right-associative**. You might now wonder whether there is an example of a right-associative operator. Indeed, the arithmetic expressions we have considered so far do not have such an operator, so let's extend them a little:

**Definition 1.10** (Extended Arithmetic Expressions).

$$\mathcal{E} \ni \varphi, \psi ::= n \mid x \mid \varphi + \psi \mid \varphi - \psi \mid -\varphi \mid \varphi \cdot \psi \mid \varphi \div \psi \mid \varphi \hat{} \psi \qquad n \in \mathbb{N}, x \textit{ is a variable}$$

As already hinted, $\hat{}$ is a right-associative operator, that is $\varphi \hat{} \psi \hat{} \rho$ is implicitly parenthesized as $\varphi \hat{} (\psi \hat{} \rho)$. We use the notation $\varphi \hat{} \psi$ instead of the more common $\varphi^{\psi}$ here. This is because $\hat{}$ behaves a bit more like the other operators. If we have $\varphi^{\psi+\rho}$ for instance, the exponent is already implicitly parenthesized (i.e. $\varphi^{(\psi+\rho)}$).

Now, let's have a look at all the rules. To distinguish the two $-$ operators, we call the one with a single operand **unary** and the one with two operands **binary**.

**Definition 1.11** (Precedence Rules for Arithmetic Expressions).

| *Operator* | *Precedence level* | *Associativity* |
|---|---|---|
| $\hat{}$ | *3* | *right* |
| $-$ *(unary)* | *2* | — |
| $\cdot, \div$ | *1* | *left* |
| $+, -$ *(binary)* | *0* | *left* |

**Example 1.12:** Fully and Minimally Parenthesized Expressions

- $((1 + 2) + 3) = 1 + 2 + 3$
- $((3 - 2) + 1) = 3 - 2 + 1$
- $(2 \cdot ((-3)\hat{}4)) = 2 \cdot (-3)\hat{}4$
- $((10 \div (-2))\hat{}2) = (10 \div -2)\hat{}2$

Just a quick remark on the example: writing $10 \div -2$ may look strange. In maths, one would rather write $10 \div (-2)$. But $10 \div -2$ is completely fine by our precedence rules and we wanted the minimal number of parentheses here. In practice, we rather want to write an expression such that it can easily be read by others. For this, it usually is a good idea to remove unnecessary parentheses. But in some cases, a technically redundant parenthesis can improve readability, and thus should be added, even if it is not strictly necessary. As a general rule, it is never wrong to add more parentheses.

Note that we did not specify the associativity of the unary $-$. The concept of left- or right-associativity only applies to binary infix operators. **Infix** notation means that the operator is written in between the operands. There is also **prefix** notation where the operator is written in front of all its operands (e.g. $\cdot\ (+\ 3\ 4)\ 2$), as well as **postfix** notation where the operator comes last.

Now imagine that you are given an expression that is not necessarily fully parenthesized, say $-x\char`\^(3 + (-2))\char`\^2 \div 2 \div 3$. How would you go about drawing its syntax tree? The idea is to work from the lowest to the highest precedence level. If an operator is left-associative, start with the right-most occurrence, if it is right-associative with the left-most occurrence. And do not consider everything inside parentheses until you are finished with the highest precedence level. When beginning with an expression inside parentheses, start again with the lowest precedence level. This way you can draw the tree from top to bottom. Try applying this procedure to the expression above!

**Checkpoint 1.13:** Associative Operators

You know that there are associativity laws for $+$ and $\cdot$, i.e. $(x + y) + z = x + (y + z)$ and $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ at the semantic level. Do we actually need to specify the operator's associativity here?

One final remark on precedence rules: We did only specify them for arithmetic expressions so far. If you come up with a new language, you have to either write fully parenthesized expressions or define precedence rules as well. Otherwise, our requirement that each valid linearized expression must correspond to exactly one syntax tree is not fulfilled.

### 1.1.4 Syntactic Equality

One thing that we only considered in the checkpoints so far is the equality of expressions. If you wrote something like $1 + 1 = 2$ back in primary school, you considered the semantics of the expressions $1 + 1$ and $2$. However, we have not defined any semantics for our arithmetic expressions yet. Then how could we write $((1 + 2) + 3) = 1 + 2 + 3$ in Example 1.12? The answer is that we were only concerned about **syntactic equality**.

> **Definition 1.14** (Syntactic Equality)**.** *Two expressions are syntactically equal if and only if*[4] *they have the same syntax tree.*

Note that this is a meta-level property. The $=$ sign does not belong to the language of arithmetic expressions but to the meta-language.

[4]"If and only if" is common to express an implication in two directions "if A then B" and "if B then A." In fact, "if and only if" is so common that it is frequently abbreviated as "iff." More on this in Section 2.1.

Looking back at Checkpoint 1.13, it becomes clear that the associativity laws at the semantic level do not influence which precedence rules we need. Note furthermore that the “=” there does not mean syntactic equality, but semantic equality instead. In Section 1.2, we formally define what our arithmetic expressions actually mean, thereby defining semantic equality. But for now, we rather care about the syntax and we should strictly separate different notions of equality.

### 1.1.5 More Complex Languages

Sometimes, it might be handy to use multiple BNFs in conjunction. If you want to model decimal numbers, we could do it like this:

$$\begin{aligned}\mathcal{D} \ni \varphi ::= {} & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ & \mid 0\,\varphi \mid 1\,\varphi \mid 2\,\varphi \mid 3\,\varphi \mid 4\,\varphi \mid 5\,\varphi \mid 6\,\varphi \mid 7\,\varphi \mid 8\,\varphi \mid 9\,\varphi\end{aligned}$$

However, needing two cases for every numeral is not really desirable. What if we were able to get rid of the base cases and replace them with a symbol ($\varepsilon$) representing literally nothing? Then, our BNF would look like this:

$$\mathcal{D}' \ni \varphi ::= \varepsilon \mid 0\,\varphi \mid 1\,\varphi \mid 2\,\varphi \mid 3\,\varphi \mid 4\,\varphi \mid 5\,\varphi \mid 6\,\varphi \mid 7\,\varphi \mid 8\,\varphi \mid 9\,\varphi$$

Technically, the syntax trees of 123 in $\mathcal{D}$ and $\mathcal{D}'$ are a bit different:

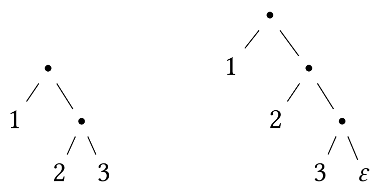


Figure 1.15: Syntax tree of 123 in $\mathcal{D}$ and $\mathcal{D}'$

This is one of the drawbacks of using $\varepsilon$. Furthermore, grammars containing $\varepsilon$ are unsuitable for some parsing approaches. However, it is possible to transform grammars containing $\varepsilon$ into grammars not containing it. And fortunately, there is also a shorter way to model decimal numbers with only a single case per numeral. The trick is to use multiple BNFs:

$$\begin{aligned}\mathcal{D}_1 \ni \varphi &::= \psi \mid \psi\,\varphi \\ \mathcal{D}_2 \ni \psi &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

It is even possible to have **mutually inductive definitions** (here, we don’t model decimal numbers):

$$\begin{aligned}\mathcal{X}_1 \ni \varphi &::= \varphi \circ \psi \mid \psi \\ \mathcal{X}_2 \ni \psi &::= \rho \bullet \psi \mid \rho \\ \mathcal{X}_3 \ni \rho &::= 1 \mid 2 \mid \langle \varphi \rangle\end{aligned}$$

**Checkpoint 1.16:** Mysterious Languages

You might ask whether the language $\mathcal{X}_1$ is equivalent to the following language:

$$\mathcal{X} \ni \varphi, \psi ::= \varphi \circ \psi \mid \varphi \bullet \psi \mid 1 \mid 2 \mid \langle \varphi \rangle$$

It turns out that the syntax trees in $\mathcal{X}_1$ are more constrained than the ones in $\mathcal{X}$. How? Does it encode precedence rules in some way?

**Going Beyond:** Concrete vs. Abstract Syntax

Do you remember our discussion about missing parentheses from the introduction? This discussion might seem strange now if we do not consider parentheses to be part of our languages anyway. But the thoughts there were not pointless at all: if we want to specify the syntax of a programming language, we should also specify where parentheses are needed. The answer is that there are two levels: **concrete syntax** and **abstract syntax**. So far, we only considered the latter.
The goal of concrete syntax in the context of programming languages is to define a way to transform the source code into an abstract syntax tree. This is typically done in two steps: First, we have a **lexer** that splits the input into tokens or words. In a second step, the **parser** turns this token stream into a phrase, which again has a tree-like structure. If we then strip of things like parentheses, we arrive at the **abstract syntax tree** (commonly abbreviated as **AST**).
If you have a look at the specification of a programming language like C[a], then you will find a description of the concrete syntax, usually in a shape that more or less corresponds to a BNF. The following is just a small excerpt from C's language syntax summary. The entire summary spans 17 pages.

(6.8) *statement:*

*labeled-statement*
*compound-statement*
*expression-statement*
*selection-statement*
*iteration-statement*
*jump-statement*

[...]

(6.8.3) *expression-statement:*

*expression*$_{opt}$ `;`

(6.8.4) *selection-statement:*

`if` `(` *expression* `)` *statement*
`if` `(` *expression* `)` *statement* `else` *statement*
[...]

The abstract syntax is typically not specified as it is rather subject to the implementation of the compiler. One just picks the representation that suits one's needs best. When defining the semantics of arithmetic expressions next, we will have to cover every case of our BNF. If we do not have an extra case for parentheses, it makes our definition shorter and nicer.

[a]For a draft of the C11 specification see https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

## 1.2 Semantics

Now it is time to give meaning to our arithmetic expressions. There are different possibilities to do this. In this chapter, we will learn about two common kinds of semantics: **operational semantics** and **denotational semantics**.

### 1.2.1 Operational Semantics

When defining the semantics of a language, we relate its expressions to some kind of values, in our case the real numbers. We express the fact that an expression $\varphi$ evaluates to a value $v$ as $\varphi \triangleright v$. So for example $1 + (2 - 3) \triangleright 0$. But why does this hold? Well, because $2 - 3$ evaluates to $-1$, 1 evaluates to itself and $1 + (-1) = 0$. A little bit more general: when does $\varphi + \psi \triangleright v$ hold? Precisely when all of the following conditions hold:

(a) $\varphi \triangleright v'$

(b) $\psi \triangleright v''$

(c) $v = v' + v''$

When defining operational semantics, we essentially ask and answer questions like these. Note that all symbols in (c) as well as the $\triangleright$ are part of the meta-language. Only the left-hand side of $\triangleright$ we write expressions of our object language, the arithmetic expressions. So actually, we are defining addition of arithmetic expressions using the addition we have at the meta-level. This might seem pointless at first. However, imagine you are programming a calculator. Your programming language provides primitive operations such as addition to you. The arithmetic expressions you need to handle are a less primitive data structure. When writing the evaluation function—in other words: defining the semantics of arithmetic expressions—, you have to connect the arithmetic expressions to the primitive operations you have.

You might still wonder what exactly the difference between $=$ and $\triangleright$ is. While $\triangleright$ denotes "evaluates to," $=$ denotes equality. We can write $1 + 2 = 3$ because $1 + 2$ and $3$ represent the same number. And yes, we really mean numbers here, not expressions. However, if $1 + 2$ denoted an arithmetic expression instead (and 3 still meant the number), then we could not write $1 + 2 = 3$. This is because they do not even have the same type. It's like comparing apples with oranges.

Furthermore, $\triangleright$ is somehow tied to computation (we'll see in a moment how). This is not the case for equality. If you saw something like $1 + 2 = \_$ back in primary school, nobody (except your teacher, probably) would have stopped you from writing $42 - 39$.

But now, let's return to defining operational semantics. Because doing so in a textual form like above would be a bit tedious, computer scientists have invented the following notation:

$$\textsc{Add}\ \frac{\varphi \triangleright v' \qquad \psi \triangleright v'' \qquad v = v' + v''}{\varphi + \psi \triangleright v}$$

Such a beast is called an **inference rule**. Above the line, there are the **premises**, below there is the **conclusion**. If all premises hold, then the conclusion holds as well. If a rule has no premises, then its conclusion holds unconditionally. We call such rules **axioms**.

Having to write premises of the form $v = v' + v''$ is still more verbose than needed, instead we define our rules like this:

**Definition 1.17** (Operational Semantics of Arithmetic Expressions Without Variables).

$$\textsc{Val}\ \frac{}{n \triangleright n} \qquad \textsc{Add}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{\varphi + \psi \triangleright v + v'} \qquad \textsc{Sub}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{\varphi - \psi \triangleright v - v'} \qquad \textsc{Neg}\ \frac{\varphi \triangleright v}{-\varphi \triangleright -v}$$

$$\textsc{Mul}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{\varphi \cdot \psi \triangleright v \cdot v'} \qquad \textsc{Div}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{\varphi \div \psi \triangleright \frac{v}{v'}} \qquad \textsc{Exp}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{\varphi {}^\wedge \psi \triangleright v^{v'}}$$

We call such a collection of inference rules a **calculus**. This calculus might seem a little pointless now, especially the rule Val. But *n* and $n$ are not the same here: on the left-hand side of $\triangleright$, we have an arithmetic expression, while on the right-hand side, there is the corresponding real number (i.e. an element of $\mathbb{R}$). And without the Val rule, we would not know whether $1 + 1 \triangleright 2$: only the Add rule matches for our expression $1+1$, however it has two times the premise $1 \triangleright 1$. Besides Val, there is no rule to match the expression 1, so we would not be able to justify $1 \triangleright 1$ and ultimately $1 + 1 \triangleright 2$.

But now let's have a look at how to evaluate an expression:

**Example 1.18:** Evaluating an Arithmetic Expression

$$\textsc{Add}\ \frac{\textsc{Val}\ \frac{}{1 \triangleright 1} \qquad \textsc{Sub}\ \frac{\textsc{Val}\ \frac{}{2 \triangleright 2} \qquad \textsc{Val}\ \frac{}{3 \triangleright 3}}{2 - 3 \triangleright -1}}{1 + (2 - 3) \triangleright 0}$$

Note that if we work from bottom to top, we always consider the outermost operator first.

We call the evaluation as it is written above a **derivation tree**. After all, the evaluation has a tree-like structure: it has a root at the bottom and it is branching towards the top. If we read the tree from top to bottom, we have $2 \triangleright 2$ and $3 \triangleright 3$ at some point. Now using the rule Sub, we may *derive* $2 - 3 \triangleright -1$. So this is where the name comes from.

A derivation tree is only complete if all premises are justified by some rule. This means that at the top of the tree, there are usually axioms. However, sometimes there are side-conditions that are not subject to our calculus. For example, this is the case for $v = v' + v''$ from our first version of the Add rule. Then we just leave this part of the tree open (i.e. do not draw a line above the premise) and check the condition separately.

When evaluating a few expressions by yourself, you will notice that this is an entirely mechanical procedure. Indeed, one could implement the inference rules in a program and let the computer do the work for us. (This is exactly what a calculator does!) While the computer really needs this level of detail, it is a little ugly for us humans. Even evaluating small expressions can produce large derivation trees.

Can we use operational semantics to define the meaning of other languages? Of course we can.

Do you remember the language of (binary) trees in Checkpoint 1.6? If we are not interested in a nice graphical representation, we can use the following BNF:

**Definition 1.19** (Trees).

$$\mathcal{T} \ni \varphi, \psi ::= L \mid U\ \varphi \mid B\ \varphi\ \psi$$

$L$ stands for a leaf, $U$ for a node with only one child and $B$ for a node with two children. The **size** of a tree is defined as the number of nodes. We can formalize this definition as follows:

**Definition 1.20** (Size of a Tree).

$$\textsc{SLeaf}\ \frac{}{L \triangleright 1} \qquad \textsc{SUnary}\ \frac{\varphi \triangleright v}{U\ \varphi \triangleright 1 + v} \qquad \textsc{SBinary}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{B\ \varphi\ \psi \triangleright 1 + v + v'}$$

### Checkpoint 1.21: More (Operational) Tree Semantics

- What is the meaning of the following semantics?

$$\textsc{DLeaf}\ \frac{}{L \triangleright 1} \qquad \textsc{DUnary}\ \frac{\varphi \triangleright v}{U\ \varphi \triangleright 1 + v} \qquad \textsc{DBinary}\ \frac{\varphi \triangleright v \qquad \psi \triangleright v'}{B\ \varphi\ \psi \triangleright 1 + \max(v, v')}$$

- The breadth of a tree is the count of leaves. Give operational semantics for that.

Now our language of arithmetic expressions also has variables. But if we have an expression $x$, where should the value of $x$ come from? The idea is to use a so-called **environment** $\rho$, a function[5] that maps variables to values. We can denote an environment like this: $[x \mapsto 42, y \mapsto 0, z \mapsto \pi]$. The arrow $\mapsto$ reads as "maps to," so in this case $x$ is mapped to 42, $y$ is mapped to 0, and $z$ to $\pi$. To express that $\rho$ maps $x$ to 42, we also write $\rho(x) = 42$.

Now the evaluation of an expression depends on the environment. So we write the fact that an expression $\varphi$ evaluates to a value $v$ in an environment $\rho$ as $\rho \vdash \varphi \triangleright v$. The $\vdash$ sign is called "turnstile." Let us adjust the definition:

**Definition 1.22** (Operational Semantics of Arithmetic Expressions).

$$\textsc{Val}\ \frac{}{\rho \vdash n \triangleright n} \qquad \textsc{Var}\ \frac{\rho(x) = v}{\rho \vdash x \triangleright v} \qquad \textsc{Add}\ \frac{\rho \vdash \varphi \triangleright v \qquad \rho \vdash \psi \triangleright v'}{\rho \vdash \varphi + \psi \triangleright v + v'}$$

$$\textsc{Sub}\ \frac{\rho \vdash \varphi \triangleright v \qquad \rho \vdash \psi \triangleright v'}{\rho \vdash \varphi - \psi \triangleright v - v'} \qquad \textsc{Neg}\ \frac{\rho \vdash \varphi \triangleright v}{\rho \vdash -\varphi \triangleright -v} \qquad \textsc{Mul}\ \frac{\rho \vdash \varphi \triangleright v \qquad \rho \vdash \psi \triangleright v'}{\rho \vdash \varphi \cdot \psi \triangleright v \cdot v'}$$

$$\textsc{Div}\ \frac{\rho \vdash \varphi \triangleright v \qquad \rho \vdash \psi \triangleright v'}{\rho \vdash \varphi \div \psi \triangleright \frac{v}{v'}} \qquad \textsc{Exp}\ \frac{\rho \vdash \varphi \triangleright v \qquad \rho \vdash \psi \triangleright v'}{\rho \vdash \varphi^\wedge\psi \triangleright v^{v'}}$$

[5]Technically, this function is partial, meaning that it does not need to define the values of all possible variables. It is also finite.

This calculus isn't too different from our first one without variables: we add the Var rule where we look up the value of some value in the environment. Note that the lookup $\rho(x) = v$ is some kind of a side-condition. We cannot derive it from the rules of the calculus. So as mentioned above, we leave this premise open when drawing the derivation tree and just check the condition separately.

The second difference is that we carry around the environment in all other rules. Because we only have these two differences, the way in which we evaluate expressions does not change a lot, as can be seen in Example 1.23.

**Example 1.23:** Evaluating an Arithmetic Expression with Variables

Assume the environment $\rho := [x \mapsto 2, y \mapsto 1337, z \mapsto 52]$. We obtain:

$$\textsc{Add}\;\dfrac{\textsc{Mul}\;\dfrac{\textsc{Neg}\;\dfrac{\textsc{Var}\;\dfrac{\rho(x) = 2}{\rho \vdash x \triangleright 2}}{\rho \vdash -x \triangleright -2} \qquad \textsc{Val}\;\dfrac{}{\rho \vdash 5 \triangleright 5}}{\rho \vdash -x \cdot 5 \triangleright -10} \qquad \textsc{Var}\;\dfrac{\rho(z) = 52}{\rho \vdash z \triangleright 52}}{\rho \vdash -x \cdot 5 + z \triangleright 42}$$

Note that $\cdot$ and $-$ bind stronger than $+$, so there is no other way than first applying the Add rule (reading the tree from bottom to top).
Instead of writing $\rho$ in the tree, we could also have used $[x \mapsto 2, y \mapsto 1337, z \mapsto 52]$, but then we would probably have run out of space. If a task asks you to evaluate an expression in an environment $[a \mapsto 0, \ldots]$ it is always fine to give the environment a name first and use that name in the derivation tree.

**Checkpoint 1.24:** Adding Bindings to the Environment

Only reading from the environment is a bit boring. We extend the definition of arithmetic expressions by let-bindings:

$$\mathcal{E} \ni \varphi, \psi ::= \ldots \mid \text{let } x = \varphi \text{ in } \psi.$$

Given an environment $\rho$, let $x = \varphi$ in $\psi$ should evaluate $\varphi$ in $\rho$ first. Let's call the result $v'$. Then, $\psi$ should be evaluated in $\rho$ with $x$ bound to $v'$ and the result should be returned. So for example,

$$[a \mapsto 2, b \mapsto 1337] \vdash \text{let } b = 20 \cdot a \text{ in } a + b \triangleright 42.$$

We can denote "$\rho$ with $x$ bound to $v'$" as $\rho, x \mapsto v'$. In the example, we evaluate $a + b$ in

$$[a \mapsto 2, b \mapsto 1337], b \mapsto 20 = [a \mapsto 2, b \mapsto 20].$$

Come up with an inference rule for let-bindings.

If you are lazy about writing, you might ask whether you should use Definition 1.22 or Definition 1.17 to evaluate an expression when it does not contain any variables and no environment is given. Using Definition 1.22 would mean to always write $[] \vdash$ or $\emptyset \vdash$ in front of $\varphi \triangleright v$ (both $[]$ and $\emptyset$ denote the empty environment). The solution is: we consider $\varphi \triangleright v$ to be **syntactic sugar** for $[] \vdash \varphi \triangleright v$, so you can ignore Definition 1.17 from now on.

In case you wonder about the term "syntactic sugar": in the context of formal languages, it is useful to distinguish the **core language** and **derived constructs**. For each construct of the core language, you need to specify semantics, for instance in the shape of an inference rule. Now if you want to prove some property of your language, you often need to consider all inference rules. Having fewer inference rules shortens your proof. However, a smaller core language can be inconvenient to work with. Thus, such core languages are often cleverly designed such that the core language can encode more powerful construct without having to amend its definition. Such transformations at the syntactic level are what we call syntactic sugar.

**Example 1.25:** Subtraction as Syntactic Sugar

Our arithmetic expressions have addition, subtraction, and negation. If we wanted to keep the language minimal, we might notice that $a - b = a + (-b)$. Thus, we could remove subtraction from our arithmetic expressions without really changing the expressive power of that language. We could then define the following syntactic sugar.

$$e_1 - e_2 := e_1 + (-e_2)$$

This then means that we can simply write $a - b$, while knowing that what is actually written there is $a + (-b)$.

**Going Beyond:** Small-Step Semantics

You might imagine that one can use the operational semantics we have seen to define the semantics of programming languages. However, there might be an issue depending on the kind of programming language we are trying to define: we do not specify an evaluation order for the operands of binary operators. In our case, this does not matter since we do not have any language construct with side effects. Let us change this by adding a function $print(s)$ that prints the given string and returns the number of characters printed. Now we would like $print(\text{"Hello "}) + print(\text{"World!"})$ to evaluate to 12 with "Hello World!" printed (and not "World!Hello ", which we would obtain in case of right-to-left evaluation order). Instead of specifying how to obtain the overall result of an expression, we now define single steps of computation. We also need to keep track of what has been printed so far, so we write our computation steps like this:

$$(\varphi, \rho, o) \succ (\varphi', \rho', o')$$

On the left-hand side of $\succ$, we have a triple of the expression, the environment and the output before the computation step. On the right-hand side, there is the corresponding triple after the computation step. Our semantics now looks like this:

$$\textsc{Print}\ \frac{}{(print(s), \rho, o) \succ (|s|, \rho, o \mathbin{+\!\!+} s)} \qquad \textsc{Var}\ \frac{\rho(x) = v}{(x, \rho, o) \succ (v, \rho, o)} \qquad \textsc{AddL}\ \frac{(\varphi, \rho, o) \succ (\varphi', \rho', o')}{(\varphi + \psi, \rho, o) \succ (\varphi' + \psi, \rho', o')}$$

$$\textsc{AddR}\ \frac{(\psi, \rho, o) \succ (\psi', \rho', o') \qquad v \text{ is a value}}{(v + \psi, \rho, o) \succ (v + \psi', \rho', o')} \qquad \textsc{Add}\ \frac{v = v' + v'' \qquad v', v'' \text{ are values}}{(v' + v'', \rho, o) \succ (v, \rho, o)}$$

$|s|$ denotes the length of $s$, $+\!\!+$ is the concatenation of two strings. We omit the other operators as the corresponding rules do not differ a lot from the three Add rules. However, there is no Val rule because no further computation steps are needed for values. Evaluating our example expression involves the following steps:

$$\begin{aligned}(print(\text{"Hello "}) + print(\text{"World!"}), [], \text{""}) &\succ (6 + print(\text{"World!"}), [], \text{"Hello "}) \\ &\succ (6 + 6, [], \text{"Hello World!"}) \succ (12, [], \text{"Hello World!"})\end{aligned}$$

For every step of computation there is a derivation tree justifying it, like this one:

$$\textsc{AddL}\ \frac{\textsc{Print}\ \dfrac{}{(print(\text{"Hello "}), [], \text{""}) \succ (6, [], \text{"Hello "})}}{(print(\text{"Hello "}) + print(\text{"World!"}), [], \text{""}) \succ (6 + print(\text{"World!"}), [], \text{"Hello "})}$$

Note that requiring $v$ to be a value in the AddR rule is crucial to enforce left-to-right evaluation order. If we omitted this, the AddR rule would have matched for the first computation step, so the evaluation order would have been non-deterministic.
The kind of semantics we just defined is called **small-step semantics** or **structural operational semantics** and—as the latter suggests—a subkind of operational semantics. What we have seen before is called **big-step semantics** or sometimes **natural semantics**.

### 1.2.2 Denotational Semantics

Now let us turn to the second kind of semantics, **denotational semantics**. The idea is that we describe what an expression means in terms of our meta-language. Because our meta-language is mathematics, denotational semantics are sometimes called *mathematical semantics* as well. Defining semantics for arithmetic expressions without variables is straightforward:

**Definition 1.26** (Semantics of Arithmetic Expressions Without Variables)**.**

$$\begin{aligned}
\mathcal{E}[\![n]\!] &:= n\\
\mathcal{E}[\![\varphi + \psi]\!] &:= \mathcal{E}[\![\varphi]\!] + \mathcal{E}[\![\psi]\!]\\
\mathcal{E}[\![\varphi - \psi]\!] &:= \mathcal{E}[\![\varphi]\!] - \mathcal{E}[\![\psi]\!]\\
\mathcal{E}[\![-\varphi]\!] &:= -\mathcal{E}[\![\varphi]\!]\\
\mathcal{E}[\![\varphi \cdot \psi]\!] &:= \mathcal{E}[\![\varphi]\!] \cdot \mathcal{E}[\![\psi]\!]\\
\mathcal{E}[\![\varphi \div \psi]\!] &:= \frac{\mathcal{E}[\![\varphi]\!]}{\mathcal{E}[\![\psi]\!]}\\
\mathcal{E}[\![\varphi^\wedge\psi]\!] &:= (\mathcal{E}[\![\varphi]\!])^{\mathcal{E}[\![\psi]\!]}
\end{aligned}$$

**Example 1.27:** Evaluating an Arithmetic Expression

$$\begin{aligned}
\mathcal{E}[\![5 + 4 - 2 \cdot 3]\!] &= \mathcal{E}[\![(5 + 4) - (2 \cdot 3)]\!]\\
&= \mathcal{E}[\![5 + 4]\!] - \mathcal{E}[\![2 \cdot 3]\!]\\
&= (\mathcal{E}[\![5]\!] + \mathcal{E}[\![4]\!]) - (\mathcal{E}[\![2]\!] \cdot \mathcal{E}[\![3]\!])\\
&= (5 + 4) - (2 \cdot 3) = 9 - 6 = 3
\end{aligned}$$

In the first step, we do not use any of the defining equations. We just add some parentheses to not mess up. In the second step, we evaluate the $-$ operator, which is the outermost. Note that there is no other way than doing this. One must not evaluate a subexpression like $5 + 4$ before that. As an exercise, evaluate this expression using operational semantics and relate the order of reduction steps.

What we do here is to define a function $\mathcal{E}$ which maps arithmetic expressions (without variables) to real numbers $\mathbb{R}$.[6] The name does not matter much, it does not necessarily have to coincide with the language's name. In fact, people tend to omit this name if there is only one semantics function in the context. The function's definition consists of seven **defining equations**. Observe that the function refers to itself in every defining equation except the first. Functions that refer to themselves are called **recursive**.

When defining a recursive function, we should be careful about a few aspects:

- We must not have two different definitions for the same argument. We also say that the defining equations must be **disjoint**.

[6]Technically, this mapping is partial, since $\frac{0}{0}$ (at the meta-level) is not defined. Similarly, $-1^{\frac{1}{2}} = \sqrt{-1} = i$ is a complex but not a real number. Then, of course, $\mathcal{E}[\![0 \cdot (0 \div 0)]\!]$ is undefined as well—we need all operands to be defined for the result to be defined.

- We should define the function for every argument. One also speaks of the defining equations to be **complete**.
- We should avoid infinite recursion. For example, infinite recursion would occur in this definition: $\mathcal{E}'[\![\varphi + \psi]\!] := \mathcal{E}'[\![1 + \psi]\!]$. If we had infinite recursion, then the function would be undefined for some arguments. In case of our denotational semantics, there is a convenient way to ensure that the recursion terminates: apply the function to subexpressions only—like we did in the definition above.

Note that using the semantics brackets $[\![\cdot]\!]$ is just some fancy notation. There is no particular reason to not use the notation for functions you already know (e.g. $f(n) := n, f(\varphi + \psi) := f(\varphi) + f(\psi)$ etc.). It is just conventional to use the semantics brackets, so we stick to this notation.

Let us now turn towards a more interesting example. Recall our BNF for the language of (binary) trees: $\mathcal{T} \ni \varphi, \psi ::= L \mid U\ \varphi \mid B\ \varphi\ \psi$. Using denotational semantics, we can now define the **size** of a tree as follows:

**Definition 1.28** (Size of a Tree)**.**

$$\begin{aligned}
\mathcal{T}_s[\![L]\!] &:= 1 \\
\mathcal{T}_s[\![U\ \varphi]\!] &:= 1 + \mathcal{T}_s[\![\varphi]\!] \\
\mathcal{T}_s[\![B\ \varphi\ \psi]\!] &:= 1 + \mathcal{T}_s[\![\varphi]\!] + \mathcal{T}_s[\![\psi]\!]
\end{aligned}$$

**Checkpoint 1.29:** More (Denotational) Tree Semantics

- What is the meaning of the following semantics?

$$\begin{aligned}
\mathcal{T}_{deg}[\![L]\!] &:= 0 \\
\mathcal{T}_{deg}[\![U\ \varphi]\!] &:= \max(1, \mathcal{T}_{deg}[\![\varphi]\!]) \\
\mathcal{T}_{deg}[\![B\ \varphi\ \psi]\!] &:= 2
\end{aligned}$$

- Remember that the breadth of a tree is the count of leaves. Give a semantics function for that.

Let us finally give semantics to variables. Again, we use an environment $\rho$. Our definition now looks as follows:

**Definition 1.30** (Semantics of Arithmetic Expressions)**.**

$$\begin{aligned}
\mathcal{E}[\![n]\!]_\rho &:= n \\
\mathcal{E}[\![v]\!]_\rho &:= \rho(v) \\
\mathcal{E}[\![\varphi + \psi]\!]_\rho &:= \mathcal{E}[\![\varphi]\!]_\rho + \mathcal{E}[\![\psi]\!]_\rho \\
\mathcal{E}[\![\varphi - \psi]\!]_\rho &:= \mathcal{E}[\![\varphi]\!]_\rho - \mathcal{E}[\![\psi]\!]_\rho \\
\mathcal{E}[\![-\varphi]\!]_\rho &:= -\mathcal{E}[\![\varphi]\!]_\rho \\
\mathcal{E}[\![\varphi \cdot \psi]\!]_\rho &:= \mathcal{E}[\![\varphi]\!]_\rho \cdot \mathcal{E}[\![\psi]\!]_\rho \\
\mathcal{E}[\![\varphi \div \psi]\!]_\rho &:= \mathcal{E}[\![\varphi]\!]_\rho \div \mathcal{E}[\![\psi]\!]_\rho \\
\mathcal{E}[\![\varphi \hat{\ } \psi]\!]_\rho &:= (\mathcal{E}[\![\varphi]\!]_\rho)^{\mathcal{E}[\![\psi]\!]_\rho}
\end{aligned}$$

Compared to our initial definition, there are just two changes. First, we add a rule for variables, of course. And secondly, we carry the environment $\rho$ around.

**Example 1.31:** Evaluating an Arithmetic Expression with Variables

Assume the environment $\rho := [x \mapsto 2, y \mapsto 1337, z \mapsto 52]$. We obtain:

$$\begin{aligned}
\mathcal{E}[\![-x \cdot 5 + z]\!]_\rho &= \mathcal{E}[\![((-x) \cdot 5) + z]\!]_\rho \\
&= \mathcal{E}[\![(-x) \cdot 5]\!]_\rho + \mathcal{E}[\![z]\!]_\rho \\
&= (\mathcal{E}[\![-x]\!]_\rho \cdot \mathcal{E}[\![5]\!]_\rho) + \rho(z) \\
&= ((-\mathcal{E}[\![x]\!]_\rho) \cdot 5) + 52 \\
&= ((-\rho(x)) \cdot 5) + 52 \\
&= ((-2) \cdot 5) + 52 = -10 + 52 = 42
\end{aligned}$$

## 1.3 Summary

In this chapter, we have learned how to define the syntax and semantics of formal languages. We have seen BNFs as a short form to specify the syntax and know that an expression has a tree-like shape. That is why we also speak of syntax trees. We gave precedence rules for arithmetic expressions such that we can write them in a linearized shape without needing too many parentheses. We specified the meaning of languages with both operational semantics (or more precisely big-step semantics) and denotational semantics. When defining the semantics of programming languages, one often uses operational semantics, but sometimes denotational semantics are involved as well. Even though we are not about to define a programming language in this book, we will use BNFs and denotational semantics again, for example when we introduce propositional logic in the next chapter.

# 2 | Logic

## 2.1 Propositional Logic

When asked, which skill is most important for future computer scientists, a common answer is thinking logically or rationally. That's what we will look at now.

We start by introducing you to a simple formalism called *propositional logic*. Propositional logic studies the logical relationships between propositions, which are a special kind of statement.

Logic is studied and used differently in different disciplines. The original use case of logic in philosophy is for evaluating (i.e. checking the validity of) arguments. By argument, we do not mean to denote the act of convincing somebody of a proposition, but rather the actual "spoken words" said when trying to convince someone. Basically, an argument is a written explanation of why something is true.

In some sense, logic also guides us towards what we ought to believe. For example, if Kurt believes both that there will be Schnitzel in Mensa whenever the current day is a Thursday, and he believes that it is Thursday, we would consider it irrational if he were not to believe that Schnitzel is served in Mensa today. In general, it is hard to spell out the exact connection between formal logical inferences and normative consequences arising from these; however, it is uncontroversial that there is some link between the two.

Lastly, logic is not about how we actually reason, but rather about how we would do so in an idealized world. We want to obtain correct proofs and not (at least not primarily) explanations why we reasoned in the wrong fashion.

In a more formal sense, logic allows us to analyze how certain statements are related by consequence, that is, which statements follow from which other statements.

**Chapter Goals**

In this chapter, we introduce propositional logic, which is a very basic logic. In particular, we

- discuss what mathematical statements or propositions are
- define the syntax and semantics of propositional logic
- analyze the reasoning principles of propositional logic

Without formal training, it is easy to make mistakes during logical inferences. This is highlighted by the Wason selection task (Checkpoint 2.1), which can quickly assess someone's logical inference skills without requiring them to know anything about logic. Interestingly, most people fail the task.

**Checkpoint 2.1:** Wason Selection Task

Consider the following task: Four two-sided cards lay on a table before you, one side of the cards contains numbers and the other contains letters. Only one side is currently visible to you. You can see the following:

- The first card has the letter 'a.'
- The second card has the number 4.
- The third card has the letter 'z.'
- The fourth card has the number 9.

Imagine a stranger tells you that "The number on a card is even only if the letter on the other side of the card is a vowel." Which cards do you need to turn over in order to check whether that rule is true?

### 2.1.1 Statements and Propositions

At the core of logic is the analysis of certain statements, so-called propositions.

> **Definition 2.2** (Proposition)**.** *A **proposition** is a statement which is definitely either true or false. We say that its **truth value** is true if the statement is definitely true and that it is false if it is definitely false.*

An important concept for our study is recognizing that propositions come in different forms and that some propositions are made up of sub-propositions. We now introduce a term for propositions not containing any sub-statements.

> **Definition 2.3** (Atomic Proposition)**.** *An **atomic proposition** is a proposition whose truth or falsity is not dependent on any other proposition. $\mathcal{A}$ is the collection of atomic propositions (so we write $a \in \mathcal{A}$ if $a$ is an atomic proposition).*

In logic, the only statements we care about are propositions. Thus, we often talk about **statements** when we actually mean propositions. The difference between statements and propositions is rather subtle and discussed in the respective going beyond box.

One particularly important atomic proposition is the proposition that is always true. This proposition is also called **truth**. Similarly, the proposition that is always false is called **falsity**.

Note that truth (the proposition) and truth (the opposite of a lie) are not actually the same, even if they share a name.

### 2.1.2 Operators

We focus on a specific class, the **truth-functional operators**. An operator is said to be truth-functional if the truth values of statements constructed using this operator only depend on the

**Going Beyond:** Statement vs. Proposition

For our purposes, it suffices to treat proposition and statement as synonyms. However, in the philosophy of language, a distinction between them is made: Statements have a propositional content, so that the term *proposition* is used to refer to something abstract, namely the thing that two statements with the same meaning are said to express. Thus, the sentences *Kleene likes Turing* and *Turing is liked by Kleene* express the same propositions, although as statements they are different.

**Example 2.4:** Atomic Propositions

These are all atomic propositions:

- Saarbrücken is in Saarland.
- Saarbrücken is not in the Saarland.
- $0 \leq 2$
- This book has 42 pages.
- The reader's favorite color is blue.
- $2 + 2 = 5$

These are not, since they either not atomic or not propositions:

- Blue is the best color.
- Saarbrücken is beautiful.
- Paris is ugly.
- Saarbrücken is in Saarland and Paris is in France.
- This statement is false.

truth values of the sub-statements to which the operator is applied.

This concept is probably easier to grasp using an example. We first consider an operator that is not truth-functional. Consider the unary operator *It is possible that* $\varphi$. Now consider the sentences

- The official language of Germany is English.
- A bachelor is married.

Obviously, both statements are false, the official language in Germany is German and bachelors are by definition unmarried men, thus no bachelor is married. But now consider what happens when we add the modifier:

- It is possible that the official language of Germany is English.
- It is possible that a bachelor is married.

The second sentence is still false since our use of the word *bachelor* stays the same.[1] However, the first one is now true (Germany could, at any point, pass a law making English the official language), which shows that possibility is not truth-preserving—the resulting truth value is different even though the truth value of the contained statement is the same.

[1]The sentence *All bachelors are unmarried* is a famous example of an analytical proposition—a proposition which is true by virtue of its meaning.

From now on, we limit ourselves to truth-functional operators. Fortunately, almost all logical operators used in mathematics are truth-functional. The next sections present the most common ones.

### 2.1.3 Syntax and Semantics

Without further ado, we present the syntax of propositional logic.

**Definition 2.5.** *The language of propositional logic $\mathcal{F}_0$ is defined using the BNF:*

$$\mathcal{F}_0 \ni \varphi, \psi ::= a \mid \top \mid \bot \mid \neg\varphi \mid \varphi \land \psi \mid \varphi \lor \psi \mid \varphi \to \psi$$

*where $a \in \mathcal{A}$, usually denoted as a truth variable.*

When working with propositional logic, our atomic propositions are usually represented as **truth variables**. This has two purposes: First, it shortens our formulas, since we usually abbreviate an informally defined proposition like “Saarbrücken is in Saarland” by a single letter, like $s$. Secondly, it allows us to analyze formulas where we do not really care about the concrete atomic propositions, to better understand how these operators work in general.

**Example 2.6:** Formulas of Propositional Logic

All these are formulas of propositional logic:

- $\top \lor \bot$
- $a \to (\neg b)$
- $s \land (b \lor \bot)$
- $\bot \to ((a \land b) \lor ((\neg a) \land (\neg b)))$

Their semantics depend on the concrete atomic propositions for $a$, $b$, and $s$, which we leave unspecified.

The binary operators are listed such that the strongest binding operator comes first. $\land$ and $\lor$ are left-associative, and $\to$ is right-associative.

**Definition 2.7** (Precedence Rules for Propositional Logic)**.**

| *Operator* | *Precedence level* | *Associativity* |
|---|---|---|
| $\neg$ *(unary)* | *3* | — |
| $\land$ | *2* | *left* |
| $\lor$ | *1* | *left* |
| $\to$ | *0* | *right* |

#### Semantics

Let’s look at what these symbols are supposed to mean. All of them are truth-functional operators we already know from natural language. We first informally describe what these operators are supposed to mean, and then define a formal semantics.

**Example 2.8:** Fully Parenthesized Propositional Formulas

If the precedence rules are unclear, the following examples might help.

- $a \wedge b \wedge c \;=\; (a \wedge b) \wedge c$
- $a \vee b \wedge c \;=\; a \vee (b \wedge c)$
- $a \wedge b \to c \vee d \;=\; (a \wedge b) \to (c \vee d)$
- $\neg\neg a \wedge b \;=\; (\neg(\neg a)) \wedge b$
- $a \to b \to c \to d = a \to (b \to (c \to d))$

Before we get to the operators itself, we have $\top$ (read "top," or "truth") and $\bot$ (read "bottom," or "falsity"). $\top$ denotes the atomic proposition that is always true, and $\bot$ denotes the atomic proposition that is always false.

The first is **conjunction** $\wedge$. It is usually called "and" in informal language, where we read $\varphi \wedge \psi$ as "$\varphi$ holds and $\psi$ holds." A conjunction of two statements is true precisely when the left and right statement are both true. Otherwise, it is false.

The next is **disjunction** $\vee$. It is usually called "or." However, "or" in natural language is often used ambiguously. If you're at a logician's party and are asking the host for "a beer or a coke," the host might just bring you a beer and a coke. This is because in propositional logic, $\varphi \vee \psi$ is true as long as at least one of the sub-statements is true, i.e. if the left statement, the right statement, or both of them are true. This is also called **inclusive or**, to make this precise. There also is exclusive or, which we will discuss later.

The last binary operator is **material implication**, written $\to$ and often simply called **implication**. In natural language, this roughly corresponds to "if." In an implication $\varphi \to \psi$, we typically call $\varphi$ the **antecedent** (also called **precondition**), while $\psi$ is the **consequent**. We would read such an implication as "If $\varphi$, then $\psi$" or as "$\psi$ if $\varphi$." However, a logician's conception of implication might be rather different from the way "if" is used in everyday language. We will explain it later.

Lastly, we have **negation** $\neg\varphi$. Negation in propositional logic denotes the opposite statement, i.e. the statement is true when the original statement is false. If $a$ := "Saarbrücken is in Saarland," then $\neg a$ represents the statement that "Saarbrücken is not in Saarland."

Note that negation in natural language is often tricky: The opposite meaning of "You must clean your room" is *not* "You must not clean your room". The latter forbids cleaning one's room, however the opposite meaning of being required to clean one's room is just that the requirement does not exist. Thus, the logical negation is more aptly captured by "You are not required to clean your room."

This is another reason why logicians avoid using normal English, and instead invented propositional formulas: There, inserting a negation always precisely denotes *the opposite*.

We now develop a semantics that allows us to formally figure out whether a formula of propositional logic is true or false. First, we define the two possible truth values:

**Definition 2.9** (Truth Values)**.** *Truth values are given by the grammar* $\mathcal{B}$ ::= `true` | `false`.

Basically, when we mean that some formula $\varphi$ is true (in some environment), we say that it has truth value `true`, and similarly `false` if it is false.

Now, the truth value of a propositional formula depends on the truth values of the atomic propositions used in the formula. Since these can be anything, their truth value must be chosen by us before we start evaluating the truth value of a formula in propositional logic. Formally, this choice is represented as an environment $\rho : \mathcal{A} \to \mathcal{B}$, i.e. the environment maps atomic propositions to truth values.

The formal semantics is now given by an evaluation function.

**Definition 2.10** (Truth Value Semantics)**.** *Given a propositional logic formula $\varphi$, and an environment $\rho$ which assigns truth values to truth variables / atomic propositions, the* ***evaluation*** *$\mathcal{B}[\![\varphi]\!]_\rho : \mathcal{B}$ of $\varphi$ under $\rho$ is defined by structural recursion on $\varphi$:*

$$\begin{aligned} \mathcal{B}[\![a]\!]_\rho &= \rho\, a \\ \mathcal{B}[\![\top]\!]_\rho &= \texttt{true} \\ \mathcal{B}[\![\bot]\!]_\rho &= \texttt{false} \end{aligned}$$

*The other cases contain sub-expression. There, the evaluation is defined by case analysis on the evaluation of the sub-expressions, as indicated by the following truth table:*

| $\mathcal{B}[\![\varphi]\!]_\rho$ | $\mathcal{B}[\![\psi]\!]_\rho$ | $\mathcal{B}[\![\varphi \land \psi]\!]_\rho$ | $\mathcal{B}[\![\varphi \lor \psi]\!]_\rho$ | $\mathcal{B}[\![\varphi \to \psi]\!]_\rho$ | $\mathcal{B}[\![\neg\varphi]\!]_\rho$ |
|---|---|---|---|---|---|
| `true` | `true` | `true` | `true` | `true` | `false` |
| `true` | `false` | `false` | `true` | `false` | |
| `false` | `true` | `false` | `true` | `true` | `true` |
| `false` | `false` | `false` | `false` | `true` | |

So far, we have considered our atomic propositions to be concrete statements, like “Saarbrücken is in Saarland.” We could now figure out whether $a \to a$ is true, where $a$ is that concrete statement. If we are to understand the laws of logic, this is too limiting. We want to know whether a law like $a \to a$ is always true, irrespective of whatever we chose for $a$. This corresponds to enumerating all possible truth values for all atomic propositions used in our formula. Formally, this means evaluating a formula under all possible environments. For this, we usually use a proof table, and then consider all the sub-formulas, evaluating them from the inside out (going up the syntax tree, starting at the leaves). In practice, we often omit the evaluation brackets $[\![\cdot]\!]_{\_}$ in a truth table, especially if the environment is clear from the context or defined by the truth table itself, like in Example 2.11.

Often, we care not just about whether a formula is true in some environment or another, but about its behavior in all possible environments. For example, $\top$ is true in all environments. Similarly, $\bot$ is false in all environments. Yet, $a \land b$ is true in some environments and false in others. To classify these cases, we have the following definitions.

**Definition 2.12** (Validity, Satisfiability)**.** *A formula $\varphi \in \mathcal{F}_0$ is called*

- ***valid*** *iff it evaluates to* `true` *under all possible environments.*
- ***satisfiable*** *iff it evaluates to* `true` *under at least one environment.*
- ***contradictory*** *iff it evaluates to* `false` *under all possible environments.*
- ***refutable*** *iff it evaluates to* `false` *under at least one environment.*

**Example 2.11:** Truth Tables

We evaluate the formula $a \land \neg b \to c \land \top$ under all possible environments:

| $a$ | $b$ | $c$ | $\neg b$ | $a \land \neg b$ | $\top$ | $c \land \top$ | $a \land \neg b \to c \land \top$ |
|---|---|---|---|---|---|---|---|
| false | false | false | true | false | true | false | true |
| false | false | true | true | false | true | true | true |
| false | true | false | false | false | true | false | true |
| false | true | true | false | false | true | true | true |
| true | false | false | true | true | true | false | false |
| true | false | true | true | true | true | true | true |
| true | true | false | false | false | true | false | true |
| true | true | true | false | false | true | true | true |

If we now consider a concrete environment $\rho := [a \mapsto \texttt{true}, b \mapsto \texttt{false}, c \mapsto \texttt{false}]$, we can immediately read that $[\![a \land \neg b \to c \land \top]\!]_\rho = \texttt{false}$ by looking at the correct line in our truth table.

Thus, the formula $\top$ is valid. Note that it is also satisfiable. Similarly, $\bot$ is contradictory. Both it and $\varphi \land \neg\varphi$ are refutable.

For now, we define that a formula is **true** iff it is valid. We can figure out whether a formula is true by just enumerating all possible environments and checking that its truth value is `true` under all of them. Note that this can be very time-consuming, especially for formulas using many variables. For example, the formula $a \land b \land c \land d \equiv d \land c \land b \land a$ is obviously true, but a truth table would take 16 rows and 12 columns.

**Checkpoint 2.13:** Validity, Satisfiability

We can see in Example 2.11 that the formula $a \land \neg b \to c \land \top$ is satisfiable and refutable. Similarly determine whether the following formulas are valid, satisfiable, contradictory, or refutable. Compute the truth table in all cases.

- $a \land \neg a$
- $a \lor \neg a$
- $a \land b \to b \land a$
- $\neg a \to a$
- $\neg\neg a \to a$
- $\neg a \to a \to \bot$

## 2.1.4 Implication and Equivalence

So far, we have delayed an intuitive explanation of how (material) implication works. We will tackle this now.

First, note that implication, as defined in Definition 2.10, is a truth-functional operator. In colloquial speech, a statement like “If I fall down the stairs, I get hurt” implies a deep causal relation between falling and getting hurt. An implication like “If my sister falls down the stairs, I get hurt” seems nonsensical, since it's supposed to be the sister that gets hurt. However, implication needs to be truth-functional, which means that we can not capture any deeper connections between actions. Thus, if both me and my sister fall down the stairs, and I get hurt, both implications should have the same truth value since all sub-expressions have the same truth value. We now construct the operator that best resembles our intuitive understanding of "if this, then that."

Let's say that your logician friend Dieter tells you: "On Monday mornings, I am very tired." Since Dieter is a logician, he rather phrases it like this: "If it is a Monday morning, I am very tired." Let's try to analyze what Dieter is actually telling us.

If it actually is a Monday morning, then Dieter should actually be very tired. If Dieter is well awake on such a morning, we would consider his statement a lie. Since Dieter is a good friend who never lies, we know that what he tells us is always true. So, if we have an implication we know is true, and we know that the antecedent (It is a Monday morning) holds, then we can conclude that the consequent also holds. So far, this is expected.

Let's repeat our scenario on any other day of the week. Note that now, we can no longer conclude whether Dieter is tired or fully awake. Dieter could be rather tired on all mornings, irrespective of the day of the week. Alternatively, Dieter could just hate Mondays. His statement does not allow us to draw any conclusions about his mental state on Tuesdays, since he only talked about Mondays.

Looking at the formalism, this tells us that an implication can be true, even if the antecedent is wrong. In fact, if the antecedent is wrong, the implication must always be true. Let's make this explicit by considering alternative definitions of implication (denoted as $\overset{1}{\to}$ to $\overset{3}{\to}$):

| $\varphi$ | $\psi$ | $\varphi \to \psi$ | $\varphi \overset{1}{\to} \psi$ | $\varphi \overset{2}{\to} \psi$ | $\varphi \overset{3}{\to} \psi$ |
|---|---|---|---|---|---|
| `true` | `true` | `true` | `true` | `true` | `true` |
| `true` | `false` | `false` | `false` | `false` | `false` |
| `false` | `true` | `true` | `false` | `true` | `false` |
| `false` | `false` | `true` | `true` | `false` | `false` |

If implication was defined like $\overset{1}{\to}$, then we would also know that if Dieter was very tired, it would be a Monday morning. This is clearly not what Dieter said,[2] since Dieter can also be tired on other days of the week.

If we would instead use the implication $\overset{2}{\to}$, then we would know that Dieter is tired on all days. Again, this is not what Dieter meant—he simply meant to tell us that he is tired on Mondays, and did not tell us anything at all about the other days of the week.

Finally, candidate $\overset{3}{\to}$ would force us to deduce that Dieter is tired and that it is always a Monday morning. This is plainly absurd since there are other days in the week. Note that candidate 3 is just conjunction.

In summary, implication should be understood through the lens of *If we know $\varphi \to \psi$ is true, what do we know about $\psi$ depending on $\varphi$?* In other words, try to understand implication by considering how it can be used to deduce other facts.

With this, we can turn to two alternative terms often used to describe implications:

[2]It can also not be what Dieter meant. Dieter is a logician and always says precisely what he means.

> **Definition 2.14** (Sufficient and Necessary Conditions)**.** *In an implication $\varphi \to \psi$, we say that*
>
> - *$\varphi$ is **sufficient for** $\psi$*
> - *$\psi$ is **necessary for** $\varphi$*

Again consider our friend Dieter, who is very tired if it is a Monday morning.

There, it being a Monday morning is sufficient for Dieter to be tired. This is because once we know that it is a Monday morning, we can simply conclude that Dieter is tired. Nothing more needs to be checked, our current knowledge about Dieter and the world is enough—it is *sufficient.*

Conversely, if we know that Dieter is tired, we can of course not conclude that it is a Monday. However, it is still possible that it indeed is a Monday. If Dieter had instead woken up well-rested, we would already know that today can not be a Monday. Thus, Dieter being tired is necessary for it to be a Monday since we could otherwise just summarily reject this possibility.

Note that these statements appear "backwards," since they make it seem like it can not be a Monday *because* Dieter is awake, while in reality, Dieter does not have the magic ability to change the day of the week. However, if you know how to spot it, you can notice that we use reasoning like this all the time. For example, Dieter sometimes wakes up in the middle of the night. At first, he is shocked since him waking up on his own, without an alarm, usually means that he overslept. However, once he sees that it's still dark outside, Dieter is relieved. This is because Dieter knows that if he oversleeps, the sun would already have risen. Thus, he can do the same "backwards inference" to realize that he did not oversleep, without even looking at his alarm clock. All of this works precisely because the sun rising is *necessary* for Dieter to oversleep.

To make things even more confusing, a backwards "if" is often called "only if." Thus, an expression like "$a$ only if $b$" is to be parsed as $b \to a$. Usually, this is used when the "backwards inference" aspect of the implication is stressed. In practice, it often negatively affects clarity and should be avoided.

Now is a good time to remind you that implication is right-associative. This means that an implication $a \to b \to c$, fully parenthesized, reads $a \to (b \to c)$. A naive verbalization of this is "If $a$, then, if $b$, then $c$." Note that this implies that we impose two preconditions on $c$, namely both $a$ and $b$. It turns out that $a \to b \to c$ and $(a \land b) \to c$ are equivalent–that is, they have the same formal meaning.

**Equivalence** Out of the "wrong implications," candidate 1 was the most interesting: It allowed us to express that something is true exactly when something else is true. Since this is very useful, it has a special name and a special symbol:

> **Definition 2.15** (Logical Equivalence)**.** *Two formulas $\varphi, \psi \in \mathcal{F}_0$ are **equivalent**, written $\varphi \leftrightarrow \psi$, precisely when*
>
> $$(\varphi \to \psi) \land (\psi \to \varphi)$$
>
> *This kind of equivalence based on material implication is also known as **material equivalence** or **logical equivalence**.*

Note that the truth table induced by this operation is precisely that of $\overset{1}{\to}$.

The formula $\varphi \leftrightarrow \psi$ can be read as "$\varphi$ and $\psi$ are equivalent," although this is rather uncommon. Instead, we usually say that "$\varphi$ if and only if $\psi$" since this captures that either implies the other. On paper, this is often abbreviated to "**iff**," which should not be confused with "if."

You might have come across "iff" in several definitions already. When defining some property or statement that is true or false, we usually express this using "iff," following a schema similar to this:

A day $d$ is called a **working day** *iff* it is Monday, Tuesday, Wednesday, Thursday or Friday.

Logical equivalence is an operator in propositional logic, defined using syntactic sugar. It allows us to reason about the equivalence of two statements within the framework of propositional logic.

Opposite to this, there is the meta-level notion of semantic equivalence:

> **Definition 2.16** (Semantic Equivalence)**.** *Two formulas $\varphi, \psi \in \mathcal{F}_0$ are* ***semantically equivalent*** *written $\varphi \equiv \psi$, iff for all possible environments $\rho$, $[\![\varphi]\!]_\rho = [\![\psi]\!]_\rho$.*

This is a meta-level definition, since it is defined by referencing concepts needed to define propositional logic itself (namely evaluation), instead of being defined *within* propositional logic. More on the difference between the object- and meta-level can be found in the respective going beyond box.

**Going Beyond:** Object- and Meta-Level Logics

When doing logic, one has to distinguish between object- and meta-level logics. You already know this distinction from the first chapter, where we had the object- and the meta-language. Let's have a closer look at why we need this distinction, and what we gain by having two different levels of logic.
For reference, object level logic is the logic one is currently studying. Here, it is propositional logic. Meta-level logic is what we use to describe how the object-level logic works. In this book, it is English, along with a colloquial understanding of words like "and" or "if and only if."
In the meta-level logic, we are often able to express things we can not express in the object-level logic. For example, in plain propositional logic, we have no notion of environments, the collection of all formulas, and so on. However, we have a very precise understanding of implication.
We study special object-level logics to better understand how to think about mathematics at the meta-level. For example, we now better understand what logicians and mathematicians mean when they say "if and only if"—namely something corresponding to the logical equivalence of propositional logic.
While $\longrightarrow$ and $\leftrightarrow$ belong to the object level, we can use $\Rightarrow$ and $\Leftrightarrow$ to express the corresponding concept in the meta-level. By this, we make explicit that the meta-level concept of "If, then" or "If and only if" works like implication and logical equivalence in propositional logic.

For example, the formula $a \leftrightarrow b$ is satisfiable, which means that there is at least one environment for which $a$ and $b$ both imply each other. However, there can be other environments where this does not hold. If they were semantically equivalent, they would be equivalent in every environment, which they clearly are not.

Semantic and logical equivalence have a rather deep connection:

**Theorem 2.17.** *Given any two formulas $\varphi, \psi \in \mathcal{F}_0$, we have that $\varphi \leftrightarrow \psi$ is valid if and only if $\varphi \equiv \psi$.*

*Proof.* We prove a meta-level implication by proving both directions separately.

- $\Rightarrow$ We know $\varphi \leftrightarrow \psi$ is valid in all environments, and need to show that for any environment $\rho$, $[\![\varphi]\!]_\rho = [\![\psi]\!]_\rho$. This holds since $[\![\varphi]\!]_\rho$ and $[\![\psi]\!]_\rho$ are either both `true` or both `false` in $\rho$, since the equivalence is also `true` in $\rho$.
- $\Leftarrow$ Since $\varphi$ and $\psi$ both evaluate to the same value $b$ in $\rho$, the equivalence $\varphi \leftrightarrow \psi$ must also hold in $\rho$. $\square$

This means that two formulas $\varphi, \psi$ are semantically equivalent if the formula $\varphi \leftrightarrow \psi$ is true (unconditionally).

Similarly, we can characterize validity using semantic equivalence.

**Lemma 2.18.** *A formula $\varphi \in \mathcal{F}_0$ is valid (or true) iff $\varphi \equiv \top$. Similarly, it is contradictory iff $\varphi \equiv \bot$.*

This means that we can prove that a formula is true by just proving that it is semantically equivalent to $\top$. In some cases, this might be easier than drawing the entire proof table.

At this point, you might wonder: Why are there two kinds of equivalences (semantic and logical)? The answer is that logical equivalence allows us to express that something is equivalent *within the logic itself.* This is a weaker equivalence than semantic equivalence, which requires that formulas are equivalent *in all environments*. To understand this, let us again consider Dieter as an example. If Dieter goes to bed late, then he oversleeps the next morning. Similarly, if Dieter oversleeps, then this is clearly because he went to bed too late. If we define $a :=$ Dieter oversleeps, and $b :=$ Dieter went to bed too late, we might pose that $a \leftrightarrow b$, or even $a \equiv b$.

However, Dieter is not stupid. Dieter has now bought an alarm clock, to wake him up even if he goes to bed too late. Thus, the relationship between oversleeping and going to bed late is no longer as simple: It only holds if Dieter does not own an alarm clock. If we let $c :=$ Dieter has an alarm clock, we can note that $\neg c \longrightarrow (a \leftrightarrow b)$. However, $a$ and $b$ are no longer semantically equal, since there is the possibility (if $c :=$ `true`), that $a$ is false (Dieter wakes up on time) but $b$ is still true (he went to bed late). The statement $\neg c \longrightarrow (a \leftrightarrow b)$ is still true, however, since there the equivalence is “limited” to when no alarm clock is present.

Note that we used the logical equivalence $a \leftrightarrow b$ as part of another, larger propositional formula. This is possible because $a \leftrightarrow b$ is a formula itself (remember that it is defined as $a \longrightarrow b \wedge b \longrightarrow a$). However, we can not write $\neg c \longrightarrow (a \equiv b)$ because $a \equiv b$ is not a propositional formula. Instead, it is a statement of the *meta-logic*, that can not be used in propositional formulas. Besides, it is also not clear what $\neg c \longrightarrow (a \equiv b)$ even if supposed to mean. A naive reading suggests “If not $c$, then $a$ and $b$ are equivalent in all environments,” which already sounds confusing, since what does “all environments” even mean if we assume $c$ to be false? Luckily, since we forbid such formulas, we do not have to think about this further.

This shows that both version have their uses: Logical equivalence can be used to describe two things as equivalent *under certain circumstances*, where we can use propositional formulas to describe the other circumstances. Semantic equivalence is used to describe that something is always equivalent, no matter the circumstances. This is much more powerful, as we will see when we formulate laws using it in Lemma 2.22.

This also explains Theorem 2.17: When we know the formula $a \leftrightarrow b$ is true *without any preconditions*, this is the same as semantic equivalence $a \equiv b$. But of course, the point of having $a \leftrightarrow b$ is that we can put additional conditions on it.

**Exclusive Or** Consider the truth table for the negation of logical equivalence:

| $\varphi$ | $\psi$ | $\varphi \leftrightarrow \psi$ | $\neg(\varphi \leftrightarrow \psi)$ |
|---|---|---|---|
| true | true | true | false |
| true | false | false | true |
| false | true | false | true |
| false | false | true | false |

This operator also has a special name: **exclusive or**. This is in contrast to the usual disjunction, which is **inclusive**. For historical reasons, it is usually not defined via logical equivalence, but like this:

**Definition 2.19** (Exclusive Or). *The exclusive or of two formulas $\varphi$, $\psi$, also written as $\varphi \oplus \psi$, is defined as follows.*

$$\varphi \oplus \psi := (\varphi \land \neg\psi) \lor (\neg\varphi \land \psi)$$

When "or" is used colloquially, it can often have a meaning that is closer to exclusive or than to inclusive or. Thus, when trying to model certain logical connections, one needs to carefully think about which variant is actually meant.

**Rules Around Syntactic Sugar** In Definition 2.5, we defined the core syntax of propositional logic. This included the operators of conjunction ($\land$), disjunction ($\lor$), and implication ($\rightarrow$). We then defined exclusive or ($\oplus$) and equivalence ($\leftrightarrow$) as syntactic sugar, by simply using the already existing operators. While this definition is precise, it still leaves open one question: What is the precedence level of these operators? If we write $a \land b \oplus c$, should this be parsed as $(a \land b) \oplus c$ or as $a \land (b \oplus c)$. To answer this question, we need to also define the precedence levels of our syntactic sugar. Here they are:

**Definition 2.20** (Precedence Rules for Propositional Logic with Syntactic Sugar).

| *Operator* | *Precedence level* | *Associativity* |
|---|---|---|
| $\neg$ *(unary)* | *3* | — |
| $\land$ | *2* | *left* |
| $\lor$ | *1* | *left* |
| $\oplus$ | *0.5* | *none* |
| $\rightarrow$ | *0* | *right* |
| $\leftrightarrow$ | *-1* | *none* |

Note that they are neither left- nor right-associative. This means that $a \oplus b \oplus c$ is ambiguous–there is no common convention on whether this should mean $(a \oplus b) \oplus c$ or $a \oplus (b \oplus c)$. Therefore, one should always use brackets. Additionally, you might be surprised by the precedence levels of $0.5$ and $-1$. While they should be kept natural numbers, this is common when one retro-fits precedence levels.

### 2.1.5 Laws

In the last section, we developed equivalence $\equiv$.

A very important property of equivalence is that it is substitutive.

**Lemma 2.21** (Substitution with Equivalence)**.** *Given two equivalent formulas $\varphi \equiv \psi$. If, in some third formula $\chi$, we replace an occurrence of $\varphi$ with $\psi$ to get a new formula $\chi'$, then $\chi \equiv \chi'$.*

In general, $\equiv$ can be used similar to how equality $=$ can be used. We discuss the principles allowing us to do this in Section 3.2.2.

For example, it is easy to see that $a \wedge b \equiv b \wedge a$. This then allows us to also conclude that $a \wedge b \rightarrow c \equiv b \wedge a \rightarrow c$, since we can replace the $a \wedge b$ by $b \wedge a$ while preserving equivalence. This kind of replacement is called **rewriting** – we say that we rewrite with the equivalence $a \wedge b \equiv b \wedge a$. Note that rewriting works both ways: we could also replace $b \wedge a$ by $a \wedge b$.

Together with Lemma 2.18, this gives us a way for proving that a formula is true, which does not involve truth tables. If we want to prove for example $a \wedge b \wedge c \rightarrow c \wedge b \wedge a$, we can show that this is equivalent to $\top$. To do so, we can repeatedly rewrite using certain equivalences we already know to be true, like $\varphi \wedge \psi \equiv \psi \wedge \varphi$ or $\varphi \rightarrow \varphi \equiv \top$, to eventually arrive at $\top$.

For this, we need a rather large library of "well-known equivalences." Such properties are also called **laws**. Since our laws are systems of equations used for rewriting, we also call them **algebraic**. These laws can later be used in proofs. we can also look at them to better understand how our operators from propositional logic actually work.

**Going Beyond:** Algebra

The word "algebra" has many meanings in mathematics, usually related to equational reasoning.
One of them is that anything that has some notion of equality (like $\equiv$), a collection of operators (like $\wedge, \rightarrow, \ldots$), and a collection of fundamental equalities (like $\varphi \wedge \psi \equiv \psi \wedge \varphi$) is called **an algebraic structure**.

Our running example $\varphi \wedge \psi \equiv \psi \wedge \varphi$ expresses a property called *commutativity*. Intuitively, an operator is commutative if we can swap the left and right arguments. The same property also holds for disjunction, but not for implication. Here is a collection of some of the most useful laws of propositional logic:

**Lemma 2.22** (Algebraic Laws of Propositional Logic)**.**

**Commutativity**
$\varphi \wedge \psi \equiv \psi \wedge \varphi$
$\varphi \vee \psi \equiv \psi \vee \varphi$

**Associativity**
$(\varphi \wedge \psi) \wedge \chi \equiv \varphi \wedge (\psi \wedge \chi)$
$(\varphi \vee \psi) \vee \chi \equiv \varphi \vee (\psi \vee \chi)$

**Distributivity**
$(\varphi \wedge \psi) \vee \chi \equiv (\varphi \vee \chi) \wedge (\psi \vee \chi)$
$(\varphi \vee \psi) \wedge \chi \equiv (\varphi \wedge \chi) \vee (\psi \wedge \chi)$

**Absorption**
$\varphi \wedge (\varphi \vee \psi) \equiv \varphi$
$\varphi \vee (\varphi \wedge \psi) \equiv \varphi$

**Idempotence**
$\varphi \wedge \varphi \equiv \varphi$
$\varphi \vee \varphi \equiv \varphi$

**Complement**
$\varphi \vee \neg\varphi \equiv \top$
$\varphi \wedge \neg\varphi \equiv \bot$

**Identity**
$\varphi \wedge \top \equiv \varphi$
$\varphi \vee \bot \equiv \varphi$
$\top \rightarrow \varphi \equiv \varphi$

**Definability**
$\varphi \rightarrow \psi \equiv \psi \vee \neg\varphi$
$\neg\varphi \equiv \varphi \rightarrow \bot$

**Double negation**
$\neg\neg\varphi \equiv \varphi$

**Contraposition**
$\varphi \rightarrow \psi \equiv \neg\psi \rightarrow \neg\varphi$

**Domination**
$\varphi \wedge \bot \equiv \bot$
$\varphi \vee \top \equiv \top$
$\varphi \rightarrow \top \equiv \top$
$\bot \rightarrow \varphi \equiv \top$

**De Morgan's laws**
$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$
$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$
$\neg\top \equiv \bot$
$\neg\bot \equiv \top$

*Proof.* Every law can be shown correct by doing a truth table and noting that both sides are equivalent. Here is one such table, for the first absorption law:

| $\varphi$ | $\psi$ | $\varphi \vee \psi$ | $\varphi \wedge (\varphi \vee \psi)$ |
|---|---|---|---|
| `true` | `true` | `true` | `true` |
| `true` | `false` | `true` | `true` |
| `false` | `true` | `true` | `false` |
| `false` | `false` | `false` | `false` |

As we can see, the rows for $\varphi$ and for $\varphi \wedge (\varphi \vee \psi)$ have the same truth values, hence both formulas are semantically equivalent. ◻

When proving these laws, we can also be more clever. Once we have proven some laws using truth tables, we can use those already-proven laws to prove other laws. For example, assume we have already proven the Definability, Commutativity, and Double Negation laws. Then, a proof of Contraposition might look like this:

$$\begin{aligned}
\varphi \rightarrow \psi &\equiv \neg\varphi \vee \psi && \text{Definability} \\
&\equiv \psi \vee \neg\varphi && \text{Commutativity} \\
&\equiv \neg\neg\psi \vee \neg\varphi && \text{Double Negation} \\
&\equiv \neg\psi \rightarrow \neg\varphi && \text{Definability}
\end{aligned}$$

We can also prove Domination without a proof table, if we already have otherwise proven the Complement, Associativity, and Idempotence laws.

$$
\begin{aligned}
\varphi \wedge \bot &\equiv \varphi \wedge (\varphi \wedge \neg\varphi) && \text{Complement} \\
&\equiv \varphi \wedge \varphi \wedge \neg\varphi && \text{Associativity} \\
&\equiv \varphi \wedge \neg\varphi && \text{Idempotence} \\
&\equiv \bot && \text{Complement}
\end{aligned}
$$

While this makes some of these proofs easier, one also has to be careful: When using some of these laws to prove a new law, the laws that are used in the proof must already be proven otherwise. It would be wrong to use the Complement law to prove Double Negation, and then use Double Negation to prove the Complement law. When using this approach, one needs to carefully track which laws have already been proven, so that no circular proof is created.

Thus, a sane approach is to start proving some laws using truth tables, and to eventually, once one has proven enough lemmas using proof tables, switch to using these laws algebraically.

Our laws here have a very special property. They together are strong enough that every valid equivalence in propositional logic can be proven using just those laws. In fact, as we've seen above, we do not even need all of these laws. We just need some of the categories.

**Theorem 2.23** (Completeness)**.** *The following laws can prove all equivalences in propositional logic.*

- *Commutativity*
- *Associativity*
- *Distributivity*
- *Absorption*
- *Idempotence*
- *Complement*
- *Identity (for $\wedge, \vee$)*
- *Definability of $\rightarrow$*

Try thinking about how you would prove the other laws from the laws of Theorem 2.23. For some laws, this is rather straightforward. For other laws, these proofs were very hard to discover, because they are very long and not at all obvious. Nonetheless, they exist.

The general property of being able to prove all true statements of some kind using just a few laws is called **completeness**. Here, it means that all of propositional logic is described by just those few laws. It tells us that there is "nothing more" to propositional logic than the facts described by these laws.

**Important Individual:** George Boole

George Boole (1815-1864) was an English mathematician and logician. His work "The Laws of Thought" established the algebraic laws of predicate logic, which was named Boolean algebra in honor of his name. The truth values $\mathcal{B}$ are often called *booleans*, also in honor of his name. He also came up with notions for quantifiers, and for multi-variate relations.
In 1864, George Boole walked to his university to give a lecture while it was raining heavily. He gave his lecture in wet clothes and fell ill with fever. He succumbed to his illness a few days later.

Let's say we wanted to prove all the other laws using just those few laws. In order to attempt this, it is useful to first prove some intermediate facts, which make proving other laws simpler later on. One of those lemmas is the following, which characterizes negation by giving two properties that, when fulfilled, say that a formula behaves exactly like the negation of some other formula.

**Lemma 2.24.** *Given $\varphi, \psi$ such that $\varphi \land \psi \equiv \bot$ and $\varphi \lor \psi \equiv \top$, then $\psi \equiv \neg\varphi$.*

*Proof.* We prove that $\varphi \equiv \neg\psi$, while using the two assumptions at certain rewrite steps.

$$\begin{aligned}
\psi &\equiv \psi \lor \bot && \text{Identity}\\
&\equiv \psi \lor (\varphi \land \neg\varphi) && \text{Complement}\\
&\equiv (\psi \lor \varphi) \land (\psi \lor \neg\varphi) && \text{Distributivity}\\
&\equiv (\varphi \lor \psi) \land (\psi \lor \neg\varphi) && \text{Commutativity}\\
&\equiv \top \land (\psi \lor \neg\varphi) && \text{Assumption}\\
&\equiv (\psi \lor \neg\varphi) \land \top && \text{Commutativity}\\
&\equiv (\neg\varphi \lor \psi) \land \top && \text{Commutativity}\\
&\equiv (\neg\varphi \lor \psi) \land (\varphi \lor \neg\varphi) && \text{Complement}\\
&\equiv (\neg\varphi \lor \psi) \land (\neg\varphi \lor \varphi) && \text{Commutativity}\\
&\equiv \neg\varphi \lor (\psi \land \varphi) && \text{Distributivity}\\
&\equiv \neg\varphi \lor (\varphi \land \psi) && \text{Commutativity}\\
&\equiv \neg\varphi \lor \bot && \text{Assumption}\\
&\equiv \neg\varphi && \text{Identity}
\end{aligned}$$

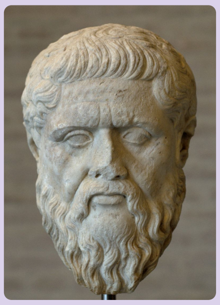
□

This theorem tells us that negation is completely defined by these two properties. Any formula that also satisfies these properties, which are based on the *identity* laws, works exactly the same way negation does. This not only tells us a lot about negation, but it also can help us when we want to prove that two formulas are equivalent, since we can use the lemma to show that one formula is equivalent to another's negation by just proving that the first formula fulfills the *identity* laws.

The *complement* laws also warrant a closer look. The first law states that $\varphi \lor \neg\varphi$ is valid. This is also known as the **Law of Excluded Middle**, which states that everything is either true or false.

This law has quite a few names: it is also known as **tertium non datur** (Latin for "a third [truth value] is not given"). The second law states that $\varphi \land \neg\varphi$ never holds. This is known as the **Law of Non-Contradiction** and simply says that nothing is ever both true and false.

**Important Individual:** Plato



Plato was an ancient Greek philosopher, active during the 5th and 4th centuries BC. He, along with his teacher Socrates, are often considered the founders of Western philosophy, which they impact until this day. Plato first explicitly used the law of non-contradiction and the law of excluded middle in his works. He also founded the first university, and heavily impacted later philosophers like Aristotle or Euclid. Today he is mostly known for his cave allegory, as well as for being namesake to a number of concepts like the Platonic solids or platonic relationships.

**Checkpoint 2.25:** Proofs Using Algebraic Rules

Prove the following laws using just the laws from Theorem 2.23.

- Domination: $\varphi \lor \top \equiv \top$
- De Morgan: $\neg(\varphi \lor \psi) \equiv \neg\varphi \land \neg\psi$
- Double Negation: $\neg\neg\varphi \equiv \varphi$

*Hint: Use Lemma 2.24 and results from the prior parts of the exercise!*

Finally, here is another very interesting property of propositional logic.

**Definition 2.26.** *For a formula $\varphi$ of propositional logic without implication, we can create its dual formula dual($\varphi$) by replacing*

- $\top$ *with* $\bot$
- $\bot$ *with* $\top$
- $\land$ *with* $\lor$
- $\lor$ *with* $\land$.

**Lemma 2.27** (Duality). *Let $\varphi, \psi$ be formulas in propositional logic (without implication) such that $\varphi \equiv \psi$. Then dual($\varphi$) $\equiv$ dual($\psi$).*

*Proof.* By Theorem 2.23, we can prove the original equivalence $\varphi \equiv \psi$ by rewriting with certain laws of propositional logic. Now, we prove dual($\varphi$) $\equiv$ dual($\psi$) by taking the exact same rewrite steps, except that we apply the dual law every time. For that, it simply needs to hold that the dual of every law used during rewriting is also a law. But this is obvious by simply looking at the laws used in Theorem 2.23 and seeing that the two laws in each category are their respective dual statements. □

The duality lemma allows us to immediately see that if we have for example $(a \land b \land a) \lor \neg a \lor b \equiv \neg a \lor b$, then also $(a \lor b \lor a) \land \neg a \land b \equiv \neg a \land b$.

**Checkpoint 2.28:** Pitfalls with Duality

What is the problem with the following proof?

*Proof that* $\top \equiv \bot$.
Let's say that we manage to prove that $\top \equiv \varphi$ for some $\varphi$. Then by duality, also $\bot \equiv \varphi$. We would then have $\top \equiv \varphi \equiv \bot$.
Since such a $\varphi$ is very easy to find, we have $\top \equiv \bot$. □

## 2.2 First-Order Logic

So far, we have seen that we can use propositional logic to formally describe basic statements. We have also seen a way of figuring out whether a statement is valid, or true, by using a proof table.

Sadly, propositional logic is not strong enough to even attempt doing regular mathematics with it. To see why, let us consider a usual mathematical statement:

*For all rational numbers $a, b$, if $a < b$, then there is a number $c$ such that $a < c$ and $c < b$.*

If we tried to formalize this in propositional logic, we might try to come up with the following statement, where the parts in quotations represent atomic propositions:

$$\text{“}a < b\text{”} \to (\text{“}a < c\text{”} \land \text{“}c < b\text{”})$$

This, however, is unsatisfactory: The original statement we wanted to formalize said a lot more than our attempt above. Our new formula is, in fact, nonsensical since we can not understand it without knowing what $a, b$ and $c$ are. Originally, we had that $a$ and $b$ could be arbitrary numbers, and that $c$ was a special number which somehow depended on $a$ and $b$.

To solve this issue, we will strengthen our logic by allowing it to talk about objects. Instead of atomic propositions, we now have predicates. A **predicate** is a statement about objects, like "$x$ loves $y$" for variable $x, y$. We add predicates and object variables to our logic, so that we can express such predicates that relate several objects, while allowing us to study what happens when we change the specific objects being related. We further add logical connectives $\forall$ and $\exists$, which allows us to express that something is true for all objects, or for at least one, respectively. This fundamentally changes our logic, so we give it a new name: first-order logic. The above statement can then be represented by the following first-order formula:

$$\forall a, b : a < b \to \exists c : a < c \land c < b$$

**Chapter Goals**

In this chapter, we develop first-order logic, a logic which can talk about objects. This includes

- The syntax of first-order logic
- A notion of truth for first-order logic
- Discussing scoping, binding, and variables
- Discussing how first-order logic is used to do useful mathematics

### 2.2.1 Syntax

To define first-order logic formally, we extend propositional logic with variables and means to introduce them. This means that variables for objects now become part of our syntax, and different kinds of variables will be required in different places. This is not uncommon in mathematics. The usual way to discriminate between different kinds of variables is by choosing them from different alphabets. We follow this convention and use the following schema:

- **Predicate symbols**: $P, Q, R, \ldots$ Predicate symbols, also called predicate variables, are uppercase characters. These variables work similarly to the atomic statements we know from propositional logic, except they can now refer to objects. For example, in the statement "For all $x$, $P(x)$," $P$ is a predicate variable.
- **Object variables**: $x, y, z, \ldots$ The variables are lowercase characters and are used as part of the syntax of first-order logic. For example, in the statement "For all $x$, $P(x)$," $x$ is an object variable.
- Meta-variables: $\varphi, \psi, \ldots$ These variables are lowercase Greek letters. We have seen these before.

The syntax of **first-order formulas** can now be specified by a BNF:

**Definition 2.29 (Syntax of First-Order Formulas).** *In the following, $x$ is a stand-in for any object variable, and $P$ is for any predicate symbol.*

$$\begin{aligned}
\mathcal{F}_1 \ni \varphi, \psi ::= {} & \neg\varphi \\
& \mid \varphi \land \psi \\
& \mid \varphi \lor \phi \\
& \mid \varphi \rightarrow \psi \\
& \mid \forall x : \varphi \\
& \mid \exists x : \varphi \\
& \mid P(x_1, \ldots, x_n) \qquad\qquad n \in \mathbb{N}
\end{aligned}$$

Most of the syntax is already known from propositional logic. There are two new logical connectives, namely $\forall$ and $\exists$, as well as the case for predicates, which we will look at first.

**Predicates** relate several objects. There are many examples for predicates:

- $\mathit{prime}(x)$, the predicate describing that something is a prime number.
- $x = y$, the predicate describing that two objects are the same. Note that $x = y$ is just syntactic sugar for $=(x, y)$.
- $x > y$, the predicate describing that a number is greater than some other number. Similarly, this is just syntactic sugar for $<(y, x)$.
- $\mathit{married}(x, y)$, the predicate describing that $x$ and $y$ are married. If we wanted syntactic sugar, we might write this as $x$ ⚭ $y$.

The way formula constructed from a predicate is read out loud is usually dependent on the predicate. For example, $\mathit{prime}(x)$ is usually read as "$x$ is prime" or "$x$ is a prime number." $x = y$ is read as "$x$ is equal to $y$" or simply "$x$ equals $y$," as expected.

Note that every predicate only talks about a fixed number of objects. *prime* only describes a single number, while = relates two numbers.

We call the number of objects the predicate talks about its **arity**. Hence, *prime* is a predicate of arity 1, and = is a predicate of arity 2, and so on. The rule for predicates – $P(x_1, \ldots, x_n)$ – allows us to apply any predicate to any number of variables. Since our formulas only make sense if the number of variables there is exactly the arity of that predicate, we require that the rule is only used like this. It is also possible to have a predicate which takes no variables at all, by setting its arity as 0. This is important because it allows us to express $\top$ and $\bot$ in our logic: by having $\top()$ and $\bot()$ as predicates of arity 0. While these two examples are the most obvious, it can be useful to have other atomic predicates of arity 0, since this allows us to translate every formula of propositional logic into first-order logic. When building formulas, we also have to specify the concrete predicates we plan to use. This is called the **signature** of our formulas.

Next, we have $\forall$ and $\exists$, which are called **quantifiers**. We call $\forall$ the **forall-quantifier** or **universal quantifier**, while $\exists$ is the **existential quantifier**. These quantifiers allow us to express statements holding for all or for some objects.

While the symbols for quantifiers may seem strange, they become familiar once we understand how they are to be read:

- A formula like $\forall x : \varphi$ is read as “for all $x$, $\varphi$ holds,” or as “for every $x$, $\varphi$,” for example. Thus, a formula like $\forall x : x = x$, is read as “for all $x$, $x$ is equal to $x$.” If we wanted to be creative, we could also say “All $x$ are equal to themselves.”
- A formula like $\exists x : \varphi$ is read as “there is $x$ such that $\varphi$,” or as “there exists an $x$ for which $\varphi$ holds,” for example. Hence, a formula like $\exists x : \mathit{prime}(x)$ can be read as “there is an $x$ such that $x$ is prime,” or as “there is a prime number” if one wants to be creative.

We have not yet defined a formal semantics for first-order logic, but based on the above intuition, you can deduce what is meant by the various connectives.

**Example 2.30:** First-Order Formulas

All of these are first-order formulas:

- $\top \to \bot$
- $\forall x : (P(x) \to (\exists y : R(x, y)))$
- $\forall x : \top$
- $\forall x : (\forall y : P(x))$
- $P()$
- $\forall x : (\forall y : (P(x, y) \to P(y, x)))$
- $\forall x : (\exists x : (\forall x : P(x, x, x)))$
- $\exists y : \bot$

First-order logic is usually abbreviated as **FOL**. Sometimes, it is also called **predicate logic**. The symbol for *first*-order formulas $\mathcal{F}_1$ thus has a 1 as subscript. Propositional logic had the symbol $\mathcal{F}_0$, and is sometimes called *zeroth*-order logic, like when comparing it to first-order logic.

### Precedence Rules for First-Order Logic

The operator precedence for first-order logic follows that of propositional logic. We have the following precedence rules, from strongest to weakest:

1. $P_n(x_1, \ldots, x_n)$: predicates bind strongest. This is unsurprising since they do not have subformulas.

**Going Beyond:** Higher-Order Logics

As the name "first-order" logic suggests, there also are **higher-order logics**. The difference is that in first-order logic, quantifiers only quantify over objects. In second-order logic, quantifiers also quantify over predicates. There, we can have statements like $\forall P : P \vee \neg P$. This would then mean that all propositions $P$ are either true or false, i.e. the law of excluded middle. Such logics also allow quantifying over all predicates, so that the following would be valid: $\exists P : P(0) \wedge \forall n : P(n) \leftrightarrow \neg P(n+1)$. The formula is valid since the searched predicate $P(x)$ is precisely $\mathit{even}(x)$. Third-order logic would then allow quantifying over all properties of such predicates, and so on. In the end, we reach infinite-order logic, where we can do all of these quantifications all at once.
In this chapter, we focus on first-order logic. In later chapters, we will see some higher-order constructs. By then, this will feel very natural.

**Going Beyond:** History of First-Order Logic

You might have noticed that there is no important individual responsible for the development of first-order logic. This is because first-order logic emerged gradually, and many people made significant contributions to it, like Boole, Frege, Russel, Whitehead, Löwenheim, Hilbert, Gödel, and many others. Modern formal logic got started in the 1800s, and it took until the 1930s for first-order logic to be recognized as a distinct concept. A detailed account of its development can be found at this website:

https://plato.stanford.edu/entries/logic-firstorder-emergence

**Checkpoint 2.31:** First-Order Formulas

- Consider the formulas of Example 2.30:
  - Make sure you can read these expressions.
  - Remove all redundant brackets from these expressions.
- Translate the following sentences into first-order logic. To express that two people are married, use $\mathit{married}(x, y)$; similarly $\mathit{single}(x)$ denotes that a person is single.
  - Everyone is either single or married.
  - If a person is married, their partner is not single.
  - There is a person who is married to two different persons. *Hint: use* =.

2. $\neg\varphi$: Like before, negation is the next-strongest connective. Multiple negation operators do not require brackets (i.e. $\neg\neg\varphi$ is valid syntax).
3. $\varphi \wedge \psi$: left-associative, following the rules of propositional logic.
4. $\varphi \vee \psi$: left-associative, following the rules of propositional logic
5. $\varphi \rightarrow \psi$: right-associative, still following the rules of propositional logic.

6. $\forall x : \varphi$ and $\exists x : \varphi$. These bind the weakest. This means that quantifiers always try to bind the largest possible subformula. Like with negation, when we put multiple quantifiers behind each other, we do not need brackets (i.e. $\forall x : \exists y : \varphi$ is valid).

Thus, the following formulas are the same:

- $\forall x : \forall y : x = y \to y = x$ and $\forall x : (\forall y : ((x = y) \to (y = x)))$
- $\forall x : (\exists y : P(y)) \to P(x) \land Q(x)$ and $\forall x : ((\exists y : (P(y))) \to ((P(x)) \land (Q(x))))$

**Variables and Scopes**

The reason we introduced quantifiers was to allow us to make statements about objects. The quantifiers then tell us which objects these statements refer to (namely “all” or “some”). Yet, our syntax still allows us to construct formulas like $P(x, y)$, where we do not know what $x$ or $y$ are. Since we want our formulas to be statements, which are either true or false, we are now going to exclude such formulas, where the variables do not “belong” to any quantifier. For this, we will formally define what it means to “belong” to a quantifier.

Every quantifier introduces a variable. This variable can then be used in the subformula contained in the quantifier. We call this formula in which that variable can be used the **scope** of that variable. For example, the scope of $x$ in the formula $\forall x : x = x$ is the subformula $x = x$.

Note that in our example, the variable $x$ occurs three times. The first $x$ is between the quantifier $\forall$ and the colon (:). The other ones are within the subformula $x = x$ of the quantifier. We notice that the first $x$ is special and different from all the other uses of $x$ because it *defines* where $x$ can then be used. This first occurrence of $x$, between a quantifier and the corresponding colon, is called the **binding place** of $x$. It **opens a scope**, in which $x$ can then be used. The other uses of $x$, which are in scope, are called **bound occurrences**. All other uses that are neither bindings nor bound occurrences are called **free uses**. All the variables which have free uses in a formula are called the free variables of that formula. We may shorten this to just saying that some variables are **free in some formula** $\varphi$. Somewhat confusingly, we also say that a formula $\varphi$ is **closed under a collection of variables** $A$ if $A$ contains all free variables of $\varphi$.

In order for a formula to be considered a statement, it must not have free variables, otherwise, we would not be able to determine whether such a formula is true or false. In fact, every (non-binding) usage of a variable must be bound at a unique binding place. We thus refrain from formulas with free variables, except when discussing the properties of formulas themselves.

While this rules out formulas like $P(x, y)$, for which we can not determine whether they are true or false, this new rule also seems to forbid another group of formulas. For example, consider $\forall x : \exists x : x = x$. Without doubt, there are no free uses of $x$ in this term: Both uses appear to be within both the scopes opened by the $\forall$-quantified $x$ and the $\exists$-quantified $x$. This seems to violate our condition that all uses of a variable must have a *unique* binding place: There are two suitable binding places which could bind $x$. To handle this case, we define a new rule: Variables are bound at the *nearest* quantifier. To be precise, when some variable $x$ makes a binding occurrence when another $x$ already is in scope, it opens a new scope. While the new scope is open, the old scope is still there, but **inactive**. It is also said that the identifier is **invisible**. Thus, there is only ever one active scope for each variable. When this happens, we say that the inner variable **shadows** the outer variable.

A connective like ∀ and ∃, which allows variables to have binding occurrence, is called a **binder**.

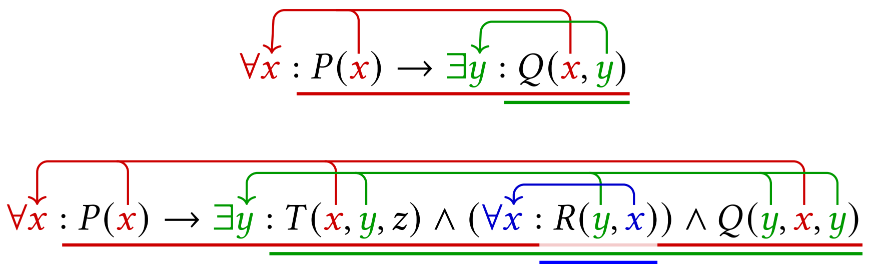
**Going Beyond:** Shadowing

Shadowing very often appears in programming languages. For example, consider the following snippet of C:

```
int x = 0;
int main() {
    int x = 1;
    printf("%d\n", x);
}
```

This code defines $x$ twice: once as a global variable with value 0, and once as a local variable with value 1. Then, it prints $x$. This program will print 1, since name analysis will deduce that the inner, local $x$ is the active definition of $x$, since the C standard specifies that local variables shadow global variables.

Since all of this was very dry, we discuss these terms by annotating the formulas in Figure 2.32.



$\forall x : P(x) \to \exists y : Q(x, y)$
$\forall x : P(x) \to \exists y : T(x, y, z) \land (\forall x : R(y, x)) \land Q(y, x, y)$

Figure 2.32: Two first-order formulas, annotated with scoping and binding information

The figure actually shows two formulas, a simpler one with two quantifiers, and a more complicated one with three quantifiers. In the first formula, we have two binders, namely $\forall x$ and $\exists y$. In the second one, there additionally is $\forall x$, which binds a variable also named $x$. Each of these opens a scope. The binders contain binding occurrences of the variables $x$ and $y$. The outline of each scope is displayed by underlining it in the corresponding color. Notice that in the second example, the scope of the outer $x$ is made inactive by that of the inner $x$. Once the inner scope is terminated, the outer scope resumes as the active scope. Each bound occurrence of a variable is connected by an arrow to the corresponding binder. In the second example, there also is a free variable: $z$. It is not bound by any quantifier.

This figure also demonstrates why it is called "shadowing:" The scope opened by the inner $\forall x$ makes the one belonging to $\forall x$ inactive, by casting a shadow over it.

We have also drawn the syntax tree of these formulas in Figure 2.33. Here, we have again drawn arrows connecting the bound occurrences with their binding places. Additionally, the scopes of each variable have been highlighted. We can then see that the scope extends over the entire subformula belonging to that quantifier.

You might wonder how this notion of scopes mixes with the shadowing going on in the second formula. Indeed, there it is not possible to refer to the $x$ bound to the outer quantifier within

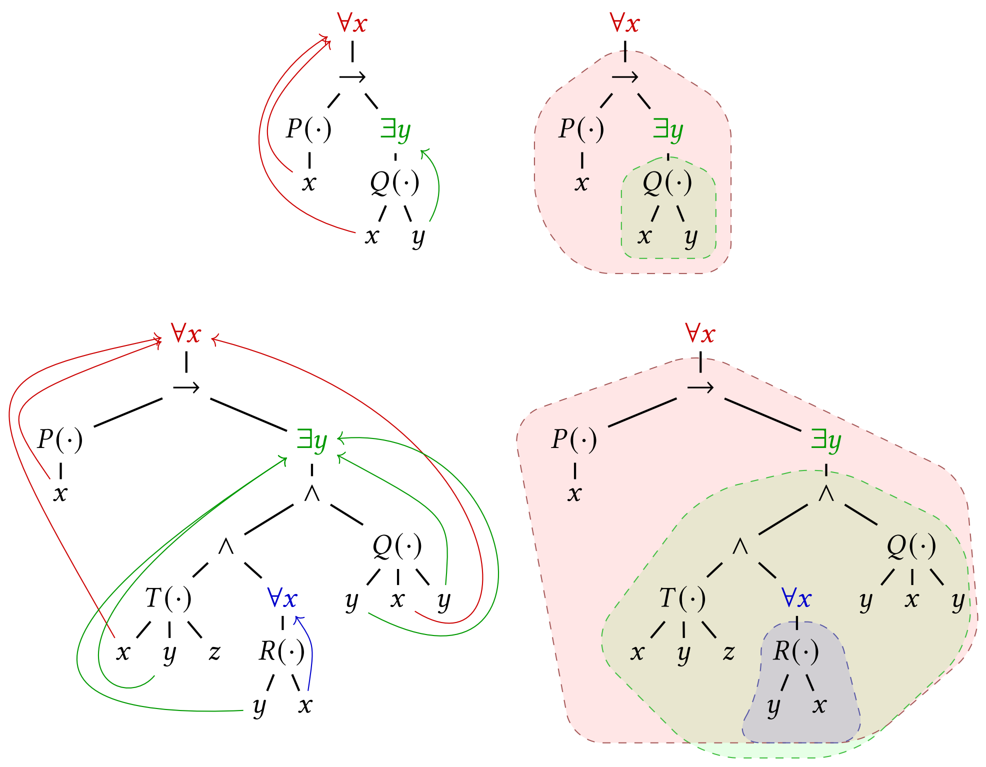


Figure 2.33: Visualizing active scopes, bindings, and shadowing on the syntax trees of formulas

the subformula bound below the inner quantifier. There, the scope is inactive. But the inactive scope is “still there.” If we were to consistently rename the variables in our formulas such that no shadowing occurs, the scope also becomes active over the entire subformula. As a general note, shadowing should be avoided when writing formulas, since this makes the formula harder to comprehend.

Also, once we have drawn the syntax tree, finding the correct binder for a variable becomes easy, by following the following procedure: Start at some variable $x$, then walk upwards until you find the *lowest* binder binding $x$ (this is the one encountered first when walking upwards). The process of resolving which names bind where is called **name analysis**.

We now know what bindings and scopes are and how we can resolve the bindings and free variables of a formula. We learned that only closed formulas are valid mathematical statements.

**Checkpoint 2.34:** Scoping

In Checkpoint 2.31, you were asked to construct several formulas.
For each of these, perform name analysis by annotating scope and binding information as in Figure 2.32. Which variables are free?

**Renaming** Now, consider the formulas $\forall x : x = x$ and $\forall y : y = y$. We can see that these formulas are the same, up to the naming of bound variables. If we consider the variable $x$ in $\forall x : x = x$, and rename the binding and all bound occurrences of that variable to $y$, then we get the other formula. We call such a renaming operation, where we rename the binding as well as all variables bound at that binding point, an $\alpha$-**renaming** (read: “alpha-renaming”), or “consistent renaming.” While we have not yet defined a semantics for formulas, we note that consistently renaming a formula should under no circumstances change the semantics of it, because the formula still ought to say the same. This is a fundamental principle in mathematics (and more importantly computer science): The names we give to things do not really matter, except that they make the statement easier to read for humans. For this reason, we say that two first-order formulas are *syntactically equal* iff they only differ in the naming of bound variables, i.e. if one can be consistently renamed to the other.

Consistent renaming seems easy at first. However, it is full of pitfalls and edge cases. For example, let’s take the formula $\forall x : \exists y : x = y$ and try to rename the $\forall x$ into a $\forall y$. If we are not careful, we end up with $\forall y : \exists y : y = y$. We can immediately see that these formulas have different internal binding structures. However, we clearly tried to do a consistent renaming, which should not have changed the internal binding structure, much less any other property of the formula. The problem is that during the renaming, we renamed a variable $x$ to $y$ “under a binder” already binding $y$. This process is called **capture**: By renaming $x$ to $y$, the binding point of the variable that used to be $x$ changes (“the variable is captured”), thereby altering the internal structure of our formula. There is no easy solution to this problem – instead, we must be very careful when renaming variables to not accidentally rename a variable $x$ to some variable $y$ which already is in scope. If we really need to do such a renaming, the solution is to first consistently rename the variable which is blocking the original renaming into something new, so that we do not incur capture. This is called **capture-avoiding renaming**, as opposed to the naive, capture-incurring renaming. For example, a proper consistent renaming of $x$ to $y$ in our example would first rename the inner $y$ into $z$, so that the end result is $\forall y : \exists z : y = z$.

### Terms

The definition of first-order formulas we just gave is actually incomplete. While it is already very powerful, it misses an important feature: Constructing new objects from given objects. This will be done by means of functions. Functions are stand-ins which describe objects constructed from other objects. For example, when working with natural numbers, + is a function that takes two natural numbers and describes a new one. In general, a function takes some objects as input and describes a new object as output. The most important property of functions in mathematics is that, given the same inputs, they always describe the same output. Importantly, we also have functions which do not take inputs. These are essentially constants since they always describe the same object.

We can already imagine how we would intuitively use functions in our logic. For example, the fact that addition is commutative can be expressed as $\forall x : \forall y : x + y = y + x$, or, if we refrain from syntactic sugar, as $\forall x : \forall y : =(+(x, y), +(y, x))$.

We will now formally extend our BNF to handle this:

**Definition 2.35 (Syntax of First-Order Terms).**

$$\begin{aligned}
\mathcal{F}_1 \ni \varphi, \psi &::= \cdots \\
&\mid P(\tau_1, \ldots, \tau_n) && n \in \mathbb{N} \\
\mathcal{T} \ni \tau &::= x \\
&\mid f(\tau_1, \ldots, \tau_n) && n \in \mathbb{N}
\end{aligned}$$

We introduced a new syntactic category $\mathcal{T}$, of **terms**. A term $\tau$ is either a variable or a term constructed by combining other terms with a function. Our notion of free and bound variables now applies to terms as it applies to formulas since one term can now contain multiple variables. Like predicates, they have an arity, which describes the number of terms we need to put into the function. To continue our example, + has arity 2. Constants (like 0 or 1) have arity 0.

The way a term is read aloud is again highly dependent on the concrete symbol, again mirroring predicates.

The function symbols are also part of the signature.

**Substitution** We have already seen capture-avoiding renaming. Capture-avoiding renaming is just a core building block of a more general operation called capture-avoiding substitution, which replaces a variable with a term.

In a formula, we denote by $\varphi[\tau/x]$ the **capture-avoiding substitution** of $x$ by $\tau$. That is, all *free* usages of $x$ are replaced by $\tau$. Note that this *only* includes free usages: If $x$ is bound in some subformula of $\varphi$, these usages are not replaced. Furthermore, $\tau$ can contain other variables, which could become captured by binders in $\varphi$. In this case, we again might need to consistently rename the conflicting binders so that no capture occurs.

We can also give a formal definition of substitution:

**Definition 2.36** (Capture-Avoiding Substitution $\varphi[\tau/x]$ and $\tau'[\tau/x]$).

$$\begin{aligned}
y[\tau/x] &= \tau && \textit{if } y = x \\
y[\tau/x] &= y && \textit{if } y \neq x \\
(f_n(\tau_1, \ldots, \tau_n))[\tau/x] &= (f_n(\tau_1[\tau/x], \ldots, \tau_n[\tau/x])) \\
(P_n(\tau_1, \ldots, \tau_n))[\tau/x] &= (P_n(\tau_1[\tau/x], \ldots, \tau_n[\tau/x])) \\
(\neg\varphi)[\tau/x] &= \neg(\varphi[\tau/x]) \\
(\varphi \mathbin{\square} \psi)[\tau/x] &= (\varphi[\tau/x]) \mathbin{\square} (\psi[\tau/x]) \\
(\Diamond y : \varphi)[\tau/x] &= \Diamond y : (\varphi[\tau/x]) && \textit{if } x \neq y, y \textit{ not free in } \tau \\
(\Diamond y : \varphi)[\tau/x] &= \Diamond y : \varphi && \textit{if } x = y
\end{aligned}$$

*$\Diamond$ is a stand-in for any quantifier, and $\square$ is a stand-in for any binary operator.*

By ensuring that $y$ is not free in $\tau$ in the rule for substituting under a quantifier, we ensure that there is no capture. However, this means that, if a naive substitution would incur capture, no rule would be applicable. To proceed anyway, we must first consistently rename $y$ in the formula $\Diamond y : \varphi$, as discussed. See Example 2.37 for how this works in practice.

**Example 2.37:** Capture-Avoiding Substitution

- $(\forall x : x + y = y + x)[0/y] = (\forall x : x + 0 = 0 + x)$
- $(\forall x : x + y = y + x)[0/x] = (\forall x : x + y = y + x)$
- $(\forall x : x + y = y + x)[x/y] = (\forall z : z + x = x + z) = (\forall w : w + x = x + w)$
- $(\forall x : x + y = y + x)[y/y] = (\forall x : x + y = y + x)$
- $(\forall x : x + y = y + x)[(1 + y)/y] = (\forall x : x + (1 + y) = (1 + y) + x)$

### 2.2.2 Semantics

So far, we have only covered the syntax of first-order logic. We now know enough formalities to actually define what it means for a first-order formula to be "true." Unfortunately, we cannot give a simple definition, as we did for propositional logic. The issue is that our formula now talks about objects, which raises the question of what objects are meant precisely.

To answer this, we define the notion of a **universe $\mathcal{U}$**, which is a special kind collection of objects. The collection might be small, large or even infinite, but it may *not be empty*. What makes this special is that each universe also defines the atomic predicates and functions for all predicate and function symbols of a given signature. These are called the **interpretations** of these symbols. Here, we need to make precise the difference between a predicate symbol and its interpretation: A function/predicate symbol is used to build first-order formulas. It takes several terms as its argument. When we define its interpretation, we need to define something that does not take terms as arguments, but instead takes objects of the specific universe as arguments. This distinction between a function/predicate symbol and its interpretation is easily glossed over, especially when one uses the same syntax for both the symbol and its interpretation. It is nonetheless important, especially since the interpretation needs to be defined for all objects in the universe so that the universal quantifier makes sense, even if not all such objects can be described using terms.

Note that universes are often called **models**. Examples 2.38 and 2.39 show how such universes can be defined.

Now, given a universe, we can finally evaluate whether a formula holds in that universe. We formally do this by defining evaluation and satisfaction, which describe how terms and formulas are interpreted in our universe. To do so, we need an **environment** $\rho$ on that universe, which assigns each variable an object from the universe. **Evaluation** $\mathcal{T}[\![\cdot]\!]_\rho$ now takes a term and gives an object from the universe:

**Definition 2.40** (Evaluation)**.** *Let $\mathcal{U}$ be a universe.*

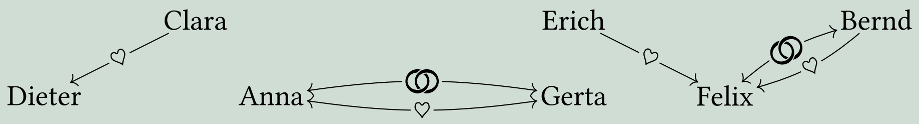
$$\mathcal{T}[\![x]\!]_\rho \coloneqq \rho(x)$$
$$\mathcal{T}[\![f_n(\tau_1, \ldots, \tau_n)]\!]_\rho \coloneqq \hat{f}_{\mathcal{U}}(\mathcal{T}[\![\tau_1]\!]_\rho, \ldots, \mathcal{T}[\![\tau_n]\!]_\rho)$$

So, to find out which object is meant by $x$, we simply look into the environment. To find out which object is described by a function, we take $\hat{f}_{\mathcal{U}}$, which is the interpretation of $f$ in our universe, and plug the objects described by the subterms into this.

Similarly, we have satisfaction, which defines whether a formula is "true" or **satisfied under a**

**Example 2.38:** A Social Clique as a Universe

We can form a universe from a group of friends, to describe how they relate to each other. Let's say we have a group of friends, namely Anna, Bernd, Clara, Dieter, Erich, Felix, and Gerta. We might define $x$ ⚭ $y$ as "$x$ is married to $y$," and $x$ ♡ $y$ as "$x$ is in love with $y$". $\mathcal{U}$ is fixed as the collection of the few friends above.
We could then draw the following relationship diagram:



Here, the arrows define which way the predicates hold. Note that Clara loves Dieter, but Dieter does not love her back, while Anna and Gerta both love and are married to each other. We might call the situation between Felix and Bernd a loveless marriage as Felix does not love Bernd, and say that Clara has a broken heart because Dieter does not love her back.

**Example 2.39:** The Universe of Natural Numbers

We can construct a universe by simply choosing $\mathcal{U} := \mathbb{N}$, i.e. the natural numbers become our universe. This then means that the objects our formulas talk about are natural numbers. In previous examples, we have used the function symbols $0, +$, and $\cdot$ when talking about natural numbers. When building a universe, we must also specify how the function and predicate symbols in our signature work for the objects of our universe. When choosing $\mathcal{U} := \mathbb{N}$, it makes sense to give these symbols the usual definition, i.e. the function symbol $0$ simply describes the natural number $0$, $+$ describes addition, and so on. Similarly, $=$ is supposed to describe when two numbers are equal, and in our universe, this is exactly the definition we will use (compare with Definition 2.42).

**certain environment** in some universe. When a formula $\varphi$ is satisfied under some environment $\rho$ defining (at least) the free variables of that formula, we denote this as $\rho \models \varphi$. Formally, it is defined like this:

**Definition 2.41** (Satisfaction). *Let $\mathcal{U}$ be a universe, $\rho$ an environment on that universe. The satisfaction relation $\rho \models \varphi$ is defined by structural recursion:*

$$\begin{aligned}
\rho \models \neg\varphi &\text{ iff not } \rho \models \varphi \\
\rho \models \varphi \wedge \psi &\text{ iff } \rho \models \varphi \text{ and also } \rho \models \psi \\
\rho \models \varphi \vee \psi &\text{ iff } \rho \models \varphi \text{ or else } \rho \models \psi \text{ (or both)} \\
\rho \models \varphi \to \psi &\text{ iff, when assuming } \rho \models \varphi, \text{ then } \rho \models \psi \\
\rho \models \forall y : \varphi &\text{ iff, for every object } u \text{ of } \mathcal{U}, \rho[y \mapsto u] \models \varphi \\
\rho \models \exists y : \varphi &\text{ iff, for at least one object } u \text{ of } \mathcal{U}, \rho[y \mapsto u] \models \varphi \\
\rho \models P_n(\tau_1, \ldots, \tau_n) &\text{ iff } \hat{P}_{\mathcal{U}}(\mathcal{T}[\![\tau_1]\!]_\rho, \ldots, \mathcal{T}[\![\tau_n]\!]_\rho) \text{ is true}
\end{aligned}$$

These definitions are similar to those of predicate logic. Again, we define the meaning of our connectives by referring to the meta-level, referencing the colloquial understanding of words like "and," "when" or "for every." By now, our colloquial understanding of "and," "or," "not", and "when" should be clear, since we discussed these in the previous chapter. This process of defining the truth value of some formula by checking whether it is satisfied in some universe, for some environment assigning free variables to values, is also called **interpretation**.

When the environment is empty, we can simply omit it, i.e. simply write $\models \varphi$. We say the environment is "initially empty," and we say that a formula is **satisfied** if it is satisfied under the empty environment. This reinforces that it does not make sense to talk about satisfaction for formulas with free variables: The environment might be undefined at places, and thus it is undefined whether a formula is satisfied or not. Further, even if the environment were defined, whether the formula ends up satisfied or not highly depends on which concrete objects end up in the environment. Since we want to avoid these ambiguities, we have forbidden such cases, so no issues actually arise.

Sadly, the rules for quantifiers make things much more difficult for us. Before, we could figure out whether a propositional formula was true by simply computing its truth value given the truth values for the atomic propositions. Now, we can no longer do so, as we would have to figure out whether a formula is true for every possible object in the universe. Since the universe can be infinite, this might take an infinite amount of time. Hence, when asked to determine whether a formula with quantifiers is true, we must reason about it using logical arguments. We will stick to an intuitive style of reasoning for now, where we simply rely on our intuitive understanding of mathematics.

There are a few predicates we would like to always behave the same in any universe:

**Definition 2.42 (Canonical Predicates).**

- $\top$: *This predicate (of arity 0) is always satisfied.*
- $\bot$: *This predicate (of arity 0) is never satisfied.*
- $=$: *The interpretation of $x = y$ (which has arity 2) for two objects $u_x, u_y$ of the universe is that $u_x = u_y$ iff $u_x$ and $u_y$ are the same object.*

*From now on, we consider all universes to define these interpretations.*

As long as our universes are small, we can check whether a formula is true by simply enumerating all the cases. Let us consider an example:

**Example 2.43:** Satisfaction in a Social Clique

We again consider our friend group from Example 2.38. The following statements are satisfied in this universe:

- Clara[3] $\heartsuit$ Dieter
- Bernd ⚭ Felix
- $\forall x : \forall y : x \text{ ⚭ } y \rightarrow y \text{ ⚭ } x$
- $\forall a : \forall b : a \heartsuit b \land b \heartsuit a \rightarrow a \text{ ⚭ } b$

Conversely, Dieter $\heartsuit$ Clara or Felix $\heartsuit$ Bernd are not satisfied.

We can now, finally, define a useful notion of truth for first-order formulas:

**Definition 2.44** (Validity)**.** *A first-order formula $\varphi$ is*

- ***valid*** *iff it is satisfied by all universes.*
- ***satisfiable*** *iff it is satisfied by at least one universe.*
- ***contradictory*** *iff there is no universe satisfying it.*

It turns out that validity is very close to a suitable notion of truth. For now, we call a formula **semantically true** iff it is valid. Until we introduce other notions of truth, we can leave out "semantic" and just say "truth."

**Example 2.45:** Validity

$\top$ is valid, as is $\forall x : x = x$. Since they are valid, they are also satisfiable. $\bot$ is contradictory. $\forall x : P(x)$ is satisfiable, but neither valid nor contradictory, since $P$ might be always satisfied in some universes, but not in others.

We are also able to define semantic equivalence of first-order formulas:

**Definition 2.46.** *Two first-order formulas $\varphi, \psi$ are **semantically equivalent** iff in all universes $\mathcal{U}$, and for all environments $\rho$, $\rho \models \varphi$ if and only if $\rho \models \psi$. We write $\varphi \equiv \psi$. Similarly, two terms $t_1, t_2$ are **semantically equivalent** iff in all universes $\mathcal{U}$, and for all environments $\rho$, the objects $\mathcal{T}[\![t_1]\!]_\rho$ and $\mathcal{T}[\![t_2]\!]_\rho$ are the same.*

**Checkpoint 2.47:** Satisfaction

For each of the following formulas, determine whether they are valid, satisfiable or contradictory. Give an explicit (counter-)example universe if possible.

- $(\forall x : P(x, x)) \to \forall x : \exists y : P(y, x)$
- $(\forall x : P(x)) \to (\exists x : P(x))$
- $(\exists x : P(x)) \to (\forall x : P(x))$
- $(\forall x : \exists y : P(x, y)) \to (\forall x, y : P(x, y) \to Q(y, x)) \to \exists y : Q(a, y) \lor Q(y, a)$ Note that $a$ is a constant.

### 2.2.3 Working with First-Order Logic

#### Syntactic Sugar

Similarly to propositional logic, working with plain first-order formulas is very tedious. Hence, we define similar syntactic sugar, to make working within first-order logic more attractive.

[3]Here, we abuse notation by referring to concrete objects of one specific universe. We could make this formal by declaring that the names (like Clara or Dieter) are constant symbols.

**Definition 2.48** (First-Order Syntactic Sugar)**.**

$$\begin{aligned}
\varphi \leftrightarrow \psi &:= (\varphi \to \psi) \land (\psi \to \varphi) \\
\varphi \oplus \psi &:= \varphi \land \neg\psi \lor \psi \land \neg\varphi \\
\forall x_1, \ldots, x_n : \varphi &:= \forall x_1 : \cdots \forall x_n : \varphi \\
\exists x_1, \ldots, x_n : \varphi &:= \exists x_1 : \cdots \exists x_n : \varphi \\
\forall \psi(x) : \varphi &:= \forall x : \psi(x) \to \varphi \\
\exists \psi(x) : \varphi &:= \exists x : \psi(x) \land \varphi
\end{aligned}$$

The first few definitions are straightforward, especially since we already know some of them from propositional logic. In general, all syntactic sugar defined for propositional logic can also be used in first-order formulas. The last two definitions might seem confusing. By $\psi(x)$ we denote a **predicate-form**, which basically is a formula where $x$ is free. $\psi(y)$ then denotes that same formula with $y$ for $x$. This basically allows one to use this syntactic sugar with arbitrary, "user-defined" predicates. The syntactic sugar is inspired by colloquial statements like "There is an $n > 5$ for which . . .," which can now be written as $\exists n > 5 : \varphi$.

Seeing why this definition of syntactic sugar is correct can be hard, so we will consider an example. Consider $\exists n > 5 : \mathit{even}(n)$, which is just $\exists n : n > 5 \land \mathit{even}(n)$. The first formula asks us to find a number $> 5$ such that it is even. When we think about this, this precisely means a number which is both $> 5$ and also even. It would not make sense to find a number which is $\leq 5$, or not even. Hence, both properties have to hold, and this is what our syntactic sugar expresses.

Similarly, we will consider $\forall n > 5 : \mathit{even}(n)$, which is just $\forall n : n > 5 \to \mathit{even}(n)$. Remember that the universal quantifier asks us to check whether its enclosed formula is satisfied for all possible objects, i.e. for all possible numbers in this case. Thus, we want the whole formula to be false if and only if we find a counterexample, which would be a number that is $> 5$, but not even. So, during the checking whether our quantifier is satisfied, we should ignore all numbers which are $\leq 5$. This is precisely what the implication $n > 5 \to \mathit{even}(n)$ accomplishes: This formula is satisfied as soon as $n \leq 5$, hence we never have counterexamples $\leq 5$, since they can not make our formula false. On the other hand, for numbers larger than 5, the implication is satisfied iff the right-hand side is satisfied, i.e. if the number is even. To summarize, this now means that when checking if the universal quantifier is satisfied, we can ignore all numbers $\leq 5$, but we must still look at whether the contained expression is satisfied for all numbers $> 5$, which is exactly what we wanted our syntactic sugar to do.

### How to Think about Quantifiers

What does it mean for a statement like $\forall x : \exists y : x \heartsuit y$ to be true? Is there a difference between that formula and the formula $\exists y : \forall x : x \heartsuit y$? The answer to the second question is "Yes," and this will be obvious very soon.

To understand this, we consider a few more friend groups like the ones of Example 2.38. Formally, these are the different universes/models we interpret our formulas in. We then look at what it means for formulas to be true in these universes, and how one reasons about this. For simplicity, we start with *finite* universes, which means that there are only finite numbers of friends. Later, we will look at infinite universes, like the natural numbers.

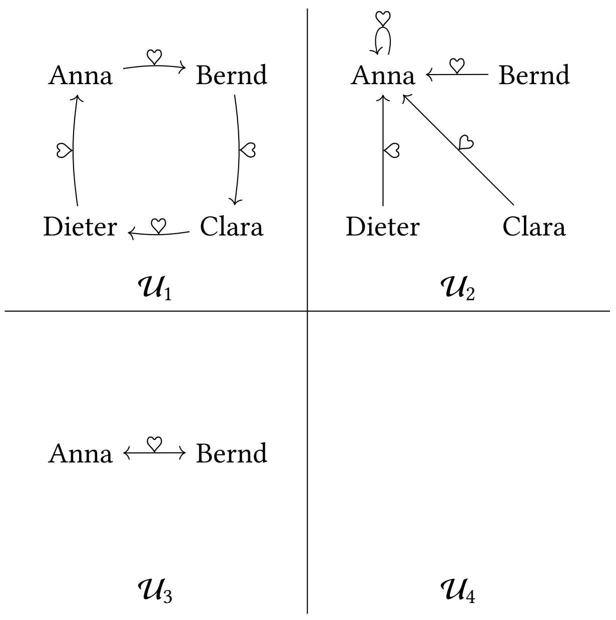


Figure 2.49: Four friend groups, displayed as universes.

| $\forall x$ | $\exists y$ | $x \heartsuit y$ |
|---|---|---|
| Anna | Bernd | ✓ |
| Bernd | Clara | ✓ |
| Clara | Dieter | ✓ |
| Dieter | Anna | ✓ |

Universe: $\mathcal{U}_1$

| $\exists y$ | $\forall x$ | $x \heartsuit y$ |
|---|---|---|
| Anna | Anna | ✓ |
| | Bernd | ✓ |
| | Clara | ✓ |
| | Dieter | ✓ |

Universe: $\mathcal{U}_2$

| $\forall x$ | $\forall y$ | $x = y \lor x \heartsuit y$ |
|---|---|---|
| Anna | Anna | ✓ (left) |
| | Bernd | ✓ (right) |
| Bernd | Anna | ✓ (right) |
| | Bernd | ✓ (left) |

Universe: $\mathcal{U}_3$

Figure 2.50: Tables showing why certain formulas are true in certain universes.

Consider the four universes $\mathcal{U}_1$ to $\mathcal{U}_4$ of Figure 2.49. In $\mathcal{U}_1$, the statement $\forall x : \exists y : x \heartsuit y$ is true. To see that it is true, consider the *left* table of Figure 2.50. This table shows the essence of how we argue that such a formula is true. Since the formula starts with a universal quantifier, we have to consider all possible choices for $x$, which is why we have four rows in our table (one for each object in our universe). In each row, we must then check whether the remaining formula $\exists y : x \heartsuit y$ is true. This formula now starts with an existential quantifier. To handle this in our table, we must give an element of our universe which makes the remaining formula true. For example, in the case where $x :=$ Anna, we choose $y :=$ Bernd, so that $x \heartsuit y$ becomes true (since Anna $\heartsuit$ Bernd). This $y$ that needs to be chosen here is called the **witness**. Since for each possible value of $x$, we can find a witness to make the corresponding formula true, the overall formula (with both quantifiers) is true as well. Note that we were able to pick a different witness in each row.

In universe $\mathcal{U}_2$, this formula is also true, since we can always pick Anna as a witness. Similarly, the statement $\exists y : \forall x : x \heartsuit y$ is true in $\mathcal{U}_2$, as shown by the right table of Figure 2.50. However, this formula is *not* true in universe $\mathcal{U}_1$. The reason for this is that with the quantifiers arranged this way, we must first choose a person $y$ such that everyone loves precisely this person. Since in $\mathcal{U}_1$, everyone loves a different person, this formula is false. This shows us that the order of quantifiers matters.

Of course, not every formula has exactly one existential and exactly one universal quantifier. As an example, we consider the formula $\forall x : \forall y : x = y \lor x \heartsuit y$. We consider this in universe $\mathcal{U}_3$, which only has 2 people (because otherwise, the table becomes too large). We can see that we have to consider all possible combinations of two people. By including $x = y$ in our formula, we ensure that it is true when we consider the same people. When $x$ and $y$ are different people (e.g., $x :=$ Anna and $y :=$ Bernd), it is then required that they are in love. In Universe $\mathcal{U}_3$, this is indeed the case.

## Quantifiers And Games

An alternative way to think about quantifiers is to think of them as a kind of game. This game is played by two players, for example by you (the reader) and a friend of yours. The game is played on a first-order formula, like $\forall x : \exists y : x \heartsuit y$, and on a certain universe, like $\mathcal{U}_1$. On this formula, the game starts with your friend selecting a person from the universe that plays the role of $x$. Then, you need to pick an object of the universe that is $y$. You win if the resulting formula $x \heartsuit y$ is true for these choices. For example, if your friend picks $x :=$ Bert, you need to pick $y :=$ Clara, otherwise you lose (since Bert only loves Clara). If you instead play this game on universe $\mathcal{U}_2$, you can always choose $y :=$ Anna, no matter what your friend picks for $x$, since everyone loves Anna.

When playing on an arbitrary formula, it is your friends' job to pick the objects for universal quantifiers, and your job to do so for existential quantifiers. If there are several quantifiers, you take turns choosing the objects. Once all quantifiers are handled, you need to determine the winner: You win if the remaining formula is true (with the corresponding choices). You lose if it is false. Thus, if you instead play on $\exists y : \forall x : x \heartsuit y$, you would start by choosing an object, and your friend can then react to this. So when playing on universe $\mathcal{U}_2$, you would start by choosing $y :=$ Anna. Then, no matter what your friend chooses, that person will always love Anna. When playing on $\mathcal{U}_1$, it turns out that your friend can always force a victory.

This game can be used to analyze whether a formula is true. If it is guaranteed that you can always win (assuming optimal play), then the formula is true. Otherwise, i.e. if it is possible for your friend to win (assuming you still play optimally), then the formula is false. To show that you can always win, you need to think of a *strategy* for countering your friend's moves. The tables of Figure 2.50 are precisely how such strategies look like. The game they are based on is called the **quantifier game**. In the intuitive explanation, you played as the prover, while your friend was the refuter.

**Definition 2.51** (Quantifier Game). *The quantifier game consists of two players: the prover, and the refuter. It is played in some universe $\mathcal{U}$. Initially, the prover asserts a formula $\varphi$ in* ***prenex normal form****, that is, all the quantifiers are at the beginning. The players take turns as follows:*

- *When $\varphi = \forall x : \varphi'$, it's the refuter's turn, and they must present an object $u_x$ to the prover.*
- *When $\varphi = \exists x : \varphi'$, it's the prover's turn, and they must present an object $u_x$ to the refuter.*

*In either case, the objects to be presented are elements of $\mathcal{U}$. Afterwards, the players continue with $\varphi'$, where $x$ is understood to refer to the object $u_x$.*

> *The game ends when the formula no longer starts with a quantifier. Then, the remaining formula (which now only contains operators of propositional logic, as well as relation symbols) is evaluated using the accumulated objects. If the formula is satisfied, the prover wins, otherwise, the refuter wins.*

**Lemma 2.52.** *Let $\varphi \in \mathcal{F}_1$ and $\mathcal{U}$ be a universe. If the prover always wins*[4] *the game on $\varphi$ (for universe $\mathcal{U}$), then $\varphi$ is true in that universe. Conversely, for a true formula, there always is a winning strategy for the prover.*

### Infinite Universes

The tables of Figure 2.50 do not quite work for infinite universes like $\mathcal{U}_4$ of Figure 2.49. If we were to try creating such a table for an infinite universe, this table would need to be infinitely large. To instead reason about infinite models, we use mathematical logic. In Chapter 3, proper mathematical proofs will be introduced, which allow reasoning about all kinds of infinite structures. For now, we will only take a baby step, towards showing that $\forall x : \exists y : x \heartsuit y$ is true in that infinite universe. In Figure 2.49, we have only drawn a small section of this universe. More formally, the persons in it are either called Anna or Bernd. In fact, we have infinitely many Annas, and infinitely many Bernds. To distinguish them, we give each a number, starting at 0 (since in computer science, we always start at 0). This means that there exists an Anna for each natural number, and similar a Bernd for each natural number. We now have that $\text{Anna}_0 \heartsuit \text{Bernd}_0$, but $\text{Bernd}_0 \heartsuit \text{Anna}_1$. Then, again, $\text{Anna}_1 \heartsuit \text{Bernd}_1$, along with $\text{Bernd}_1 \heartsuit \text{Anna}_2$. Formally, we have that $\text{Anna}_n \heartsuit \text{Bernd}_n$ and $\text{Bernd}_n \heartsuit \text{Anna}_{n+1}$. The following table now demonstrates why the formula is true.

| $\forall x$ | $\exists y$ | $x \heartsuit y$ |
|---|---|---|
| $\text{Anna}_i$ | $\text{Bernd}_i$ | ✓ |
| $\text{Bernd}_i$ | $\text{Anna}_{i+1}$ | ✓ |

Universe: $\mathcal{U}_4$

The idea is that we can analyze entire classes of objects at the same time. This table remains useful for giving a strategy as to how the quantifier game can be won: If your friend (playing as refuter) challenges you by choosing $x := \text{Bernd}_{42}$, you know to reply with $y := \text{Anna}_{43}$.

### Reordering Quantifiers

In the last section, we saw that $\forall x : \exists y : x \heartsuit y$ was true in universe $\mathcal{U}_1$, while $\exists y : \forall x : x \heartsuit y$ was not. Both formulas are the same, except we have swapped $\forall x$ and $\exists y$. Thus, we can not generally swap quantifiers around. This should be obvious since this would mess up the order of moves in the game outlined in the last chapter.

However, quantifiers can sometimes be swapped around. In general, if you have several quantifiers of the same kind (i.e. universal or existential), then they can be swapped around, as long as they are directly "next to" each other.

[4]Assuming that both players play optimally.

Let's again consider the universe $\mathcal{U}_3$. Consider the formula $\forall x : \forall y : x = y \lor y \heartsuit y$. In that formula, $\forall x$ and $\forall y$ are next to each other, since there are no other quantifiers between those. Thus, the formula where we swap $\forall x$ and $\forall y$ to end up with $\forall y : \forall x : x = y \lor y \heartsuit y$ is equivalent to the original formula, since $\forall x$ and $\forall y$ have the same kind (both being universal quantifier).

This can also be seen when playing the quantifier game on those formulas. If two quantifiers of the same kind are next to each other, then one player gets to choose two objects, one immediately after the other. What is really happening is that the player chooses a combination of two objects, and it does not matter which object is chosen first. It is different when another kind of quantifier is sandwiched in-between. Then, first one player makes a choice, then the other player can make a choice *depending* on the first player's choice. There, if we swap the order, the other player does no longer have the proper information.

To summarize, if we have a group of similar quantifiers (like $\forall x : \forall y : \forall z : \cdots$), we can re-order them. But we can not re-order across a quantifier alternation, like in $\forall x : \exists y : \forall z : \cdots$. There, no quantifier can be re-ordered. This motivates the syntactic sugar $\forall x, y, z : \cdots$, where we can combine several quantifiers of the same kind, since it does not matter which is actually first.

### Negating Quantifiers

Consider the statement "All people are married to someone." Formally, this reads $\forall x : \exists y : x ⚭ y$. What is the negation of this? An easy answer is $\neg\forall x : \exists y : x ⚭ y$. However, we can do better by thinking about what happens when not everyone is married to someone. If that is the case, there must be some person who is not married (to anyone). Formally, this reads $\exists x : \forall y : x ⚭ y$.

This is the essence of the rule for negating quantifiers: When negating a universal quantifier, it turns into an existential quantifier. Similarly, a negated existential quantifier turns into a universal quantifier. When doing so, the negation moves "under" the quantifier.

Formally, this is captured by the law $\neg\forall x : \varphi(x) \equiv \exists x : \neg\varphi(x)$, and the similar law for negated existential quantifiers.

This law is sometimes called **De Morgan's Law for Quantifiers**, since it mirrors De Morgan's laws for conjunction and disjunction. These similarly allow negating conjunction by turning it into disjunction, and moving the negation into both sides (and vice-versa).

### More on Quantifiers

With our quantifier, we can model the important properties that some property holds "for all" or "for at least one" object. This might make us wonder whether we can also express properties like "for at least two" or "for exactly one." In fact, we can define many such quantifiers just using the basic quantifiers of $\forall$ and $\exists$.

It turns out that "unique existence" $\exists! x : \varphi(x)$, which states that $\varphi$ holds for exactly one object, is another important property in mathematics. This property allows us to indirectly characterize objects by just describing a certain property that uniquely defines that object.

**Definition 2.53** (Uniqueness)**.** *The formula $\exists! x : \varphi(x)$ is syntactic sugar for the following:*

$$\exists x : \varphi(x) \land \forall y : \varphi(y) \to x = y$$

This captures our intuitive notion of "there is only one": Imagine that there were two objects $x, y$ both satisfying $\varphi(x)$. Then we can derive that $x = y$, so they are in fact the same object.

**Checkpoint 2.54:** Fancy Quantifiers

Dieter has tried to define the quantifier $\exists!2$, which is supposed to mean that "there are exactly two." What is wrong with the following definition?

$$\exists!2x : P(x) := \exists!x : \exists!y : P(x) \land P(y) \land x \neq y$$

Can you fix it?
Can you further define the following quantifiers? If not, why not?

- There are at least two
- There are at most three
- For all but four
- For finitely many
- For infinitely many
- For all but finitely many
- For none
- For exactly 50% of the universe

This is precisely what **uniqueness** means in maths: That any two objects satisfying a property are the same. Thus, the $\exists!$ quantifier is sometimes read as "there exists a unique." For an intuitive understanding, consider that every person has a unique fingerprint. Thus, if your fingerprint is found at a crime scene, this can be used to convict you of that crime, since your fingerprint uniquely identifies you. In that situation, you can not argue that another person could have left your fingerprint, since all people who could leave that fingerprint are *equal* to yourself.

Uniqueness also affects how we talk about objects. For example, we might say that 4 is *a* common divisor of 8 and 12. However, we say that 4 is *the* greatest common divisor of 8 and 12. This is because there can be multiple common divisors, but only one unique greatest common divisor. If we were to say that 4 is *the* common divisor of 8 and 12, this would be wrong, since we would imply that is the only such divisor, which it is not, as 2 and 1 also are divisors of both 8 and 12.

**Quantifiers and Finite Universes** The universe of Example 2.38 is a special kind of universe: It only has finitely many elements.

In general, first-order logic does not care about whether the model is finite, or infinite. In fact, later on, we mostly work in infinite universes.

Finite universes, however, have a very nice property: We can easily check whether a formula is true in that finite model by simply enumerating all the elements in order to check whether something holds for all, or for at least one.

In fact, if we are in an universe where there are terms $t_1, \ldots, t_n$ describing all the objects in that universe, we can even completely remove quantifiers:

- A quantifier $\forall x : \varphi(x)$ is transformed to $\varphi(t_1) \land \varphi(t_2) \land \cdots \land \varphi(t_n)$.
- A quantifier $\exists x : \varphi(x)$ is transformed to $\varphi(t_1) \lor \varphi(t_2) \lor \cdots \lor \varphi(t_n)$.

We can also use this to explain the $\forall\psi(x) : \varphi$ syntactic sugar from before. Let's say we want to express the property that "*Felix is married to all people loving him*" In first order logic, this reads as $\forall x \heartsuit \text{Felix} : \text{Felix} ⚭ x$.

Note that this is just syntactic sugar for $\forall x : x \heartsuit \text{Felix} \rightarrow \text{Felix} ⚭ x$. This is often confusing since the quantified formula is true for all persons not married to Felix. It becomes more clear when we consider the formula for every person in that universe.

| x | $x \heartsuit \text{Felix}$ | $\text{Felix} ⚭ x$ | $x \heartsuit \text{Felix} \rightarrow \text{Felix} ⚭ x$ |
|---|---|---|---|
| Anna | `false` | `false` | `true` |
| Bernd | `true` | `true` | `true` |
| Clara | `false` | `false` | `true` |
| Dieter | `false` | `false` | `true` |
| Erich | `true` | `false` | `false` |
| Felix | `false` | `false` | `true` |
| Gerta | `false` | `false` | `true` |

We can see that there is one person (Erich) for whom the quantified formula is false, so the overall formula with the universal quantifier is also false. We can see that the formula can only become false if there is a person for whom the precondition is true, but the postcondition is not. By using an implication, we are able to ignore all other elements of our model, for which the precondition is false. They no longer affect the truth of the overall quantifier since they are already true and do not need to be considered. Thus, the only elements left to check are those we actually care about, namely Erich, and Bernd.

### 2.2.4 Laws

In the last chapter, we have seen some laws of propositional logic. All of these laws are still laws of first-order logic.

However, we can now give more laws, which make use of quantifiers:

**Lemma 2.55** (Algebraic Laws of First-Order Logic)**.**

*$\varphi(x)$ denotes a predicate-form on $x$, i.e. a user-defined predicate on $x$.*

**Quantifier negation**

$\neg\forall x : \varphi \equiv \exists x : \neg\varphi$
$\neg\exists x : \varphi \equiv \forall x : \neg\varphi$

**Quantifier reordering**

$\forall x : \forall y : \varphi \equiv \forall y : \forall x : \varphi$
$\exists x : \exists y : \varphi \equiv \exists y : \exists x : \varphi$

**Pulling out universal quantifiers** ⚠

*We require that $x$ does not appear free in $\psi$.*
$(\forall x : \varphi) \land \psi \equiv \forall x : \varphi \land \psi$
$(\forall x : \varphi) \lor \psi \equiv \forall x : \varphi \lor \psi$
$\psi \rightarrow (\forall x : \varphi) \equiv \forall x : \psi \rightarrow \varphi$
$(\forall x : \varphi) \rightarrow \psi \equiv \exists x : \varphi \rightarrow \psi$

**Pulling out existential quantifiers** ⚠

*We require that $x$ does not appear free in $\psi$.*
$(\exists x : \varphi) \land \psi \equiv \exists x : \varphi \land \psi$
$(\exists x : \varphi) \lor \psi \equiv \exists x : \varphi \lor \psi$
$\psi \rightarrow (\exists x : \varphi) \equiv \exists x : \psi \rightarrow \varphi$
$(\exists x : \varphi) \rightarrow \psi \equiv \forall x : \varphi \rightarrow \psi$

**Quantifier splitting**

$(\forall x : \varphi) \land (\forall x : \psi) \equiv \forall x : \varphi \land \psi$
$(\exists x : \varphi) \lor (\exists x : \psi) \equiv \exists x : \varphi \lor \psi$

**Unused quantifier removal** ⚠

*We require that $x$ does not appear free in $\varphi$.*
$\varphi \equiv \forall x : \varphi$
$\varphi \equiv \exists x : \varphi$

**Environment weakening**

*If $\rho_1 \sqsubseteq \rho_2$ and $\rho_1 \models \varphi$ then $\rho_2 \models \varphi$*
*If $\rho_1 \sqsubseteq \rho_2$ then $\mathcal{T}[\![\tau]\!]_{\rho_1} = \mathcal{T}[\![\tau]\!]_{\rho_2}$*

**Laws of equality**

| | |
|---|---|
| $\forall x : x = x$ | *Reflexivity* |
| $\forall x, y : x = y \land \psi(x) \to \psi(y)$ | *Substitutivity* |
| $\forall x, y : x = y \to y = x$ | *Symmetry* |
| $\forall x, y, z : x = y \land y = z \to x = z$ | *Transitivity* |

*By $\rho_1 \sqsubseteq \rho_2$, we denote that $\rho_2$ has at least all the mappings $\rho_1$ has.*

*Proof sketch.* We delay the proof of most lemmas to the next chapter.

- Laws of equality: The reflexivity and substitutivity laws are better understood as axioms (see Definition 2.60). They formally capture the properties we intuitively mean when we say *the same*. Thus, they can not be proven, we can only make sure that they match up with our intuitive understanding of *being the same*.

  Symmetry and transitivity can be proven using the mentioned axioms. We also delay this to the next chapter.

- Environment weakening:
  Since $\mathcal{T}[\![\cdot]\!]_{\rho_1}$ and $\rho_1 \models \cdot$ only look up some objects in $\rho_1$, if we have a map $\rho_2$ with additional mappings, they are not used and do not affect the result.

  Formally, this is proven by structural induction (see Section 5.4) on the formula/term, with the environment quantified. □

**⚠ Warning**

Laws marked with ⚠ rely on the fact that the universe is not empty. They might no longer work when the syntactic sugar $\forall \varphi(x) : \psi(x)$ is used, since $\varphi(x)$ might be false for all objects. The other laws still work when using that syntactic sugar, except that quantifier splitting only works when the two quantifiers on the left have the same $\varphi$.

Here, we have proven the laws of equality by referring to our intuitive notion of "sameness." Alternatively, these laws can be seen as the definition of equality as a predicate with two properties: First, everything is equal to itself. Second, when two things are equal, all properties of one element hold for the other. That is, these elements are indistinguishable. The first two laws are powerful to derive the other laws, by cleverly choosing $\psi$.

The weakening laws allow us to use the fact that a formula is true in one environment, and substitute it into a different environment as long as the environment only becomes larger, without changing existing definitions.

### 2.2.5 Going Beyond: Theories

In this chapter, we have already seen how we can use first-order logic. Now, we will have a closer look at how this is done. For this, we will explore how we model mathematical definitions in first-order logic.

The first step is to pick a signature of symbols. For example, when modeling relationships as in Example 2.38, we add the predicate symbols $\heartsuit$ and ⚭ to our signature, besides the usual predicate symbols $=, \top, \bot$ we always require.

From these basic definitions, we can derive more complicated properties. For example, we can model when someone is heartbroken. This way, we can give a rigorous definition to these derived propositions.

**Definition 2.56** (Derived Propositions for Social Cliques)**.**

- $\mathit{heartbroken}\, a := \exists b : a \heartsuit b \land \neg(b \heartsuit a)$.
- $a \mathbin{⚭_{\not\heartsuit}} b := a \mathbin{⚭} b \land \neg(a \heartsuit b)$.

We can now write down first-order formulas describing the internal workings of social cliques. For example, consider $\forall a, b : a \mathbin{⚭} b \to b \mathbin{⚭} a$. This formula simply says that marriages are reciprocal, you can not be married to someone without them also being married to you. Formally, we say that marriage is symmetric. We agree that this law should hold for all actual social cliques, so it should be true. However, with our notion of truth so far – validity in all universes – the formula is not true. This is because we can easily construct "strange" universes where $a \mathbin{⚭} b$ holds, but the converse does not. However, we agree that these universes do not describe well-formed social groups. When we want to work with social groups expressed as universes of first-order logic, we agree that there are some basic properties these universes must fulfill, like the above property.

To exclude these unwanted universes, we lay down a set of requirements all universes we want to consider when trying to model social groups must fulfill. We formalize these requirements as first-order formulas. Each such first-order formula is called an **axiom**. A **theory** is a collection of such axioms for a given signature. Note that "theory" also refers to all true statements under that theory, not just the specific set of axioms.

Now that we have defined what universe "make sense" for a theory, we refine our notion of truth to limit itself to those universes.

**Definition 2.59.** *Given a theory $\mathcal{M}$ (which is a collection of axioms), a formula $\varphi$ is*

- ***valid modulo** $\mathcal{M}$ iff it is satisfied in all universes also satisfying all axioms in $\mathcal{M}$.*
- ***satisfiable modulo** $\mathcal{M}$ iff it is satisfied in at least one universe also satisfying all axioms in $\mathcal{M}$.*
- ***contradictory modulo** $\mathcal{M}$ iff it is not satisfiable modulo $\mathcal{M}$.*

When talking about formulas in the context of a specific theory $\mathcal{M}$, we consider them **semantically true** iff they are valid modulo $\mathcal{M}$.

### **Going Beyond:** On Formalization

This process of making reasoning more formal by first fixing and then sticking to a collection of axioms has several advantages. First, we improve our intuitive understanding of the structure in question (e.g. the natural numbers) by trying to find the basic properties of it. Second, when validating whether something is true by basing it on the axioms, we can be as certain of its validity as we are of the validity of the axioms. Lastly, this allows us to communicate our results to other mathematicians without having to completely spell out our intuitive notion of the structure in question—other mathematicians need only look at our axioms to see whether our results apply to their intuitive notion of the structure in question. This way, axioms define a common set of assumptions, without which collaborating on problems would be impossible.
This process of finding axioms is called **axiomatization** or **formalization**. Historically, it has been an extremely successful endeavor. To see why, try defining what a natural number is without referencing the word "number" or related words. You might see that this is almost impossible. By axiomatizing numbers, we are able to define the numbers as "members of some universe satisfying these axioms." That definition is rather clever since we do not actually describe what a number *is*. Instead, we defined numbers by describing how they behave and which properties they obey. Nonetheless, this definition enables anyone to work with natural numbers. This kind of definition is called *extensional* and is extremely common in mathematics.

### **Example 2.57:** The Theory of Social Groups

Social groups are described by two binary predicates, $\text{⚭}$ and $\heartsuit$, for which the following axioms must hold:

(a) $\forall a, b : a \text{ ⚭ } b \to b \text{ ⚭ } a$

(b) $\forall a, b, c : a \text{ ⚭ } b \to a \text{ ⚭ } c \to b = c$

(c) $\forall a : \neg(a \text{ ⚭ } a)$

(d) $\forall a, b : a \heartsuit b \land b \heartsuit a \to a \text{ ⚭ } b$

### **Checkpoint 2.58**

What does each of the axioms in Example 2.57 mean? Describe them in natural language!

We can now also give a more formal axiomatic description of the canonical predicates of Definition 2.42:

**Definition 2.60** (Axioms of the canonical predicates)**.**

- *The axiom for the predicate* $\top$ *is simply* $\top$, *that is,* $\top$ *is always valid.*
- *The axiom for* $\bot$ *similarly is* $\neg\bot$, *that is,* $\bot$ *is never valid.*

- *The axioms for equality are as follows:*
  - $\forall x : x = x$.
  - $\forall x, y : x = y \to \varphi(x) \to \varphi(y)$

  *We already know these axioms from our collection of laws (Lemma 2.55). The second axiom is actually an axiom scheme, which means that there are infinitely many axioms, one for each possible predicate form that first-order logic can express.*

To summarize this chapter, we now have a very formal way to make mathematical statements about something, for example social groups: First-order formulas over the signature of social groups. We also have a somewhat precise notion of truth for these statements: validity modulo the theory of Example 2.57.

This approach of finding axioms to indirectly describe the objects we work with is the foundation of modern mathematics. It is used to precisely define the natural numbers, the real numbers, or sets, and to formally define other classes of mathematical objects like groups.

# 3 | Proofs and Deductions

To do mathematics, we need to figure out whether our statements are true. In the last chapter, we already defined what it means for a statement to be true. This definition, however, is hard to work with in practice.

For example, consider the statement $P \wedge Q \to Q \wedge P$. We can quickly see that this should be true for all possible values of $P, Q$, since conjunction is commutative. However, our current notion dictates that we enumerate all possible combinations, for example in a truth table. While this is possible for propositional logic, this quickly becomes impossible for first-order logic.

For instance, let's say we wanted to figure out whether the following formula is true:

$$(\forall x, y : P(x, y) \to Q(y, x)) \wedge (\forall x : P(x, x)) \to \forall y : Q(y, y)$$

Our definition of truth requires we check this formula in all possible universes, and for each universe, consider every possible individual of that universe. This is of course impossible since there are infinitely many universes, each of which may contain infinitely many individuals.

Yet, our intuition tells us that the formula is true in all universes: This is because if $P(x, x)$ holds for all $x$, and since $P(x, x)$ means that $Q(x, x)$ is also true for any $x$, we can conclude that $Q(y, y)$ holds for any $y$.

A proof is a formal argument that tries to explain why something is true, using reasoning like that presented in the above paragraph. Instead of trying to manually check our formula in every possible universe, we attempt to reason logically about all universes all at once.

This is, of course, problematic: How do we avoid mistakes when arguing? How can we be absolutely sure that we are not misled when reading arguments made by other people, or worse, how do we avoid misleading ourselves?

Over time, mathematicians have figured out a system of reasoning that is generally agreed to not allow such mistakes. This system involves strict limits on the kind of arguments that are allowed. Thus, when following this system, we can be sure that we have not made mistakes in our reasoning. Further, other mathematicians can rely on our results. This chapter introduces this system as well as how to use it to find and write down proofs.

**Chapter Goals**

In this chapter, we discuss how statements are proven in mathematics. This includes

- Proof systems in general
- The proof system used for first-order logic
- How proofs are communicated among mathematicians

In the end, you will be able to write proofs yourself and read proofs written by others.

Before we start, we need to add a disclaimer: There are many different ways people write proofs, and even more ways they find these proofs. The method chosen here is useful because it shows all the basic building blocks, while still being reasonably close to one variety of "real" proofs. Later in your academic career, you might find that proofs are often presented differently: For example, they can be written seemingly backwards, or have any number of stylistic variations. By then, you will be experienced enough to understand these proofs. While learning, it is best to stick to one style, preferably the one presented here.

## 3.1 A Proof System for Propositional Logic

Usually, proofs are written in natural language, interspersed with some formal mathematical notation. When written like this, proofs have a rather peculiar style of writing, as they assume a reader proficient in reading and writing proofs. People not familiar with this kind of language usually miss what the proof is actually doing, why it's true, or where there might be errors in it.

Put differently, a natural language proof merely describes the actual logical argument that makes the proof work. We must understand the actual way a proof is built before we can start to prove statements on our own.

### 3.1.1 What Makes a Proof

Consider the statement $P \land Q \to Q \land P$. We have already seen that this statement is true, and by now, you should be able to recognize this rather quickly without iterating all possible evaluations of $P$ and $Q$.

We now look at a proper natural language proof of that statement. Afterwards, we will analyze that proof in-depth to understand its structure.

**Lemma 3.1.** $P \land Q \to Q \land P$

*Proof.* To show the implication $P \land Q \to Q \land P$, we can assume that $P \land Q$ is true because if it were false, the implication would be trivially true. Since $P \land Q$ is assumed, we also know that both $P$ and $Q$ on their own are true. So $Q \land P$ remains to show. We show both sides separately:

$Q$: $Q$ holds since we have previously assumed it.

$P$: $P$ similarly holds since we have previously assumed it. □

**Checkpoint 3.2**

We encourage you to read this proof several times. Does it convince you that the lemma it is supposed to prove is actually true?
How is the proof structured? Are there several individual steps you can pick out? What does each of the steps do?
Notice that the conjunctions $P \land Q$ and $Q \land P$ are used entirely differently. What is the difference? Does it "feel right" to use them in that way?

Now, that proof was very verbose. Usually, we do not expect you to be *that* verbose when writing proofs. In fact, most mathematicians would consider the statement itself "obvious" or "trivial," and not even bother writing a proof at all. However, written out like this, the proof demonstrates most of the action happening during proofs in general:

A mathematical proof works by maintaining a set of assumptions towards a specific goal. During the proof, the goal and the assumptions might be changed in several ways. The goal and the assumptions define the next steps that can be taken. Let's analyze how both are manipulated during the above proof:

1. Initially, the goal is $P \land Q \to Q \land P$. Our collections of assumptions is empty.
2. We use the fact that the goal is an implication to assume $P \land Q$. The goal changes to $Q \land P$, the assumptions are $P \land Q$.
3. Looking at the assumption $P \land Q$, we can assume that either side is also true. The goal remains $Q \land P$, the assumptions are now $P \land Q$, $P$, and $Q$. Note that our previous assumptions remain.
4. Since the goal is $Q \land P$, we can show both sides of the conjunction separately. We get two new goals and the assumptions in both are the same as we had before.
   - 4.1. One goal is $Q$. The assumptions are $P \land Q$, $P$, and $Q$.
     - 4.1.1. Since $Q$ is both the goal and an assumption, we are done.
   - 4.2. The other goal is $P$. The assumptions are $P \land Q$, $P$, and $Q$.
     - 4.2.1. Since $P$ is both the goal and an assumption, we are done.

When writing a proof, at any point, we should precisely know what our goal is and what our assumptions are. This is also called the **proof state**. Conversely, reading a proof involves figuring out what the goal and the assumptions were at each step, and how the original author manipulated them.

So, what are goals and assumptions, actually? The **goal** is the formula we are currently trying to prove. The **assumptions** are formulas we "know" or rather assume to be true at the current state. What goals are provable highly depends on the current assumptions: At the end of the example proof above, we could prove $P$, because $P$ was an assumption. But on its own, without any assumptions, $P$ is not provable. This should not be surprising since $P$ on its own is not true in all universes. During our proof, we try to expand our assumptions to figure out more and more "true" statements. In the end, this hopefully concludes with us being able to show that our goal is a true statement.

To recapitulate, let us again write down the states during our example proof, but now with our new notation:

1. No assumptions, goal is $P \land Q \to Q \land P$
2. $P \land Q$ is assumed, the goal is $Q \land P$
3. $P$, $Q$, and $P \land Q$ are all assumptions, the goal is $Q \land P$
   - 4.1. $P$, $Q$, and $P \land Q$ are all assumptions, $Q$ is the goal
   - 4.2. $P$, $Q$, and $P \land Q$ are all assumptions, $P$ is the goal

You might notice that we had more steps before. This is because previously, we did not simply describe what the proof state was at every point. We also argued why we were allowed to go from one state to next and why we were done at the end. Now that we know which states a proof consists of, we can discuss the "moves" which allow us to move from one state to the next.

The proof above is already detailed enough so that each step is atomic. The notion of an atomic step is important: every proof can be decomposed into a sequence of simple atomic steps, which can not be divided further. There only is a small number of different kinds of atomic steps, or *proof rules*. We continue by discussing these rules. For now, we use these rules very explicitly, one after the other, in order to understand how a proof is built. Later on, when writing proofs in natural language, we might use several rules at once. But before we can do so, we must understand the individual rules, and how they can be combined with each other.

So, the first and (almost) easiest rule is the one we used for concluding our proof. Remember how we finally managed to show $Q$:

$Q$ holds since we have previously assumed it.

The rule we used here is called the Assumption rule:

Assumption: To prove $\varphi$, have $\varphi$ in the assumptions.

So, this rule tells us that we can prove a formula $\varphi$ if it is in the assumptions. $\varphi$ here is a meta-variable, meaning that this rule can be used to prove any formula, as long as that specific formula is part of the assumptions.

It is important to think about why the Assumption rule can be used without causing problems, like allowing us to prove false statements. Formally, a rule that can be used without causing problems is called **sound**. Assumption is sound since assumptions are formulas we already consider to be true, and we want to show that the goal is true, which it must be if it also appears in our assumptions.

For every rule we learn, we must make sure that it is sound. Otherwise, if we use a wrong rule, we might be able to prove formulas which are not actually true, which would make proving things pointless. The nice idea about the rules we lay down now is that we only need to think about why they are correct once. In fact, if you trust us,[1] you do not need to think about this at all. As long as you stick to these rules, you will not prove anything invalid.

Working backwards through our proof, we next describe the rule that allowed us to prove both sides of the conjunction separately:

AndIntro: To prove $\varphi \land \psi$, prove $\varphi$ and $\psi$ separately.

This rule codifies the move we did when we proved the conjunction $P \land Q$ by proving each side on its own. Again, $\varphi$ and $\psi$ are meta-variables, so this rule can be used to prove any conjunction. In the above example, we used it with $\varphi := P, \psi := Q$. The rule is sound since, if we can prove that both $\varphi$ and $\psi$ hold, then clearly their conjunction must also hold.

Unlike the Assumption rule, this rule does not **conclude** the proof. This means that the proof is not done after using this rule. Instead, we get two new **proof obligations**. This means that we have two new goals (more formally: two new proof states – sometimes, the goal remains the same and only the assumptions change), for which we must continue building a proof.

Note that the rule does not mention the assumptions. In general, when a rule does not mention the assumptions, they remain unchanged and get carried over to new proof obligations. So, we

[1]A wise woman once said: Trust, but verify!

have two new proof obligations, each of which has a new goal, but both still have the same assumptions.

The next rule actually changes the assumptions. Here it is:

ImplIntro: To prove $\varphi \to \psi$, prove $\psi$ while assuming $\varphi$.

In this rule, we prove an implication $\varphi \to \psi$ by assuming $\varphi$. This means that we add $\varphi$ to our assumptions. Then, we continue to prove $\psi$ under our new assumptions (which are the old ones in addition to $\varphi$).

We use this rule at the very beginning of the proof of $P \land Q \to Q \land P$, where we assume that $P \land Q$ is true and continue to prove $Q \land P$. There, we also already argued why this rule is in general: We know that $\varphi$ is either true or false. If it is false, then the implication is true since the precondition is false. So, the only interesting case is when $\varphi$ is true. Then, $\psi$ must also be true. So $\psi$ is the new proof obligation, but since we know $\varphi$ must be true, we can assume it.

This rule hints at a deep connection between implication and assumptions, which we discuss later.

We have now described three out of four rules used in the proof above. One remains:

AndElim: If $\varphi \land \psi$ is assumed, also assume $\varphi$ and $\psi$.

This rule does not read like the previous rules, since it does not start with "to prove." It also does not mention a goal. This is because this formula does not care about the goal—it can be used on any goal and does not change it. Instead, it manipulates assumptions. The rule allows us to analyze the formulas we previously had assumed: if we have assumed a conjunction $\varphi$ and $\psi$, then this rule tells us that we can also assume $\varphi$ and $\psi$ separately, so it adds two new assumptions. Importantly, it does not remove the assumption $\varphi \land \psi$.[2]

If we wanted to reformulate this rule to sound like the others, we might write:

Alternate AndElim: To prove $\chi$ when $\varphi \land \psi$ is assumed, prove $\chi$ also assuming $\varphi$ and $\psi$.

By using the meta-variable $\chi$ for the goal, we make it explicit that this rule does not change the goal. Lastly, we need to argue why this rule can be used: If $\varphi \land \psi$ is assumed, we know that this conjunction is true. The only way for a conjunction to be true is by both sides being true. Thus, we can assume either side individually.

Using our rules, we can now write down the initial natural-language proof like this.

1. No assumptions, goal is $P \land Q \to Q \land P$, continue with ImplIntro
2. $P \land Q$ is assumed, the goal is $Q \land P$, continue with AndElim on $P \land Q$
3. $P$, $Q$, and $P \land Q$ are all assumptions, the goal is $Q \land P$, continue with AndIntro
    4.1. $P$, $Q$, and $P \land Q$ are all assumptions, $Q$ is the goal, conclude with Assumption using $Q$

[2]We could have formulated the rule so that it removes the existing assumption. This would make things more complicated later on. Since we have not done so, the rule as is now can be used several times on the same assumption.

4.2. $P$, $Q$, and $P \land Q$ are all assumptions, $P$ is the goal, conclude with Assumption using $P$

This proof is the same proof as the original natural language proof. It uses the same rules, in the same order. However, it is much more *explicit* than the natural language one. It shows the proof state at each step and makes each rule explicit.

Note that for the AndElim rule, we additionally noted the assumption we used that rule on. Especially for rules like AndElim, which only manipulate the assumptions, it can be very hard to keep track of what is happening when the context becomes large. A valid proof must thus not only mention the rules but also make clear how the meta-variables are instantiated. In practice, for rules like ImplIntro, this becomes obvious by looking at the goal. However, as a general convention, whenever a rule uses assumptions, you should write down the precise assumption used.

For Assumption, this seems redundant, since the assumption is precisely the goal. You will see why the convention of *always explicitly state the used assumptions* is necessary once your proofs become larger.

## 3.1.2 Proof Tables

If we look at the proof we wrote down at the end of the previous sub-chapter, we can see that it is very verbose. We have to write down the context at each step, and since the context only gets larger, this can quickly become problematic. The way to fix this is by inventing new notation. Concretely, we now introduce proof tables, which avoid having to explicitly copy everything all the time.

**Proof tables** are tables containing the successive proof states, as well as the rules allowing us to transition between them. A proof table has three columns:

| Context | Goal | Rule |
|---|---|---|
| | | |

For now, the context column contains our assumptions, while the goal column contains the current goal. The rule column will denote which rule we used to move from one proof state to the next.

Let's look at the example from before:

| Context | Goal | Rule |
|---|---|---|
| | $P \land Q \to Q \land P$ | ImplIntro |
| H1 : $P \land Q$ | $Q \land P$ | AndElim H1 |
| H2 : $P$<br>H3 : $Q$ | $Q \land P$ | AndIntro |

| Context | Goal | Rule |
|---|---|---|
| | $Q$ | Assumption H3 |

| Context | Goal | Rule |
|---|---|---|
| | $P$ | Assumption H2 |

When you compare this to the previous attempt, you might see some similarities. We start with an empty context, and the goal is the formula we want to prove. Next, we apply the rule ImplIntro and get to the next state. We have the new assumption and the new goal.

A new feature is that our assumptions get names. This will make using them in rules much easier, and also allows us to better keep track of them in general.

We next use the assumption we have just obtained in AndElim. This leads us to the next new feature: the context cell in that row only contains the new assumptions, instead of all of them. We try to save space by recognizing that our context never gets smaller, so we can always say that our context implicitly includes everything in the lines above. This avoids repeating the context every time, however, we must remember to look at the context above.

So far, this seems reasonable. The next rule, AndIntro, is a bit more complicated: Our table splits up into two new tables. This is because AndIntro has two proof obligations: the left and the right side of the conjunction. Unfortunately, tables can not be split nicely, so we start two new tables. We use arrows to connect the new tables to the previous table, which was split. This is important because the context is inherited from the old table: When we next use the Assumption rule, we can also use every assumption that we introduced before the split. If we had introduced more assumptions after the split, we would of course allowed to use them, too. What we are not allowed to do is use assumptions from the left table in the right table, so while both tables start out with the same assumptions inherited from their origin table, they continue independently. At the end, we use a double bottom line to denote that we are done.

We now denote e.g. which assumption the AndElim rule was applied to using the names introduced before. The procedure for checking the proof, including checking that all claimed assumptions are actually part of the assumptions, remains the same, however, we now know precisely which hypotheses are used in any rules. Remember that for checking whether a referenced assumption is in fact in the context, you can go upward in a table, or follow the arrows *backwards*.

Note that the names of our hypotheses can be chosen arbitrarily, but in order to avoid confusion later, we try to stick to names always starting with a capital `H` (for hypothesis). While this script tends to just number the hypotheses, you can sometimes pick more descriptive names. For example, induction hypotheses (introduced in Chapter 5) are named `IH`, and otherwise you might name hypotheses based on their origin. For example, if you have several hypotheses all describing a person named Anna, you might call these hypotheses `HAnna1` to `HAnna3`. Note that no two hypotheses may share a name.[3]

A slightly different proof of the same formula is shown in Figure 3.3. The difference there is that we first decide to prove both sides of the conjunction, and only then look at the context to figure out that we know either side is already true. While this proof is longer than the other one (we have to use AndElim twice), it is still correct.

Our new language for writing down proofs can also be used to specify the proof rules. This is how we denote the Assumption rule:

| Context | Goal | Rule |
|---|---|---|
| $\varphi$ | $\varphi$ | Assumption |

[3]You can have two hypotheses with the same name in different sub-proofs, where they do not interfere with each other. This is fine and sometimes useful, but it can also lead to confusion. In our examples, we sometimes reuse names when the assumptions are actually the same in different subproofs, but avoid doing so otherwise.

| Context | Goal | Rule |
|---|---|---|
| | $P \land Q \to Q \land P$ | ImplIntro |
| H1 : $P \land Q$ | $Q \land P$ | AndIntro |

$\swarrow \qquad \searrow$

| Context | Goal | Rule |
|---|---|---|
| | $Q$ | AndElim H1 |
| H2 : $P$<br>H3 : $Q$ | $Q$ | Assumption H3 |

| Context | Goal | Rule |
|---|---|---|
| | $P$ | AndElim H1 |
| H2 : $P$<br>H3 : $Q$ | $P$ | Assumption H2 |

Figure 3.3: A slightly different proof of $P \land Q \to Q \land P$

This means that as long as $\varphi$ is both the goal and *anywhere* in the context, this rule allows us to finish the proof. So, the context line has a slightly different meaning when we define a rule since we in particular do not require that $\varphi$ was introduced by the rule used immediately prior. The double bottom denotes that this rule concludes the proof. Also, when specifying our rules, we do not care about the names of assumptions. Instead, we rely on the convention that if a rule uses assumptions (like $\varphi$), then, when using that rule, we must specify which specific assumption was used here.

Next, the ImplIntro rule:

| | | |
|---|---|---|
| | $\varphi \to \psi$ | ImplIntro |
| $\varphi$ | $\psi$ | |

In this rule, we make no requirements about the context. All we care about is that our goal has a particular shape. Also, the ImplIntro rule does not finish the proof, so it introduces a new obligation below it. This rule extends the context, adding $\varphi$ to it, and also changes the goal to $\psi$.

We continue with the AndElim rule:

| | | |
|---|---|---|
| $\varphi \land \psi$ | $\chi$ | AndElim |
| $\varphi$<br>$\psi$ | $\chi$ | |

By now, you should be able to explain this rule: It expects $\varphi \land \psi$ somewhere in the context, and then adds $\varphi$ and $\psi$ to the context. It does not change the goal.

Finally, AndIntro:

| | | |
|---|---|---|
| | $\varphi \land \psi$ | AndIntro |

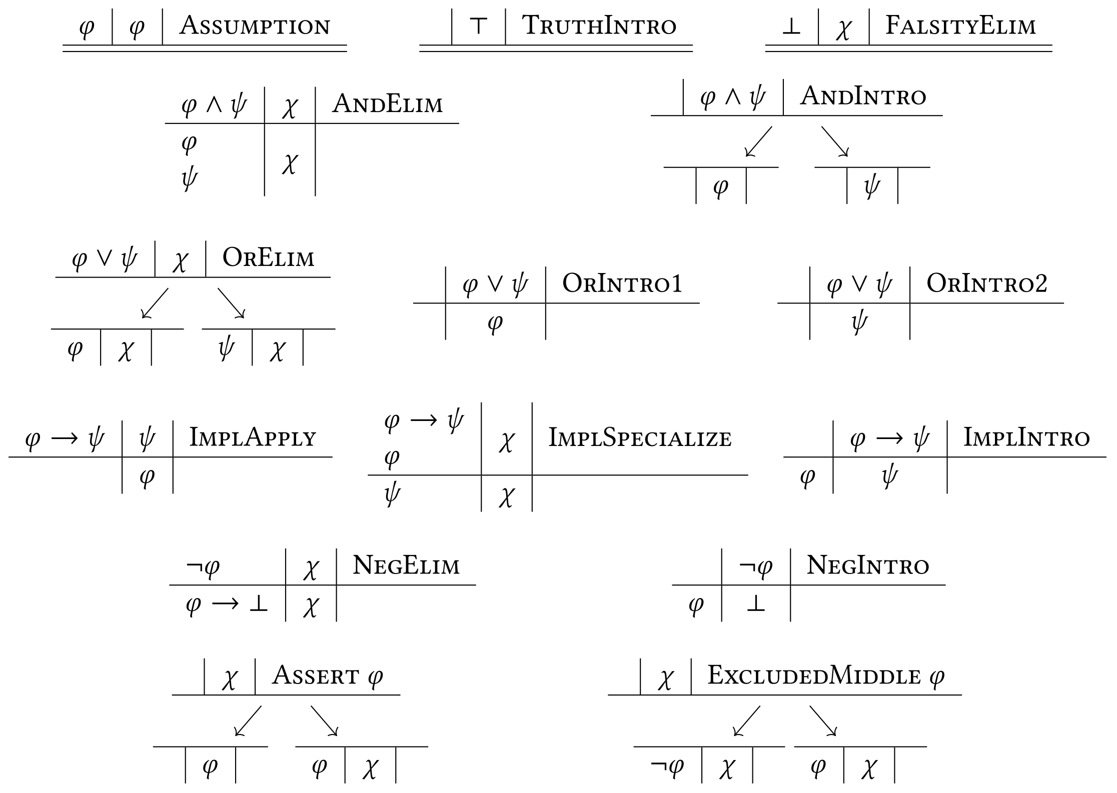
$\swarrow \qquad \searrow$

| | | |
|---|---|---|
| | $\varphi$ | |

| | | |
|---|---|---|
| | $\psi$ | |

This rule has two subgoals, so we split our table up. In one, $\varphi$ is the new goal, in the other, it is $\psi$. The context remains unchanged.

Finally, we can state all the rules for propositional logic:



Okay, let's discuss these rules. First, we can notice some patterns: For each connective of propositional logic, there are (usually) introduction and elimination rules. Introduction rules allow us to prove a formula. Elimination rules allow us to use an assumption. Introduction rules do not care about what is in the context. Elimination rules (usually) do not affect the goal.

Note that an introduction rule also seems to "eliminate" the goal into smaller sub-goals. However, this is not actually what happens, and thinking about introduction rules like this makes it easy to confuse them with elimination rules. What an introduction rule actually does is allow us to gradually build up a proof from proofs of sub-terms: If we have proofs of $\varphi$ and $\psi$, the AndIntro rule allows us to introduce the operator $\land$ and yields a proof of $\varphi \land \psi$.

Conversely, the elimination rules actually "deconstruct" a proof of a combined formula (like $\varphi \land \psi$). In a way, they allow us to break apart the fact that a formula holds into facts about the sub-formulas of that formula.

Introduction rules are also sometimes called "construction rules." Elimination rules can also be called "deconstruction" or "destruction" rules. Here, we chose the more traditional names.

Additionally, we have three extra rules that do not correspond to any logical connective. Let's start with these.

The first is Assumption, which we already know. The next is Assert, which allows us to create an arbitrary subgoal. This subgoal must first be proven, but we can then use this as an assumption later on. This allows us to organize our proofs by first finding some simpler goal, then working towards it and finally using it to proceed in the remainder of the proof.

The last unusual rule is ExcludedMiddle. This rule is important: It allows us to use the fact that in propositional logic, everything is either true or false. So, this rule is just a formulation of the law of excluded middle.

With this out of the way, let's focus on the rules for logical connectives. It makes sense to think of these rules as answers to the questions "How do you prove a conjunction/disjunction/...?" and "How do you use a ...?" For conjunction, we know we prove a conjunction by proving both sides independently. Similarly, we can use a conjunction to extract the knowledge that both sides of it are true.

Disjunction works similarly. We have two introduction rules: OrIntro1 allows us to prove a disjunction using the left side, while OrIntro2 allows us to prove it using the right side. This corresponds to our intuitive understanding of introduction: It is true as soon as either side is true. You might wonder why there is no rule that allows us to prove a disjunction by proving both sides. The answer is that this is not necessary: If both sides are true, we can use either of our existing introduction rules and just prove the side that is easier.

The elimination rule for disjunction performs a case distinction. We know that at least one of the sides is true, but we don't know which. Therefore, we have to prove our goal twice, once assuming the left side is true and once assuming the right side is true.

The introduction rule for $\top$ just says that $\top$ is always true. There is no elimination rule for $\top$, because we can not really use $\top$—we gain no new information from having $\top$ in our assumptions.

The rule for $\bot$ is notoriously hard to understand. We know that $\bot$ is always false. Thus, its stands to reason that *there is no proof of* $\bot$. However, if we have assumed $\bot$, we are in a situation quite similar to proving an implication where the precondition is false. When that happens in an implication, the whole implication is true, and for similar reasons, we also allow proving any goal when we have $\bot$ in our assumptions. This rule is commonly known as *ex falso quodlibet* (Latin for: "from falsity, everything follows"). We discuss this further when we look at proofs by contradiction.

The rules for negation parallel our rule for $\bot$. The introduction rule says that if we want to prove $\neg\varphi$, we can assume $\varphi$ to prove $\bot$. The only way to prove $\bot$ is using FalsityElim or Assumption, thus it is only provable if $\varphi$ itself is false. The elimination rule makes this relation even more explicit: It simply converts an assumption $\neg\varphi$ into one of shape $\varphi \to \bot$. This works, since both are equivalent in our logic. So, since we already know how to use implication and $\bot$, we can use those rules to also work with negation. In fact, sometimes people consider $\neg\varphi$ to just be syntactic sugar for $\varphi \to \bot$.

The implication introduction rule was already discussed. Interestingly, here, we have two elimination rules: The first, ImplApply, allows us to prove a goal $\psi$ by proving $\varphi$, as long as we know that $\varphi \to \psi$. The second, ImplSpecialize, allows us to add the consequence $\psi$ of an implication $\varphi \to \psi$ to the context as long as we already know that $\varphi$ is true. In fact, we only need one of the rules, as the other can be derived using Assert. The ImplSpecialize rule is also known as **modus ponens**, which states the core idea behind an implication: If $\varphi \to \psi$ and $\varphi$ are true, we can derive $\psi$.

### 3.1.3 Example Proofs

Our proofs are a so-called formal system. This means that we have tried to use formal mathematical language to make everything very precise. We did this by presenting a lot of rules.

This makes formal systems easy to present: Just write down all the rules. However, learning a new formal system is not as easy. Just learning all the rules by heart will not be sufficient and is also not very practical. Instead, we recommend that you practice using the rules until you get a “feeling” for how they can be used. This way, you don’t even have to remember the rules, since they will eventually feel so natural that you can just re-derive them on the spot.

Thus, we will now showcase several proofs using our system so that you can get a feel for how the rules are used. Again, just learning the examples by heart will not be sufficient. Instead, try practicing writing down these proofs yourself, and then comparing them with the example.

So, let’s start. We have already seen a proof that conjunction is commutative. A proof of the same property for disjunction can be found in Figure 3.4.

| Context | Goal | Rule |
|---|---|---|
| | $P \lor Q \to Q \lor P$ | ImplIntro |
| H1 : $P \lor Q$ | $Q \lor P$ | OrElim H1 |

| Context | Goal | Rule |
|---|---|---|
| H2 : $P$ | $Q \lor P$ | OrIntro2 |
| | $P$ | Assumption H2 |

| Context | Goal | Rule |
|---|---|---|
| H3 : $Q$ | $Q \lor P$ | OrIntro1 |
| | $Q$ | Assumption H3 |

Figure 3.4: A proof table using OrElim

Figure 3.5 presents another example, which has implications as assumptions.

| Context | Goal | Rule |
|---|---|---|
| | $(P \to Q \to R) \to (P \land Q \to R)$ | ImplIntro |
| H1 : $P \to Q \to R$ | $P \land Q \to R$ | ImplIntro |
| H2 : $P \land Q$ | $R$ | AndElim H2 |
| H3 : $P$<br>H4 : $Q$ | $R$ | ImplSpecialize H1 H3 |
| H5 : $Q \to R$ | $R$ | ImplApply H5 |
| | $Q$ | Assumption |

Figure 3.5: A proof table using ImplIntro

For yet another example, using implications and disjunctions, see Figure 3.6.

The rules for truth and falsity are rather straightforward. We first consider Figure 3.7, which demonstrates how truth is proven. This proof should not be too surprising, since TruthIntro is a rule even simpler than Assumption.

Now, let’s turn to a proof of a closely related statement in Figure 3.8, which uses falsity elimination.

In this proof, we considered the two possible cases of $P \lor \bot$. Either $P$ holds, or $\bot$. The first case is easy. In the case that $\bot$ holds, we get $\bot$ as an assumption. Intuitively, this means that the case we are currently in is “impossible.” While it came up in our case distinction, it can not actually happen “in practice,” because $\bot$ is never true. So, we basically just stop with our proof. This is

| Context | Goal | Rule |
|---|---|---|
| | $(P \lor Q \to R) \to (P \to R) \land (Q \to R)$ | ImplIntro |
| H1 : $P \lor Q \to R$ | $(P \to R) \land (Q \to R)$ | AndIntro |

| Context | Goal | Rule |
|---|---|---|
| | $P \to R$ | ImplIntro |
| H2 : $P$ | $R$ | ImplApply H1 |
| | $P \lor Q$ | OrIntro1 |
| | $P$ | Assumption H2 |

| Context | Goal | Rule |
|---|---|---|
| | $Q \to R$ | ImplIntro |
| H3 : $Q$ | $R$ | ImplApply H1 |
| | $P \lor Q$ | OrIntro2 |
| | $Q$ | Assumption H3 |

Figure 3.6: A proof table using implications and disjunctions

| Context | Goal | Rule |
|---|---|---|
| | $P \to P \land \top$ | ImplIntro |
| H1 : $P$ | $P \land \top$ | AndIntro |

| Context | Goal | Rule |
|---|---|---|
| | $P$ | Assumption H1 |

| Context | Goal | Rule |
|---|---|---|
| | $\top$ | TruthIntro |

Figure 3.7: A proof table proving truth

| Context | Goal | Rule |
|---|---|---|
| | $P \lor \bot \to P$ | ImplIntro |
| H1 : $P \lor \bot$ | $P$ | OrElim H1 |

| Context | Goal | Rule |
|---|---|---|
| H2 : $P$ | $P$ | Assumption H2 |

| Context | Goal | Rule |
|---|---|---|
| H3 : $\bot$ | $P$ | FalsityElim H3 |

Figure 3.8: A proof table using falsity elimination

exactly what the FalsityElim rule does: It allows us to stop the proof since whatever we are trying to prove right now is within an unreachable context.

Of all the rules in our system, the FalsityElim rule is often considered the most confusing rule, so if you feel confused by that proof, that is perfectly normal.

Reasoning with falsity is closely linked to reasoning with negations. In proofs, $\neg P$ is handled by transforming it to $P \to \bot$ using the rule NegElim. Similarly, NegIntro behaves like applying ImplIntro on this. So sometimes, reasoning with negations can be relatively simple, since it almost is like reasoning with implications. The example in Figure 3.9 shows this: There, we prove one direction of the contraposition law. This direction is rather straightforward to prove, since it just involves reasoning with implications.

To finish off this series of examples, let's do a proof using the ExcludedMiddle rule. This one, to be found in Figure 3.10, looks large and rather complicated. There certainly is a lot going on

| Context | Goal | Rule |
|---|---|---|
| | $(P \to Q) \to (\neg Q \to \neg P)$ | ImplIntro |
| H1 : $P \to Q$ | $\neg Q \to \neg P$ | ImplIntro |
| H2 : $\neg Q$ | $\neg P$ | NegElim H2 |
| H3 : $Q \to \bot$ | $\neg P$ | NegIntro |
| H4 : $P$ | $\bot$ | ImplApply H3 |
| | $Q$ | ImplApply H1 |
| | $P$ | Assumption H4 |

Figure 3.9: A proof table using NegElim and NegIntro

there. Importantly, three rules are used for the first time.

ExcludedMiddle is used near the beginning. Here, ExcludedMiddle allows us to show that the implication $P \to Q$ either holds or does not hold. We need to handle these two cases, and the right one (where $P \to Q$ does not hold) is the more interesting one. In this, we know that the right side of the disjunction must be true. This allows us to introduce a $P$, so we can deduce $Q \lor R$ from our initial assumption. Now, the intuition is that if this $Q \lor R$ is $Q$, then our initial implication was just $P \to Q$, but we already know that this is false. So, if after case-analyzing this, we get $Q$, we are able to use our previous assumption that $P \to Q$ does not hold.

Here, the rule NegElim appears, which we have also never used before. NegElim here allows us to get an implication with $\bot$ as the post-condition. Our next steps are to get access to that $\bot$ so that we can apply FalsityElim. The overall principle here is that of a proof by contradiction, which we explain shortly.

When doing so, we run into a somewhat unpleasant situation. We have an implication towards $\bot$ and would like to use it. However, ImplApply can not be used, since our goal is not $\bot$. Similarly, ImplSpecialize can not be used, since we do not have $P \to Q$ in our assumptions. The solution is to use Assert to introduce a helpful intermediate goal, where we can apply ImplApply.

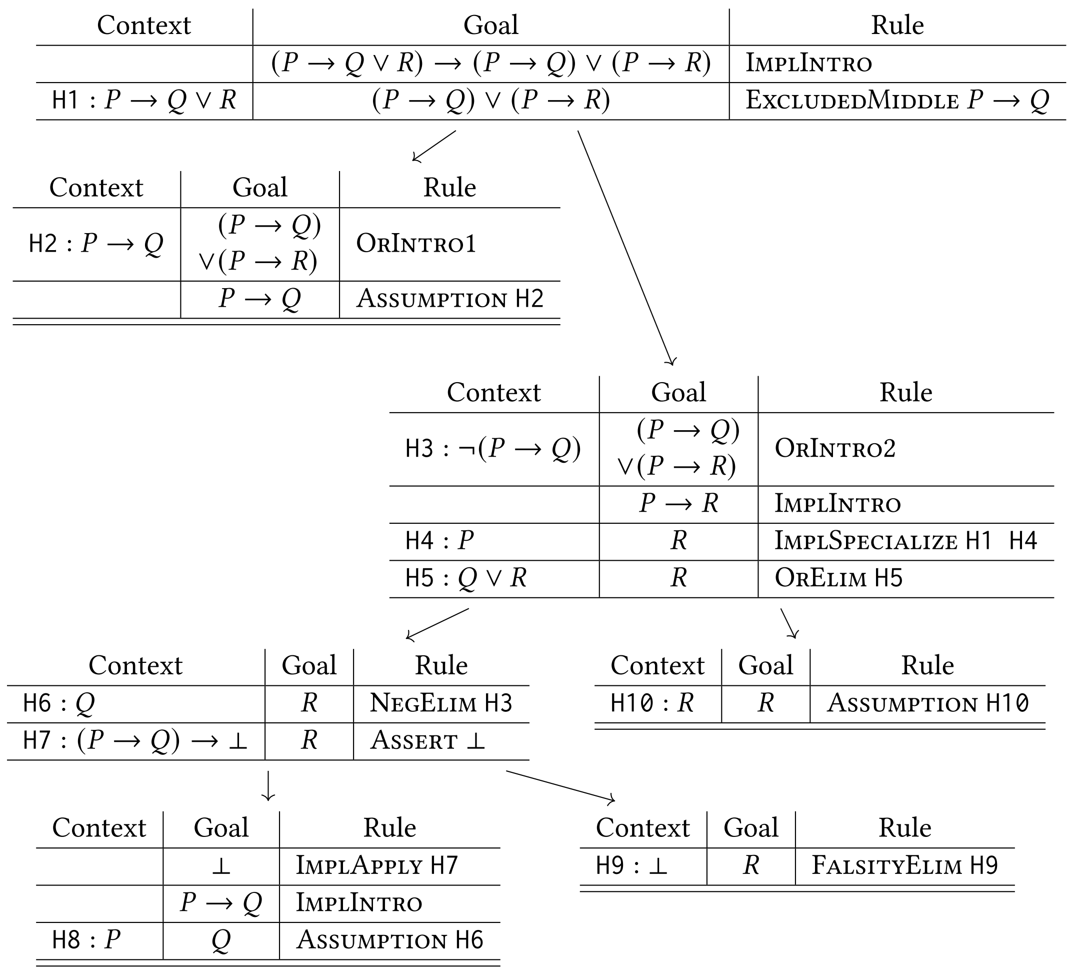
The proof also shows another important aspect of the proof system: There are formulas that are provable, but the proof is not direct. Here, we must cleverly apply the ExcludedMiddle rule to figure out which side of the disjunction we should try to prove. In the second case, we must do some work and use the assumption that $\neg(P \to Q)$ to eventually reach a case where we can derive $\bot$. Finding such proofs can be very difficult, especially if one is not sure whether the formula to be proven even is true.

Our proof system has a rather minimal number of rules. In fact, there is only one unnecessary rule. All other rules are necessary to make the proof system complete (see Theorem 3.18). Therefore, it is often necessary to use Assert to prove some intermediary goal, especially in order to then apply rules like ImplSpecialize or FalsityElim.

### 3.1.4 Proof Strategies

Now that we are able to use the basics of our proof system, we can focus on how you actually find these proofs. Unfortunately, this chapter can not teach this—proving is again something learned only by repeated practice.

However, we can discuss some common strategies and patterns used in proofs.



| Context | Goal | Rule |
|---|---|---|
| | $(P \to Q \lor R) \to (P \to Q) \lor (P \to R)$ | ImplIntro |
| H1 : $P \to Q \lor R$ | $(P \to Q) \lor (P \to R)$ | ExcludedMiddle $P \to Q$ |

| Context | Goal | Rule |
|---|---|---|
| H2 : $P \to Q$ | $(P \to Q)$ $\lor(P \to R)$ | OrIntro1 |
| | $P \to Q$ | Assumption H2 |

| Context | Goal | Rule |
|---|---|---|
| H3 : $\neg(P \to Q)$ | $(P \to Q)$ $\lor(P \to R)$ | OrIntro2 |
| | $P \to R$ | ImplIntro |
| H4 : $P$ | $R$ | ImplSpecialize H1 H4 |
| H5 : $Q \lor R$ | $R$ | OrElim H5 |

| Context | Goal | Rule |
|---|---|---|
| H6 : $Q$ | $R$ | NegElim H3 |
| H7 : $(P \to Q) \to \bot$ | $R$ | Assert $\bot$ |

| Context | Goal | Rule |
|---|---|---|
| H10 : $R$ | $R$ | Assumption H10 |

| Context | Goal | Rule |
|---|---|---|
| | $\bot$ | ImplApply H7 |
| | $P \to Q$ | ImplIntro |
| H8 : $P$ | $Q$ | Assumption H6 |

| Context | Goal | Rule |
|---|---|---|
| H9 : $\bot$ | $R$ | FalsityElim H9 |

Figure 3.10: A proof table using ExcludedMiddle, Assert, and NegElim

### Chasing Contradictions

When finding proofs, we often like to call $\bot$ a **contradiction**. The main idea behind chasing contradictions is to try to derive additional hypotheses until we reach a contradiction. In our proof system, this means that we try to continue in our proof, adding more and more hypotheses, until these hypotheses either allow us to derive $\bot$ or, like in the proof in Figure 3.10, we can eliminate a negation and thereby prove our goal.

This was the strategy we used in that proof—we tried to chase a contradiction by trying to collect assumptions until we had enough assumptions to contradict $\neg(P \to Q)$.

Intuitively, being able to derive $\bot$ can be understood as being in an impossible situation. For example, in our earlier proof of $P \lor \bot \to P$, we have two options: One is that $P$ holds, the other is that $\bot$ holds. The idea is that the right side can never be true. Yet, we still need to do our case distinction, because this is what the rules require us to do. Thus, falsity elimination provides an "escape mechanism" to get out of impossible proof obligations. Concretely, we use it to quickly finish the $\bot$ side of $P \lor \bot$, since that is never true.

In the earlier example, we similarly used it after we arrived at the point where we managed to derive $Q$ from $P$, even though $\neg(P \rightarrow Q)$ was in the context. This means that deriving $Q$ from $P$ should be impossible. So, to get out of the seemingly impossible case, we again use falsity elimination.

## Proof by Contradiction

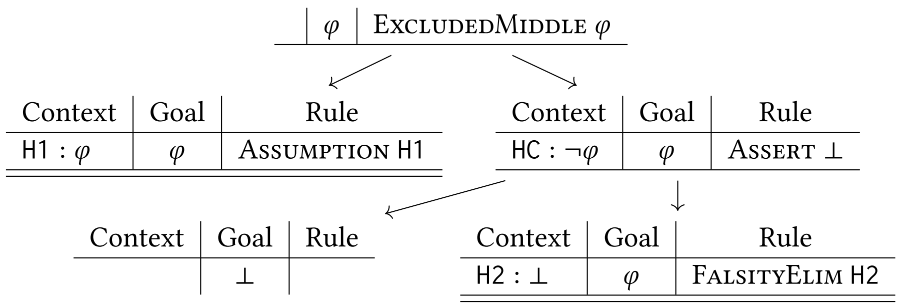
Often, when proving $P$, it is easier to instead assume $\neg P$, and try to prove a contradiction (i.e. try to prove $\bot$).

We then know that $\neg P$ can not be true, so $P$ must be true.

This corresponds to the following proof rule:

| | $\varphi$ | CONTRADICTION |
|---|---|---|
| $\neg\varphi$ | $\bot$ | |

Before we can use this rule, we need to check that it is actually sound. For our foundational rules, we did this by appealing to our intuition on propositional logic. Here, we can be much more formal, and simply give a scheme that shows how to build our new rule from our already known rules, like in Figure 3.11. There, we use our already existing rules, starting in a context similar to the CONTRADICTION rule. However, the proof is not done, there is an unfinished subgoal remaining. That subgoal corresponds to the new state generated by the CONTRADICTION rule. In that new subgoal, we have a new hypothesis `HC` : $\neg\varphi$, and the CONTRADICTION rule requires us having exactly[4] that hypothesis. Thus, the CONTRADICTION rule is correct, since we can always replace it with rules we are already sure are true.



$\varphi$ | EXCLUDEDMIDDLE $\varphi$
Context | Goal | Rule
H1 : $\varphi$ | $\varphi$ | ASSUMPTION H1
Context | Goal | Rule
HC : $\neg\varphi$ | $\varphi$ | ASSERT $\bot$
Context | Goal | Rule
$\bot$
Context | Goal | Rule
H2 : $\bot$ | $\varphi$ | FALSITYELIM H2

Figure 3.11: A justification of the CONTRADICTION rule

Notice that in this proof, there is one goal remaining. Also, that goal has access to the assumption `HC`, which is exactly the assumption it is required to have by our CONTRADICTION rule. So we know we can use that rule because we could always replace it by the proof building block of figure Figure 3.11.

Of course, that rule is much simpler, so you are allowed to use it everywhere from now on. Figure 3.12 shows how to use it. That proof is worth a closer look: We can see that after three steps, we reach the same goal we started with. However, we have picked up a crucial assumption

[4]The replacement needs to provide all the assumptions required by the new rule. If there are more assumptions, that is fine, since we can simply not use them.

| Context | Goal | Rule |
|---|---|---|
| | $P \vee \neg P$ | CONTRADICTION |
| H1 : $\neg(P \vee \neg P)$ | $\bot$ | NEGELIM H1 |
| H2 : $(P \vee \neg P) \to \bot$ | $\bot$ | IMPLAPPLY H2 |
| | $P \vee \neg P$ | ORINTRO2 |
| | $\neg P$ | NEGINTRO |
| H3 : $P$ | $\bot$ | IMPLAPPLY H2 |
| | $P \vee \neg P$ | ORINTRO1 |
| | $P$ | ASSUMPTION H3 |

Figure 3.12: A proof using CONTRADICTION

(H2), which we then use. We reach the goal $P \vee \neg P$ a third time, but this time we have picked up yet another assumption, namely H3, which now makes the proof very easy.

In the previous examples, we have seen that working with negation and $\bot$ is tedious. It turns out that the following rules make our lives much simpler:

| | | |
|---|---|---|
| $\neg\varphi$ | $\chi$ | NEGELIMAPPLY |
| | $\varphi$ | |

| | | |
|---|---|---|
| | $\chi$ | EXFALSO |
| | $\bot$ | |

The first rule combines NEGELIM, IMPLAPPLY, and ASSERT, relieving us of having to show a trivial side goal separately.

The second rule makes using the “escape mechanism” easier. If we, at any point, notice that our current situation is contradictory, we can immediately proceed to prove $\bot$.

**Checkpoint 3.13**

Construct proof trees justifying these rules!

### Equivalence Chains

So far, we have not touched upon the logical equivalence $P \leftrightarrow Q$.

Note that $P \leftrightarrow Q$ is just notation for $(P \to Q) \wedge (Q \to P)$. This means that we can always prove an equivalence by using ANDINTRO and then proving both directions separately. In *most* cases, this is how you should prove an implication.

While this always works, it can sometimes be unnecessarily tedious. Proving that $P \leftrightarrow Q$ is often much simpler by cleverly picking some $R$ so that $P \leftrightarrow R$ and $Q \leftrightarrow R$. In fact, this can be used multiple times: To prove $P \leftrightarrow Q$, we can just prove each intermediate implication in $P \leftrightarrow R_1 \leftrightarrow R_2 \leftrightarrow \cdots \leftrightarrow R_n \leftrightarrow Q$ holds.

We can also formulate this as a proof rule:

$$\begin{tikzcd} & {P \leftrightarrow R \mid \textsc{EquivChain}\ Q} \arrow[dl] \arrow[dr] & \\ {P \leftrightarrow Q} & & {Q \leftrightarrow R} \end{tikzcd}$$

**Checkpoint 3.14**

Construct a proof tree justifying this rule (use Assert)!

### 3.1.5 Metatheory

In Section 2.1, we discussed validity and satisfiability, and eventually arrived at the notion of truth that a formula is *true* if it is valid.

Another suitable notion of truth is provability:

> **Definition 3.15** (Provability). *A formula $\varphi \in \mathcal{F}_0$ is* ***provable*** *if we can construct a proof for it, starting in the empty environment.*

We have already given some intuition why all our proof rules are OK to use. Formally, this means that each proof rule *preserves truth.* When reading the proof backwards (starting with the last rule that is applied), the goal that is proven there should be a true statement. Then, as one continues backwards through the proof, the rules should only ever combine already true statements (i.e. the subgoals) into statements that remain true. Thus, whenever we have a proof of a formula, we know that this formula is valid (compare Corollary 3.17).

To prove this, it is not sufficient to look at the goal, but instead we must consider formulas that are "true under the assumptions." For this, we introduce a big implication into our proof.

**Lemma 3.16** (Soundness). *If we have a proof of $\varphi$ under assumptions $\psi_1, \ldots, \psi_n$, then the formula*

$$(\psi_1 \wedge \cdots \wedge \psi_n) \longrightarrow \varphi$$

*is valid.*

*If there are no assumptions, the formula is $\top \longrightarrow \varphi$.*

*Proof sketch*[5] *by induction on the deduction.* The general idea of this proof is that we show that each rule is sound. When doing so, we assume that we are working in a fully closed proof. So for AndElim, which opens a new goal, where we have $\varphi_1$ and $\varphi_2$ separately as new assumptions (in addition the the existing ones), we know that this new goal is also part of a fully closed proof. By the magic of induction (more on this in Chapter 5), we can then assume that soundness holds for this smaller sub-proof, so that the formula described by its initial proof state is valid. This means that the formula $(\psi_1 \wedge \cdots \wedge \psi_n \wedge \varphi_1 \wedge \varphi_2) \longrightarrow \chi$, which encapsulates the subgoal, can be assumed to be valid. There, $\psi_n = \varphi_1 \wedge \varphi_2$ is the hypothesis that AndElim is used on.

Since the large conjunction already contains $\psi_n = \varphi_1 \wedge \varphi_2$, it contains $\varphi_1$ and $\varphi_2$ "twice." So by idempotence (and commutativity, associativity), we can remove the latter occurrences, to just have $(\psi_1 \wedge \cdots \wedge \psi_n) \longrightarrow \chi$. This is thus also valid. But this is just the formula describing our original goal. So for any goal, when we use the rule AndElim on it (and then prove that the remaining proof ensures the formula is actually true), we get that this original goal (before applying the rule) is also true.

Basically, we peel of one rule after the other, and we always verify that this rule never proves anything wrong (while assuming that the subgoals are properly proven). By the magic of induction (see Chapter 5), this suffices to prove soundness.

For reference, here is the reasoning for the Assumption rule: This rule is applied to a proof state with assumptions $\psi_1, \dots, \psi_n$ and goal $\varphi = \psi_k$ for $1 \le k \le n$. We thus need that $(\psi_1 \wedge \cdots \wedge \psi_n) \longrightarrow \psi_k$, which is true because $\psi_k \wedge \cdots \longrightarrow \psi_k$ is always true (because if $\psi_n$ is false, the whole formula is immediately true).

The proofs of the other cases (one case per rule) are left to the reader. □

**Corollary 3.17.** *If $\varphi$ is provable (under the empty environment), then it is valid.*

Lemma 3.16 describes a deep connection between implication and assumptions: We can, at any time, represent our current proof state as a formula, where all the assumptions together imply our goal.

While we can easily show that any proven formula is valid, constructing a proof for any valid formula is harder. It is, however, possible.

**Theorem 3.18** (Completeness)**.** *If a formula $\varphi \in \mathcal{F}_0$ is valid, it is provable.*

*Proof sketch.* We use Theorem 2.23 and Lemma 2.18 and get a rewriting chain which, using the algebraic axioms of propositional logic, transforms our formula to $\top$.

We know that $\top$ is provable, and we can also prove every possible rewriting step. For this, it suffices to show that if $\varphi \leftrightarrow \psi$, then also any formula $\chi$ that contains $\varphi$ is equivalent to $\chi$, but with some occurrences of $\varphi$ replaced by $\psi$. What remains is to show that the (necessary) axioms of propositional logic are provable, which is left as an exercise to the reader. □

Thus, we know that a formula is valid if and only if it is true. This means that we have a new notion of truth, which coincides with the old one, while having a very different definition.

> **Definition 3.19** (Syntactic Truth)**.** *A formula $\varphi \in \mathcal{F}_0$ is* ***syntactically true****, also called* ***deductively true****, iff it is provable.*

**Theorem, Lemma, Corollary** In this section, you have seen a theorem, a lemma, and a corollary. All of these were statements that had a formal proof. In fact, you might wonder why we make a difference between these three. The answer is that there is no formal difference between them, each of them is just some statement with a proof attached.

The reason different names are used is that a **theorem** is usually considered *more important* than a **lemma**. For example, formally proving the completeness theorem above is much harder than proving the soundness lemma (which is why we only give a short sketch). It is also much more profound.

[5] You are not expected to fully understand this proof right now. It uses structural induction, see Section 5.4.

Usually, a math textbook (like this book) is structured by first defining the relevant terms. Then, there are several lemmas, usually in somewhat increasing difficulty, complexity, or abstractness. Eventually, these lemmas can be combined to prove some theorems, which are the "main goal" the textbook is aiming for, or otherwise more important than the other lemmas.

A **corollary** usually is a statement that is important on its own, but which is easily proven as a special case of the lemma or theorem immediately in front of it. For example, our Corollary 3.17 is a direct consequence of Lemma 3.16, when the list of assumptions is empty.

Finally, some authors have "proposition" as a separate category next to lemmas and theorem. There is no real difference between propositions and lemmas. Some authors use propositions for statements that are less important than lemmas, others make no discernible difference.

## 3.2 A Proof System for First-Order Logic

We now extend our proof system to first-order logic. The first important addition is that in first-order logic, we will be working with objects.

This complicates our proof system somewhat since, during our proof, we will not only be managing the assumptions, but also the objects which we can use. Our assumptions will also be able to reference objects.

Thus, our context will contain both assumptions and variables representing these objects. To prevent us from confusing objects and assumptions, we write objects in lower case letters, and propositions in upper case letters. Additionally, we can explicitly denote that something is an object by writing $x : \texttt{Object}$.

The proof system is best understand by looking at examples, like Figure 3.20.

| Context | Goal | Rule |
|---|---|---|
| | $(\forall x, y : P(x,y) \to Q(y,x))$ $\land (\forall x : P(x,x))$ $\to \forall y : Q(y,y)$ | ImplIntro |
| H1 : $(\forall x, y : P(x,y) \to Q(y,x))$ $\land (\forall x : P(x,x))$ | $\forall y : Q(y,y)$ | AndElim H1 |
| H2 : $\forall x, y : P(x,y) \to Q(y,x)$ H3 : $\forall x : P(x,x)$ | $\forall y : Q(y,y)$ | ForallIntro $z$ |
| $z$ : Object | $Q(z,z)$ | ForallElim H2 $z$ |
| H4 : $\forall y : P(z,y) \to Q(y,z)$ | $Q(z,z)$ | ForallElim H4 $z$ |
| H5 : $P(z,z) \to Q(z,z)$ | $Q(z,z)$ | ImplApply H5 |
| | $P(z,z)$ | ForallElim H3 $z$ |
| H6 : $P(z,z)$ | $P(z,z)$ | Assumption H6 |

Figure 3.20: Our first proof of a first-order formula

The first two steps of the proof are routine: We assume two formulas since they were on the left side of an implication. Then, we get to use the first quantifier rule, which is ForallIntro. This rule is used to prove $\forall y : Q(y,y)$. To understand this rule, as well as the other rules for quantifiers, one should look at them through the lens of the quantifier game of Section 2.2.3. In that game, a universal quantifier as part of a true statement meant that we could give some object to the opponent. Now, we are trying to prove that a universal quantifier is true, so the roles are reversed, we are the opponent, playing against the refuter. Thus, we now take an object $z$ for which $Q(z, z)$ needs to be shown.

Now, we get to use our assumptions. For these, we play the quantifier game as originally introduced, since assumptions are formulas we know to be true. We play it on $\forall x, y : P(x, y) \rightarrow Q(y, x)$, by putting in $z$ two times, to get $P(z, z) \rightarrow Q(z, z)$, which must be true. Later, we also play it on $\forall x : P(x, x)$, to get $P(z, z)$. The remainder is just using the appropriate rules for predicate logic.

### 3.2.1 Rules for Quantifiers

We've now seen how to use the two rules for universal quantifiers. The introduction rules correspond to playing the quantifier game as the prover, who is given an object for which they have to continue proving the quantified property. The elimination rules correspond to playing the quantifier game as the refuter, so we have to give the prover an object to get a proof that this object satisfies the quantified property. We now formally state these rules, as well as the rules for the existential quantifier.

| | | |
|---|---|---|
| | $\forall x : \varphi$ | ForallIntro $y$ |
| $y$ : `Object` | $\varphi[y/x]$ | |

As mentioned, we introduce a universally quantified variable. This variable is the object given to us (playing as the prover) by the refuter in the quantifier game. Since this can be any object, all we can do is give it a name, since we do not know anything else about it. The name we give to the object can be different from the name bound by the quantifier. In that case, we have to change the new goal such that it refers to the name of the object given to us, which is why there is a renaming $[y/x]$. Of course, if we use the same name, the renaming is trivial.

To re-iterate, what this rule does is allow *introducing* a new object. We know nothing about this object (except that it is from the universe). But now that we have this object in our context, we can use it (to e.g. construct new objects). For example, if we use this rule to introduce $x$, we can work with $x$, and also "derived terms" like $x + 1$ (assuming that our universe is that of numbers).

**Going Beyond:** Game Semantics

Previously, the quantifier game was defined only on formulas in prenex normal form, i.e. on formulas where all quantifiers are at the beginning.
The proof system can be thought of as an extension of the quantifier game "strategy tables" to arbitrary formulas. For this, it becomes necessary that prover and refuter switch role, since implication forces the refuter to admit a formula, which the prover can then try to refute.
When thinking about proofs, it can be very helpful to think about these as some kind of game against mathematics / yourself.
Such games can also be used to give a precise semantics to first-order formulas, like Lorentzian dialogues.

$$\begin{array}{c|c|l} \forall x : \varphi & \chi & \textsc{ForallElim}\ t \\ \hline \varphi[t/x] & \chi & \end{array}$$

This rule allows us to use a universally quantified formula. The way we use such a formula is rather straightforward: since we know that it is valid “for all objects”, we can put in a specific object to get that the formula holds for that object. This is also what happens in the quantifier game: since we are using an assumption, we play as the refuter, and can give an arbitrary object to the prover.

Note that in first-order logic, we describe objects using terms $t$, where $t$ must be a well-formed term. Well-formed terms are terms that can use the function symbols from our signature, as well as the variables we have added to our context as of now (e.g. by using ForallIntro in the rule before). When we put this term $t$, describing an object, into the universally quantified formula, we get back a proof that the property holds for this object.

As an example, let’s say that we have $\forall x : P(x)$ in our context. We can now use this rule on this assumption to get $P(0)$ (by instantiating with 0). We can then use the same rule again, on *the same assumption*, to get $P(1)$, and again to get $P(2)$. (Remember that assumptions can be used multiple times!) This allows us to get the enclosed property for every object we can imagine (and even for several). This is precisely what it means for something to be true for *all* objects.

$$\begin{array}{c|c|l} & \exists x : \varphi & \textsc{ExistsIntro}\ t \\ \hline & \varphi[t/x] & \end{array}$$

To prove an existentially quantified formula, we must provide a **witness** for which the quantified property holds. This witness is a well-formed term $t$, and the object this describes is the object given to the refuter by the prover. Since when proving a formula, we are the prover, we have to provide this object, which corresponds to us choosing an appropriate term $t$. So, if we need to prove $\exists x : P(x)$, and we already know that $P(42)$ holds (because e.g. we already have $P(42)$ in our context), then we can use this rule with 42 so that all that remains to be shown is indeed $P(42)$.

$$\begin{array}{l|c|l} \exists x : \varphi & \chi & \textsc{ExistsElim}\ y \\ \hline y : \texttt{Object} & & \\ \varphi[y/x] & \chi & \end{array}$$

In some ways, eliminating an existential quantifier is similar to what happens when introducing a universal quantifier with ForallIntro. In both cases, we get a new object. But when using ExistsElim, we additionally gain a fact about this object: This object satisfies the property described by the formula we just eliminated. So if we had $\exists x : P(x)$, then we know that this new object satisfies the property $P$. Again, we are able to change the name from the one bound by the quantifier to a name chosen by us.

We previously mentioned a well-formed term. A formula or term is **well-formed** only if all the first-order variables used in it are bound. We already know that quantifiers can bind variables. However, we can now have formulas in our context that refer to objects we previously added to that context. Therefore, the context also binds variables, which can then be used later. The rule here is that a variable can be used only *after* it has been added to the context.

Also, when adding a variable to the context, it may not shadow another variable already existing in the context. Thus, we might need to appropriately rename variables when using ForallIntro or ExistsElim.

Finally, note that all rules above have an extra "argument." For ForallIntro and ExistsElim, this is the name of the variable referring to the given object. For the other rules, it is the term describing the object given to the other player of the game.

Let's do some example proofs. We start with one of the quantifier reordering laws.

**Lemma 3.21.** *For any formula $\varphi(x, y)$, we have $(\forall x : \forall y : \varphi(x, y)) \leftrightarrow (\forall y : \forall x : \varphi(x, y))$*

*Proof.* See Figure 3.22. □

| Context | Goal | Rule |
|---|---|---|
| | $(\forall x : \forall y : \varphi(x, y)) \leftrightarrow (\forall y : \forall x : \varphi(x, y))$ | AndIntro |

| Context | Goal | Rule |
|---|---|---|
| | $(\forall x : \forall y : \varphi(x, y)) \to \forall y : \forall x : \varphi(x, y)$ | ImplIntro |
| H1 : $\forall x : \forall y : \varphi(x, y)$ | $\forall y : \forall x : \varphi(x, y)$ | ForallIntro $y$ |
| $y$ : Object | $\forall x : \varphi(x, y)$ | ForallIntro $x$ |
| $x$ : Object | $\varphi(x, y)$ | ForallElim H1 $x$ |
| H2 : $\forall y : \varphi(x, y)$ | $\varphi(x, y)$ | ForallElim H2 $y$ |
| H3 : $\varphi(x, y)$ | $\varphi(x, y)$ | Assumption H3 |

| Context | Goal | Rule |
|---|---|---|
| | $(\forall y : \forall x : \varphi(x, y)) \to \forall x : \forall y : \varphi(x, y)$ | ImplIntro |
| H1 : $\forall y : \forall x : \varphi(x, y)$ | $\forall x : \forall y : \varphi(x, y)$ | ForallIntro $x$ |
| $x$ : Object | $\forall y : \varphi(x, y)$ | ForallIntro $y$ |
| $y$ : Object | $\varphi(x, y)$ | ForallElim H1 $y$ |
| H2 : $\forall x : \varphi(x, y)$ | $\varphi(x, y)$ | ForallElim H2 $x$ |
| H3 : $\varphi(x, y)$ | $\varphi(x, y)$ | Assumption H3 |

Figure 3.22: The proof of Lemma 3.21

Our proof proceeds by showing both directions separately. Both proofs are almost identical. The proof makes explicit the intuitive reasoning explaining why we can permute the quantifiers here: We can reason that playing the quantifier game in both is equivalent since it does not matter which one needs to be given first, as both are given "at the same time." To make this argument formal, we show any game played on one formula can be turned into one played with the other formula. We do this by first receiving both objects, and then using them on the other formula in the opposite order. This is precisely what this proof does.

**Checkpoint 3.23:** Non-empty universes

When we defined universes, we required that they are not empty, so that there always is at least one object. Thus, the formula $(\forall x : \varphi(x)) \to \exists x : \varphi(x)$ is true, since we can always find a witness, because otherwise the universe would be empty.
If you try to prove this in our proof system, you will see that is impossible. This is because the proof system does not actually internalize the rule that the universe is not empty. Thus, if we were to change the definition of universes to also allow the empty universe, the proof system still works.
The following rule internalizes that the universe is not empty:

$$\begin{array}{r|c|l} & \varphi & \textsc{UniverseNotEmpty}\ x \\ \hline x\ \ : \texttt{Object} & \varphi & \end{array}$$

How does this rule work? Can you now prove the above formula?

### 3.2.2 Rules for Equality

So far, we can only use our objects to put them back into assumptions. Notably, we do not yet support reasoning about equality. We fix this now.

We have previously said that equality $a = b$ means that $a$ and $b$ describe the *same* object. This definition is not really useful, since it does not tell us what we can *do* with a proof that such objects are the same, or how we can construct one. We focus on how we can prove that two objects are the same first.

$$\begin{array}{r|c|l} & t = t & \textsc{EqualsIntro} \\ \hline\hline \end{array}$$

This rule tells us that the object described by a term $t$ (which must be well-formed) is the same object as the one described by $t$, which makes sense because $t$ can not describe more than one object. It also is the only direct way to prove equality. The principle behind this rule is that any object is equal to itself.[6]

However, since equality is very hard to prove, it is a very powerful connective. In particular, we have the following elimination rule.

$$\begin{array}{r|c|l} t_1 = t_2 & \chi & \textsc{EqualsElim} \\ \hline & \chi' & \end{array}$$

where $\chi'$ is $\chi$, but some occurrences of $t_1$ are replaced by $t_2$.

This rule describes the substitutive property of equality: Whenever $t_1$ and $t_2$ are equal, we can replace one with the other. Somewhat remarkably, this works within arbitrary sub-expressions.

Let's use this to show that equality is symmetric.

**Lemma 3.24** (Symmetry of Equality)**.** $\forall x, y : x = y \to y = x$

[6] This is called *reflexivity* and shows up again in Section 4.2.

*Proof.* See Figure 3.25. □

| Context | Goal | Rule |
|---|---|---|
| | $\forall x, y : x = y \to y = x$ | FORALLINTRO $x$ |
| $x$ : `Object` | $\forall y : x = y \to y = x$ | FORALLINTRO $y$ |
| $y$ : `Object` | $x = y \to y = x$ | IMPLINTRO |
| `H1` : $x = y$ | $y = x$ | EQUALSELIM `H1` |
| | $y = y$ | EQUALSINTRO |

Figure 3.25: The proof of Lemma 3.24

In the proof, we can replace $x$ with $y$ in the goal $y = x$ since $x = y$ is assumed. We also call this **rewriting** with $x = y$. Note that we can not replace $y$ with $x$, since our EQUALSELIM rule is written so that the left side ($x$) is replaced by the right side.

As we have seen, this is a silly restriction. We can thus add the following derived rule, which works backwards.

| $t_1 = t_2$ | $\chi$ | EQUALSELIM $\leftarrow$ |
|---|---|---|
| | $\chi'$ | |

where $\chi'$ is $\chi$, but with some occurrences of $t_2$ replaced by $t_1$.

Similarly, we can show that equality is transitive.

**Lemma 3.26** (Transitivity of Equality)**.** $\forall x, y, z : x = y \to y = z \to x = z$

*Proof.* See Figure 3.27. □

| Context | Goal | Rule |
|---|---|---|
| | $\forall x, y, z : x = y \to y = z \to x = z$ | FORALLINTRO $x$ |
| $x$ : `Object` | $\forall y, z : x = y \to y = z \to x = z$ | FORALLINTRO $y$ |
| $y$ : `Object` | $\forall z : x = y \to y = z \to x = z$ | FORALLINTRO $z$ |
| $z$ : `Object` | $x = y \to y = z \to x = z$ | IMPLINTRO |
| `H1` : $x = y$ | $y = z \to x = z$ | IMPLINTRO |
| `H2` : $y = z$ | $x = z$ | EQUALSELIM `H1` |
| | $y = z$ | EQUALSELIM $\leftarrow$ `H2` |
| | $y = y$ | EQUALSINTRO |

Figure 3.27: The proof of Lemma 3.26

That proof uses both variants of the EQUALSELIM rule. First, $x$ is replaced by $y$, as $x = y$. Then $z$ is replaced by $y$, as $y = z$, by reading the equality the other way around.

**Checkpoint 3.28**

There is a proof of Lemma 3.26 that only uses EQUALSELIM once. Can you find it?

Here are two more rules we will need:

| | | |
|---|---|---|
| | $\chi$ | Axiom |
| $\varphi$ | $\chi$ | |

$\varphi$ is an axiom of our theory

| | | |
|---|---|---|
| | $\chi$ | Lemma |
| $\varphi$ | $\chi$ | |

$\varphi$ was proven in a separate lemma

The second rule, Lemma, is merely so that we can structure our proofs and make them easier to understand. We could always replace it by an application of Assert, and then re-roll the proof of the original lemma. Since the resulting proofs would be both large and redundant, we instead introduce this rule. When using the Lemma rule, one should always state which lemma is used. Typically, this is done by naming or numbering all lemmas, like for instance in this book.

The Axiom rule allows us to define **provability modulo a theory** $\mathcal{M}$. With it, we are able to use the axioms of our theory. Thus, when proving a formula using that rule, we are no longer showing that this formula is provable in general, but only that it is provable modulo some theory.

We now do some examples in the theory of natural numbers, as introduced in appendix A. This means that we can use all the laws listed in that appendix.

**Lemma 3.29** ($\leq$ is reflexive)**.** $\forall x : x \leq x$

*Proof.* See Figure 3.30. □

| Context | Goal | Rule |
|---|---|---|
| | $\forall x : x \leq x$ | ForallIntro $x$ |
| $x$ : `Object` | $x \leq x$ | ExistsIntro 0 |
| | $x = x + 0$ | Axiom |
| A1 : $\forall n : n + 0 = n$ | $x = x + 0$ | ForallElim A1 $x$ |
| H2 : $x + 0 = x$ | $x = x + 0$ | EqualsElim H2 |
| | $x = x$ | EqualsIntro |

Figure 3.30: The proof of Lemma 3.29

You might be confused how we can apply ExistsIntro to a goal of shape $x \leq x$. The answer is that $x \leq y$ is defined (Definition A.7) as $\exists k : y = x + k$, and thus the rule is applicable. This can be confusing, as we usually do not remember all definitions when reading proofs. Thus, for readability, we sometimes want to explicitly unfold definitions. We denote this by Defn. Using this, the proof becomes much more readable, as Figure 3.31 shows.

Note that Defn is not a rule. It does not change the proof state. Instead, it allows us to transform a goal by adding or removing syntactic sugar, or by folding or unfolding definitions. For further readability, we should state the used definitions, as is done in Figure 3.31. We can also use it to change a hypothesis. To do so, we simply re-state the hypothesis, keeping the same name.

Let's look again at the formal argument made by that proof. We argue that every number is less than or equal to itself. This sounds tautological, but it is not: We have a very precise definition of $\leq$, and it is not immediately obvious why this definition has the desired property $x \leq x$. The definition works by saying that $x \leq y$ if we can add some $k$ to $x$ to reach $y$. The reason it is reflexive is that we can always add 0 and because adding 0 has no effect.

| Context | Goal | Rule |
|---|---|---|
| | $\forall x : x \le x$ | FORALLINTRO $x$ |
| $x$ : `Object` | $x \le x$ | DEFN $\le$ |
| | $\exists k : x = x + k$ | EXISTSINTRO 0 |
| | $x = x + 0$ | AXIOM |
| A1 : $\forall n : n + 0 = n$ | $x = x + 0$ | FORALLELIM A1 $x$ |
| H2 : $x + 0 = x$ | $x = x + 0$ | EQUALSELIM H2 |
| | $x = x$ | EQUALSINTRO |

Figure 3.31: The proof of Lemma 3.29, using DEFN

The statement "Adding 0 has no effect" is an axiom of natural numbers. It is something we take for granted or have otherwise justified as true, like by appeal to intuition. Alternatively, we can see it as (part of) a definition of 0 and of addition—two mathematical concepts behaving such that adding 0 has no effect.

For another example, here is a proof that adding two even numbers yields another even number. $even(n)$ is defined in Definition A.10.

**Lemma 3.32.** $\forall x, y : even(x) \longrightarrow even(y) \longrightarrow even(x + y)$

*Proof.* See Figure 3.33. □

**Checkpoint 3.34**

What is the intuitive reasoning behind the proof in Figure 3.33?

## 3.3 Proof Tables in Practice

When looking at the proofs we have derived so far, we can see that we are often doing one of two things:

Our proofs usually start by using IMPLINTRO or FORALLINTRO. Those directly move assumptions or objects into our context. Often, we then additionally use ANDELIM or EXISTSELIM on those, to get more elementary assumptions. In general, we call this use of taking assumptions and organizing them into their constituent parts **introducing**.

Then, we have some lemma as part of our assumptions (or maybe by using AXIOM), and we **apply** this lemma. Often, before we can use IMPLAPPLY, we must provide specific objects using FORALLELIM, and show some preconditions (using ASSERT and IMPLSPECIALIZE). Also, we might not be able to show the goal, but instead only get another new assumption, which we can then again break apart using ANDELIM or EXISTSELIM.

Introducing and applying lemmas is often *mechanical*. By this we mean that it does not really involve any deeper thought, requiring only the application of rules.

What is not captured by these two activities is usually the more interesting part of the proof. For example, in order to apply a lemma, we need to pick it first, and usually we need to choose at least

| Context | Goal | Rule |
|---|---|---|
| | $\forall x, y : even(x) \rightarrow even(y) \rightarrow even(x+y)$ | ForallIntro $x$ |
| $x$ : `Object` | $\forall y : even(x) \rightarrow even(y) \rightarrow even(x+y)$ | ForallIntro $y$ |
| $y$ : `Object` | $even(x) \rightarrow even(y) \rightarrow even(x+y)$ | ImplIntro |
| `H1` : $even(x)$ | $even(y) \rightarrow even(x+y)$ | ImplIntro |
| `H2` : $even(y)$ | $even(x+y)$ | Defn *even*, \| |
| `H1` : $\exists k : 2k = x$<br>`H2` : $\exists k : 2k = y$ | $\exists k : 2k = x + y$ | ExistsElim `H1` $k_1$ |
| $k_1$ : `Object`<br>`H3` : $2k_1 = x$ | $\exists k : 2k = x + y$ | ExistsElim `H2` $k_2$ |
| $k_2$ : `Object`<br>`H4` : $2k_2 = y$ | $\exists k : 2k = x + y$ | ExistsIntro $k_1 + k_2$ |
| | $2(k_1 + k_2) = x + y$ | Axiom |
| `ADist1` : $\forall a, b, c : a(b+c) = ab + ac$ | $2(k_1 + k_2) = x + y$ | ForallElim `A5` 2 |
| `ADist2` : $\forall b, c : 2(b+c) = 2b + 2c$ | $2(k_1 + k_2) = x + y$ | ForallElim `H6` $k_1$ |
| `ADist3` : $\forall c : 2(k_1 + c) = 2k_1 + 2c$ | $2(k_1 + k_2) = x + y$ | ForallElim `H7` $k_2$ |
| `ADist3` : $2(k_1 + k_2) = 2k_1 + 2k_2$ | $2(k_1 + k_2) = x + y$ | EqualsElim `H8` |
| | $2k_1 + 2k_2 = x + y$ | EqualsElim $\leftarrow$ `H3` |
| | $2k_1 + 2k_2 = 2k_1 + y$ | EqualsElim $\leftarrow$ `H4` |
| | $2k_1 + 2k_2 = 2k_1 + 2k_2$ | EqualsIntro |

Figure 3.33: The proof of Lemma 3.32

some of the variables needed for this lemma. Alternatively, performing a case distinction (using ExcludedMiddle or OrElim) is also usually more involved. Similarly, when choosing a witness for an existential quantifier (ExistsIntro), we also need to be clever, since it has to be correct.

When trying to write a proof, or when reading it, it is often useful to explicitly write down the proof state to check that no lemmas are forgotten or used in an invalid way. Proof tables could be a great way to do this. However, our current proof tables were developed to show how mathematical proofs work in detail. Now that we know these details, they are much too large to be useful.

In this chapter, we make our proof tables much more informal, while also making them easier to work with. For this, we add several pseudo-rules. These "rules" rely on the reader to fill in the details.

Our proof that addition preserves evenness can then be shortened, as Figure 3.35 shows.

We have used an Intro pseudo-rule to handle introducing all the theorems. Similarly, after introducing our axiom `A3`, we directly eliminate it, and then directly specify what objects we use to eliminate our rules.

Another proof, where we liberally use a pseudo-rule we call Apply, can be found in Figure 3.36

We can see that there are two new subgoals generated when applying `H1`. Usually, we would have to Assert the first subgoal, to later use ImplSpecialize to discharge it from our assumption we

| Context | Goal | Rule |
|---|---|---|
| | $\forall x, y : \quad even(x) \longrightarrow even(y)$<br>$\longrightarrow even(x+y)$ | Intro, Defn |
| $x, y, k_1, k_2$ : `Object`<br>`H1` : $2k_1 = x$<br>`H2` : $2k_2 = y$ | $\exists k : 2k = x + y$ | ExistsIntro $k_1 + k_2$ |
| | $2(k_1 + k_2) = x + y$ | Axiom |
| `ADist` : $\forall a, b, c : a(b+c) = ab + ac$ | $2(k_1 + k_2) = x + y$ | EqualsElim `A1` with $2, k_1, k_2$ |
| | $2k_1 + 2k_2 = x + y$ | EqualsElim $\longleftarrow$ `H1`, `H2` |
| | $2k_1 + 2k_2 = 2k_1 + 2k_2$ | EqualsIntro |

Figure 3.35: A shorter proof of Lemma 3.32

| Context | Goal | Rule |
|---|---|---|
| | $(\forall a, b, c : P(a,b) \longrightarrow P(c,b)$<br>$\longrightarrow P(a,c))$<br>$\longrightarrow \forall a, b : P(a,b) \longrightarrow P(a,a)$ | Intro |
| `H1` : $\forall a, b, c : P(a,b) \longrightarrow P(c,b)$<br>$\longrightarrow P(a,c)$<br>$a, b$ : `Object`<br>`H2` : $P(a,b)$ | $P(a,a)$ | Apply `H1` with $a, b, a$ |

$\swarrow$ $\searrow$

| Context | Goal | Rule |
|---|---|---|
| | $P(a,b)$ | Assumption `H2` |

| Context | Goal | Rule |
|---|---|---|
| | $P(a,b)$ | Assumption `H2` |

Figure 3.36: A proof using the Apply pseudo rule

wanted to apply. However, since doing this does not help us understand the proof better, we now start to gloss over the details when using proof tables to get a rough idea.

**Checkpoint 3.37**

Construct a formal deduction for the proof sketched in Figure 3.36. Only use the core rules of our calculus, not any derived rules.

In general, the idea of a proof table is to make several things precise:

- What are our current assumptions?
- Which lemma/assumption/axiom are we applying, and what remains thereafter?
- Which objects do we use to instantiate the assumption being applied?
- Which objects do we use for existence proofs?
- Where do case distinctions happen? On what?
- Which side of a disjunction do we choose?

When doing so, we should always make sure that we know how to unpack an informal rule like Apply. Luckily, that level of detail is often unnecessary in practice.

**In Other Words**

Proof tables are often unnecessarily long. We can make them simpler by coming up with more powerful rules, that can do many things at once, and ease certain common operations. Unfortunately, these rules are kind of made up and require certain care. It can be possible to misuse them, and so, to construct an invalid proof which nonetheless looks correct. Thus, when using them, we should at least shortly think about how to unpack them into smaller steps closer to the actual basic rules.
This also means that these rules can *not* be used when we ask for a formal deduction in an exam.

## 3.4 Textual Proofs

Proof tables can be nice for visualizing proofs, and for finding them. Unfortunately, actual proofs are almost always written in plain language, using a particular style. Note that this chapter presents a particular style, while in reality, proofs might be considerably more diverse.

At its core, any textual proof can be read as a manual for constructing a proof table. That is, all the concepts we know from proof tables still apply. In a textual proof, we still progress from goal to goal. We still need to keep track of our assumptions, and the objects we can use to construct new objects. We are still bound by the rules of our logic, and can only do certain operations.

However, since we are no longer bound by the strict formalism of a proof table, we can express this in a much more natural way. Before we can start writing such proofs on our own, we should look at a few examples.

First, let's again look at the fact that addition preserves even-ness.

**Lemma.** $\forall x, y : \mathit{even}(x) \longrightarrow \mathit{even}(y) \longrightarrow \mathit{even}(x + y)$

*Proof.* Let $x, y$ be even numbers. By definition, there then are $k_1, k_2$ such that $2k_1 = x$ and $2k_2 = y$. It remains to be shown that $\exists k : 2k = x + y$. We choose $k := k_1 + k_2$, and conclude, as $2(k_1 + k_2) = 2k_1 + 2k_2 = x + y$ by elementary arithmetic. □

This proof is almost a verbal reading of the informal proof table from a few pages ago. Indeed, if you design your proof tables well, you can usually create a formal proof by just reading out. Conversely, a nicely written textual proof immediately suggests how to construct a proof table.

We can already state the most important difference: In a textual proof, it is uncustomary to give names to assumptions. Instead, we simply state the assumptions and then state them again when later using them. This can be more cumbersome and harder to track than a proof table, and a skilled proof writer will know how to construct a textual proof so that this can be avoided. Unfortunately, since this is not common, there is no universal way to name assumptions. Simply adding a name, separated by a colon, like in "and assume H1 : $x = y$," usually gets the point across. Another difference is that we verbalized some of the formalism. Instead of writing "Let $x, y$ be given such that $\mathit{even}(x) \ldots$," we wrote that $x, y$ are even numbers, which is much nicer to read.

Here is another example, which just uses propositional logic.

*Proof that* $P \lor Q \to Q \lor P$. We assume $P \lor Q$. Case distinction on this:

- $P$ holds. We are done by choosing the right side.
- $Q$ holds. We are done by choosing the left side. □

As a general rule, we can always create a bullet point list. If we get further sub-tasks, we can always create more nested bullet points. While it depends on your style, you might want to avoid more than three levels of indentation. Instead, use other methods of structuring the proof, which we outline soon. For now, we focus on how we conventionally phrase the several proof rules we have seen so far.

Before we get into the details of writing a proof, we should note where writing such a proof is appropriate. In mathematical writing, proofs should always start with "*Proof*," in italic. Afterwards, one can state the main idea of the proof, like "by induction" or "by contradiction." Of course, if the proof is not by some "special" method, it is fine to simply write *Proof.*

At the end of a proof, a square □, called **tombstone**, is common. It is put at the right side of the page, right after the last part of the proof. In the past, people used to put certain important-sounding phrases like "quot erat demonstrandum."[7] instead. You can also end your proof with "qed.", but the tombstone is preferred nowadays.

Further note that when we start a proof, it should always be completely clear which goal we are actually trying to prove right now. Usually, this is accomplished by proving lemmas immediately after they are stated. If not, the goal to be proven should be repeated. It should not be described informally, like by saying "Addition preserves even-ness," instead it should be spelled out formally. Once this is done, the proof can begin properly.

For this, there is one rule that always holds: *Know the reader.* The main reason to write a proof is so that people, including the author[8], can read it. Thus, one should not clutter the proof, to make it unnecessarily long. The reader certainly knows the basics of first-order logic, and need not be told that a conjunction is true iff both sides are true. Conversely, they might have a bad short-term memory and must be reminded of certain definitions, lemmas, and assumptions. After writing a proof, one should read it again, and rephrase the parts where one got lost.

### 3.4.1 How to Translate a Proof Table

We now give rules for writing a proof that are based on a proof table. The idea is that a proof table already closely captures what happens during a proof, and since we already know how to construct these, writing a proof with an already existing proof table in mind becomes a lot easier. Eventually, you will be able to construct the proof table in your head while writing the proof. Even then, drawing the proof table (even if just the shortened version) on paper is beneficial.

[7] Latin for "which was to be proven."

[8] Writing down a proof can be useful way to gain more confidence that this proof is correct, since it forces you to write out why each step is justified.

**Concluding the Proof** The rules EqualsIntro, FalsityElim, and TruthIntro all terminate the goal. In a proof, we usually indicate that a proof is over using the word "done," or by saying "conclude." For example, if the last non-trivial operation is to apply a lemma, we might then conclude our proof like this:

- We apply the lemma and are done.
- Conclude by applying the lemma.
- Finally, apply the lemma.

The Assumption rule can similarly be used to conclude a proof. Here, we have to be a bit more careful. If the assumption has recently been introduced and is still present in the short-term memory of the person reading our proof, we can simply *be done* like with the rules above. Otherwise, we should hint that our current goal is already assumed, by just mentioning that we conclude by assumption. This might be phrased like this:

- Apply the lemma and conclude by assumption.
- We are done since the goal was previously assumed.
- Conclude using `H3`.
- We are done since this was already shown using Lemma 42.
- Finish since this was assumed initially.

When we named an assumption, we should consistently use that name to refer to it. As mentioned, this is uncommon. Instead, we can also try to describe where we got this assumption. If we had it since the beginning, we can state this. If we applied some special lemma and kept the stated assumptions around until now, we can also reference this lemma. Creative proof writers might also refer to the reader using phrases like "Which was assumed two pages ago." If nothing is applicable, the assumption can also be stated in full, although this stops the reader, since they are now searching for where this assumption was introduced.

Again, you can simply *be done* if the assumption is fresh. When in doubt whether an assumption is fresh, it is helpful to remain on the side of caution and state it explicitly.

**Introducing New Assumptions** In our informal proofs, we have already seen the shorthand Intro, which combined the rules ForallIntro, ImplIntro, AndElim, and ExistsElim. In a textual proof, we use the same approach: We break apart all our assumptions as soon as we introduce them. Then, it is usually sufficient to merely state the new, decomposed assumptions and variables.

You can expect that your readers know how to break apart an assumption $P \wedge Q$, or that one assumes the precondition of an implication to prove the consequences. Use phrases like

- We are given $a, b, c$ such that $a = b + 1$ and[9] $b = c + 1$.

[9]In an actual proof, introducing an $a = b + 1$ would be pretty useless. We use this assumption here for didactical reasons.

- Assume $a, b, c$ with $a = b + 1$ and $b = c+1$.
- We have $a, b, c$ for which $a = b$ and $b = c$.
- Let $a, b, c$ be arbitrary, but fixed.
- Let $a, b$ be arbitrary and $c$ such that $a < b < c$.

The phrase "arbitrary, but fixed" is usually used when introducing a variable that is not further constrained by a property.

Phrases like the following should be *avoided* since they only clutter the proof:

- We have $a, b, c$ and $a = b + 1 \land b = c + 1$. Since both are true, we can deduce $a = b$ and $b = c$ separately.
- We can assume that $a = b + 1$ is true since if it is false, we are done since an implication is true when the precondition is false.

The first phrase can sometimes be used, but only if we need to break apart a "spurious" conjunction that we did not have the chance to break apart otherwise. Even then, avoid phrases like "since both sides are true"—both you and your audience already know how conjunction works, and do not need to be told this.

**Applying Lemmas and Hypotheses** In our proof, we will almost certainly use our assumptions or other lemmas or axioms we have already proven. The first and most important rule is that we should clearly state which lemma, axiom, or assumption we are using. We have already seen how we reference assumptions. For lemmas, we usually give them either names or consistently number them, in which case we can simply refer to "lemma 42." Similarly, axioms might have names, like "Distributivity of +," which make them easy to reference in a textual proof. Of course, some variation is allowed here—"since + is distributive" is completely fine. If it is not clear which axiom is meant, you should make this explicit. When in doubt, formally state the complete axiom.

When we apply a lemma (axiom, hypothesis), this usually has a similar structure: First, we specialize any quantifiers, then we need to handle several preconditions. Eventually, we get to the "meat" of the lemma—some new, useful assumptions which help us continue our proof. These new assumptions are then handled using the already discussed techniques for introducing new assumptions. As discussed above, we simply state the assumptions we get after introduction.

One might think that the reader knows the lemma, and thus the assumptions we get need not be re-stated. In practice, it is extremely helpful for the reader to note the assumptions the lemma introduced. This can alleviate confusion, especially since it hints to the reader what the lemma was about, just in case they already forgot.

Applying a lemma can be phrased like this:

- We apply lemma 42 to get $k$ and $r$ such that $a = k^2 + r$.
- Using lemma 42 with $c$, we get that $a < c \lor a > c$.
- We apply lemma 42. For this, we first show that $a = b$ and $b = c$ actually hold, and can then derive … *[followed by two cases, one for $a = b$ and one for $b = c$]*

- We continue with lemma 42, which is applicable since $k^2 > 0$, and find …
- We use lemma 42. Now, $x = y$ and $y = z$ additionally need to be shown.

The last example highlights how to mention that there are new goals. This is usually expressed by saying that something "needs to be shown." If the original goal disappears because it could be concluded, the phrase "remains to be shown" should be used instead, which indicates that we have concluded this goal by introducing others. We again note that new goals should always be stated formally so that the reader can orient themselves if they were unsure about the specific proof obligations imposed by some lemma.

When applying a lemma, we often do not do it in the fully general fashion, where we simply introduce all the new assumptions into our context. Instead, it often is the case that there is only one such assumption. Then, we can immediately use it, instead of first introducing it. We will see several examples of this in the next few examples.

**Working with Equality** The main way to prove equality is by noting that both sides of the equality sign are syntactically the same. We have already seen that this is so obvious that we can simply state "We are done" and successfully conclude the proof.

We still need to discuss how to use an equality, i.e. how to describe uses of the EqualsElim rule. Since this is called rewriting, we express this by saying that we *rewrite* using an equality. If we want to be brief, we can also just mention that the next step uses the equality since it should be obvious that it is used for rewriting. This is commonly expressed like this:

- Using $x = y$, showing $y = z$ suffices.
- We rewrite with $a = b$ to get $b > B$.
- Conclude with $x = 2k_1$.

Again, since the goal changes, it should be re-stated.

So far, rewriting was limited to the goal, even though rewriting in an assumption is totally fine. In a proof table, this would work by first asserting the changed assumption, and then proving the new subgoal generated by Assert using ordinary rewriting. In a textual proof, this is easier. We simply state that we rewrite in some assumption, like this:

- Since $a = b$ we additionally have $b > c$.
- We rewrite with $q = r^2$ in the assumption $a = \sqrt{q}$ to get $a = \sqrt{r^2}$.

When repeatedly rewriting with several equalities, it is often useful to give an equality chain. This can be done inline, like this:

- We conclude, since $a = b = c > d = e$ (Provided the reader knows that $a = b$, $b = c$, $c > d$, and $d = e$ are assumptions).

- We show this by rewriting repeatedly:

$$\begin{aligned} a &= b && \text{As } a = b \\ &= c && \text{As } b = c \\ &> d && \text{As } c > d \\ &= e && \text{As } d = e \end{aligned}$$

This style of reasoning is likely familiar. You now know that this is just repeated elimination of the several equalities.

Finally, we can also combine rewriting with lemma application. This is much more simple than it sounds. For example, imagine that there is some lemma (named lemma 42) stating $\forall a, b : a \geq b \rightarrow a \leq b \rightarrow a = b$. Then, one can combine rewriting and application like this:

- We rewrite using lemma 42 to get $k = b$, while $a \geq b$ and $a \leq b$ also need to be shown. [if the goal before was $k = a$]
- .. so $k + x = k + y$ still needs to be shown. After rewriting lemma 42 with $x, y$, just $x \geq y$ and $x \leq y$ remain to be shown.
- We use lemma 42 in our assumption that $x^2 + y^2 = z^2$. We next show $x \leq y$ and $x \geq y$ and then thus get $y^2 + y^2 = z^2$.

We remark that in a textual proof, it is often natural to combine several closely coupled steps into just one operation, like we did with the last example from above. There, we applied a lemma and rewrote in an assumption all at the same time.

**Organizing Cases** So far, we have seen several examples which introduce new proof goals. In our proof tables, we would create new sub-tables.[10] In a textual proof, we can use a list and simply prove the several goals one after the other. By appropriate indentation, we can communicate which subgoals start where, if there are nested subgoals.

If the new goals are very simple, we can also solve them inline, without explicitly creating a new list. For example, one might write this:

- We apply lemma 42 and note that $x \geq y$ and $x \leq y$ are assumed.
- $a = b$, $b = c$, and $c = d$ need to be shown. The first two are straightforward, so $c = d$ remains.

Occasionally, a proof might have so many sub-goals that spacing becomes an issue. Having more than three nested sub-goals should be avoided, and a proof with that many subgoals can likely be refactored. Usually, one can find a helpful lemma which can be proven separately and eliminates the need to have that many nested sub-goals. In fact, it is always possible to extract the current proof state into a separate lemma, but a skilled proofwriter will identify what parts should be refactored into which lemmas so that the proof, and the new lemmas, are "pretty."

[10]In fact, proof tables merely optimize the case where the proof obligation remains. In general, proofs do not go linearly but have a tree shape. Usually, one then also uses inference rules for presenting formal proofs. We don't here, since they are bad for learning proofs.

Alternatively, it sometimes is possible to group different goals by using one paragraph for each goal, or by putting separating lines between different parts of the proof. There are no hard rules here, as long as it is obvious where one goal ends and the next one starts.

Additionally, instead of using plain bullets • to separate different proof goals, we can use more helpful symbols. A common case is when proving a logical equivalence, like $x = y \leftrightarrow y = x$. There, one usually does the following:

*Proof that* $x = y \leftrightarrow y = x$. We show both directions separately:

$\rightarrow$: We assume $x = y \ldots$

$\leftarrow$: We assume $y = x \ldots$ □

This is particularly useful if there are several similar-looking proof goals that only differ in a few symbols. Furthermore, if there are so many proof goals that the average reader will already have forgotten goal number 4 after proving the first three goals, one should again remind the reader what the specific goal is. If a special bullet (like above) does not suffice, it is helpful to recall the entire sub-goal.

**Case Distinctions** In our proof tables, there are two different rules that are usually called **case distinctions**: ExcludedMiddle and OrElim.

The difference is that ExcludedMiddle does a case distinction based on whether a statement is true or false. For OrElim, the case distinction is between two sides of a disjunction. In either way, we typically start a case distinction like this, and then follow it by proving the different cases:

- Case distinction on whether $a = b$ is true:
- Case distinction on $P \lor Q$:
- We are given $x$, and that $x = 3$ or $x = 4$. Case distinction:
- We apply lemma 42 and continue, depending on whether $P$ or $Q$ holds: *[where lemma 42 shows that* $P \lor Q$*]*

Again, a case distinction can be combined with the application of a lemma.

There are several more special cases of case distinctions. A common one is checking whether some objects are the same or not:

- We do a case distinction on $x$. If $x = 0$, the claim is obvious, otherwise …
- Case distinction on $x$:
  - $x = 0$ ...
  - $x = 1$ ...
  - otherwise, ...
- We analyze the relation between $x$ and $y$:

<: We know $x < y \ldots$

=: We know $x = y \ldots$

>: We know $x > y \ldots$

In fact, the last case was an implicit application of one of the properties of <. In that case, we trust that the reader of the proof can figure out which axiom was used, or is otherwise able to see that this case distinction is correct and does not miss any cases.

**Falsity and Contradictions** Remember the FalsityElim rule: It allowed us to conclude any proof if one of the assumptions is $\bot$. We also derived a ExFalso rule, which similarly allowed us to always change the current proof goal to falsity, i.e. to show that the assumptions in the context are contradictory in a more general way.

In a textual proof, this also works. We do not call it *eliminating falsity*, but *deriving a contradiction.* As a first approximation, a formal proof replaces the word *falsity* by the word *a contradiction.* We might say:

- We show that our assumptions are contradictory.
- Assume … and derive a contradiction.
- Since $a = b$ and $a < b$ are contradictory, we are done.
- We assume $P \lor \bot$. Case distinction. The second case is done, since it is contradictory. For the first case, …

Unfortunately, the word "contradiction" is also used to describe the proof technique of "proof by contradiction." In a proof by contradiction (remember the Contradiction rule), we prove a goal $P$ by assuming $\neg P$ and showing $\bot$, i.e. deriving a contradiction. Intuitively, this establishes that $\neg P$ can not be true, so $P$ must be.

**Trivial Statements** When reading proofs, one often comes across statements like "this is obvious" or "holds trivially." Sadly, not all statements claimed to be obvious actually are. Thus, this should be avoided, unless one knows that the audience will similarly be able to see that the goal is obvious.

For example, statements involving "trivial" operations on numbers, using just addition, multiplication, and constants, are usually better described as *holding by elementary arithmetic.* Similarly, it is usually obvious that a goal of shape $\neg(\varphi \land \phi)$ can be transformed to $\neg\varphi \lor \neg\phi$. Such transformations can then be said to hold *by elementary propositional logic.* Here, the word *trivially* is also fine, since the audience must already know propositional logic to even have a chance at understanding the proof. Even then, it helps to be more specific. If it is not immediately obvious how a trivial statement can be deduced from the algebraic laws of propositional logic, it should not be claimed to hold trivially.

**Some Examples** We now translate some of the examples from the previous chapter. We have already seen that addition preserves even-ness. We can also show that adding two odd numbers gives an even number again:

*Proof that* $\forall x, y : odd(x) \to odd(y) \to even(x + y)$. Let $x, y$ be odd numbers. Then there are there are $k_1$ and $k_2$ such that $2k_1 + 1 = x$ and $2k_2 + 1 = y$ by the definition of *odd*. It remains to show that there is a $k$ for which $2k = x + y$. We choose $k := k_1 + k_2 + 1$, and can then conclude $2(k_1 + k_2 + 1) = x + y + 2 = 2k_1 + 1 + 2k_2 + 1$ by elementary arithmetic. □

Here is the proof that $\leq$ is reflexive:

*Proof that* $\forall x : x \leq x$. Let $x$ be arbitrary, but fixed. We must find $k$ such that $x = x + k$. Choose $k := 0$ and be done, as $x + 0 = x$. □

Here is another example, which does not use any arithmetic, but just plain first-order logic:

**Lemma 3.38.** $(\forall x, y : P(x, y) \to Q(y, x)) \to (\forall x : P(x, 42)) \to \exists z : \forall k : Q(z, k)$

*Proof.* We assume $\forall x, y : P(x, y) \to Q(y, x)$ and $\forall x : P(x, 42)$. Choose $z := 42$, and let $k$ be arbitrary, but fixed, so that $Q(42, k)$ remains to be shown. Using the first assumption, $P(k, 42)$ suffices, which can be shown using the second assumption. □

**Final Remarks** As always, the usual writing advice applies:

- Be as short as possible, and as precise as is necessary.
- Know your reader.
- If you do not understand what you have written, you should rewrite it.

Apart from this, one is free to deviate from the rules laid out here, as long as it is clear how the corresponding proof table would look. The suggestions made in this chapter are just suggestions and no hard rules.

# 4 Sets and Relations

## 4.1 Sets

Throughout the history of mathematics, it has been the main goal of mathematicians to describe certain phenomena occurring in nature and, ultimately, explain everything that is to understand about our world and, quite literally, the universe. Now, with the knowledge gathered in the previous chapters, we are not yet in a position to do this because we lack certain tools. Until now, we have formulated predicates containing variables, that reason about properties of *objects*. And, what we can certainly say, is that the world around us is, undoubtedly, made up of *objects*. However, this sentence, despite being true, does not give you any useful information about either the world or the objects. Therefore, in this chapter, we want to explore those tools that we need to effectively describe what is going on around us. Mathematicians (and also computer and other scientists) call this process *modeling*. We employ mathematical infrastructure to describe certain phenomena in the real world. Thereby, we can actively focus on aspects that are especially important for us in a specific situation or context and we can ignore the remaining aspects.

The exact tool we are talking about is *set theory*. This is a general term that embraces many different notions and concepts that enable us to group and categorize objects and also to then mathematically reason about such collections of objects. We will also see that certain laws emerge that help us even more to make statements about objects and collections and to argue about them.

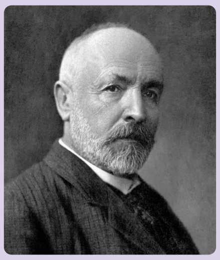
**Chapter Goals**

In this chapter, you will learn

- the foundation of mathematical set theory
- how to state and prove facts about sets and their elements
- about the concept of cardinality; first, for finite sets only

Although plenty of mathematicians have bothered with set theory we want to take a look at one of the earliest attempts to define collections of objects. The first definition was made by Georg Cantor and although the exact name of the term has changed over time (in modern mathematics we talk about *sets*), the description itself still provides a good starting point to get into the topic.

**Important Individual:** Georg Cantor



Georg Cantor (1845-1918) was a German mathematician and one of the founding fathers of modern set theory. His most known works discuss infinite sets, showing that infinitely many differently sized infinities exist, and the proof that the set of real numbers $\mathbb{R}$ is larger than the set of natural numbers $\mathbb{N}$. Before focussing on set theory, Cantor also researched in the fields of number theory and calculus with a special focus on representing functions using trigonometric series.

> **Definition 4.1** (Set [Naive]). *By an "aggregate" (**set**) we are to understand any collection into a whole $M$ of definite and separate objects $m$ of our intuition or our thought. These objects are called the "elements" of $M$.*

This sentence is rather vague, which you don't expect from a mathematically sound definition. Additionally, there might be a bit of confusion about the exact wording employed by Cantor versus "modern" formulation. So, before we do anything else, let's clarify what the term *set* does and does not mean. First of all, the word *set* does not occur in the original version of Cantor's definition, and neither does the word *aggregate*. This is because Cantor's works were in German. Therefore, to get a better grasp of the concept of a set, we need to look at the German word "Menge" that was used in the original German text. Most of the time, "Menge" is used to describe the quantity of objects where the exact amount is not known or relevant. For example, "eine Menge Äpfel" translates to "plenty of apples." However, this is not the intended meaning. As we have seen, the translation of Cantor's definition features the word *aggregate* which refers to the collection itself and not the number of objects in the collection. This is an important difference because the number of objects is just one of many properties of such an aggregate. Apparently, over time, the word *set* has replaced *aggregate* (for unknown reasons) but the meaning has not changed.

With that out of the way, in the following paragraphs, we will try to get an intuition for the basic rules sets must follow and some important properties they have.

You can imagine a set like a box. This box typically has a name consisting of a capital letter, which we already saw in Cantor's definition. You can put anything you like into the box. For describing its contents, we typically use curly braces and then simply list all the objects in the box: Let's say you have a box $B$ and put in a shirt, a pen, and a coin. In short, this would be

$$B := \{\text{shirt}, \text{pen}, \text{coin}\}.$$

Nothing unusual so far, but here comes the first caveat with this metaphor. When you put in another pen identical to the first one, strange things start to happen. Remember how the definition of a set mentioned that objects need to be *separate*? This means that any object can not appear *twice* in a set, it can only be *contained* or *not contained*. This has the consequence that the second pen disappears as soon as you put it into the box, as it is equal to the pen that is already in the set. So, after putting the second pen into our set, it would still look like this:

$$B = \{\text{shirt}, \text{pen}, \text{coin}, \text{pen}\} = \{\text{shirt}, \text{pen}, \text{coin}\}.$$

Thus, we say that the elements of a set are *distinct*.

Another property that is not stated explicitly in the above definition is that the order of the objects in a set, that is, the order you put your objects into the box, does not matter at all. We consider a set containing an apple and a pear equal to a set containing a pear and an apple:

$$\{\text{apple}, \text{pear}\} = \{\text{pear}, \text{apple}\}.$$

To summarize, a set is an abstract structure that can hold an arbitrary number of (even infinitely many) distinct objects. But, except for the one-letter name, this still does not sound like a real mathematical concept.

Therefore, we want to take a more formal approach to defining sets.

**Definition 4.2** (Set). *Let $\varphi$ be a predicate. Then*

$$\{x \mid \varphi(x)\}$$

*describes the set containing all objects $x$ which $\varphi(x)$ holds for. In practice, we often further simplify such representation by explicitly giving a set from which to draw the possible elements. In other words, if $A$ is a set, then*

$$\{x \in A \mid \varphi(x)\}$$

*describes the set that contains all elements of $A$ that fulfill the predicate $\varphi$.*

If you have read the definition carefully, you might have noticed the term "element" together with a strange-looking symbol, that we have not yet talked about, namely $\in$. This symbol is used for the very important notion of **set membership** which we are going to look at next.

**Definition 4.3** (Set Membership). *Let $A = \{x \mid \varphi(x)\}$. Then, $\in$ denotes a binary predicate that is defined as*

$$a \in A \coloneqq \varphi(a).$$

*If $a \in A$, we say that $a$ is an* ***element*** *of $A$. Otherwise, we write*

$$a \notin A$$

*and say that $a$ is not an element of $A$. Particularly, for every object $x$ either $x \in A$ or $x \notin A$.*

What does that mean in natural language? The set $A$ is defined via the predicate $\varphi$, meaning, it contains all objects $x$ for which $\varphi(x)$ holds. Now, $a \in A$ is assigned whether $\varphi(a)$ holds, which gives us $\top$ if $a$ fulfills the predicate (and is, therefore, an element of $A$) and $\bot$ if it does not (in which case $a$ is not an element of $A$). Ultimately, we use this to decide whether an object belongs to or rather is an element of a certain set. This is what really enables us to argue about sets and their properties.

But before we start doing this, we first want to take a look at a very special set. Back in our box metaphor, of course, you can have a box into which you put nothing. This box is unique and therefore gets its own definition.

**Definition 4.4** (Empty Set). *The set that contains no elements is called the* ***empty set*** *and is denoted with*

$$\emptyset \quad \text{or} \quad \{\}.$$

*Formally, this means*

$$\forall x : x \notin \emptyset.$$

Throughout the rest of the book, and especially in the following chapter about relations, we will explore many ways in which the empty set has special properties in contrast to sets actually containing some elements.

### 4.1.1 Notation

You have probably seen sets written differently than in this predicate notation. This is because, most of the time, you actually use other, better readable notations. There are plenty of ways to write down sets, each with their own advantages and disadvantages.

**Enumeration** This is the most common way to write a set and we have already encountered it in the previous paragraph. You write out all the elements separated by a comma and surrounded by curly braces. For example,

$$F := \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\}$$

denotes the set of the four different card faces. Obviously, this does not work for infinite sets as you cannot write down infinitely many objects.

**Dot Notation** If you do not want to write *all* elements by hand but there is a visible pattern, you can abbreviate a sequence of elements with dots. This also enables you to write sets with infinitely many elements. However, this is not considered a formal notation and therefore rarely seen, especially since the meaning is often neither obvious nor unambiguous.

For example,

$$\{0, 2, 4, 6, \ldots, 42\}$$

denotes the set containing all even numbers until 42 and

$$\{0, 2, 4, 6, \ldots\}$$

denotes the set that holds all even numbers. However, consider the following example:

$$\{1, 2, \ldots\}$$

could express the set of all numbers starting with 1. But it could also denote the set of the powers of two $\{1, 2, 4, 8, 16, \ldots\}$. You can see, this notation is strongly ambiguous because these patterns are in no way formalized and we encourage you to use other notations whenever possible.

**Special Sets** Some sets are so special and important to mathematicians that they were given their own letters. For example,

- $\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$ is the set of the **natural numbers**. There are disputes among mathematicians whether $0 \in \mathbb{N}$ (i.e., 1 is the first natural number) or not. In our case, we state $0 \in \mathbb{N}$. More on natural numbers can be found in the appendix A.
- $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \ldots\}$ is the set of the **integers**.
- $\mathbb{Q} = \left\{\frac{p}{q} \;\middle|\; p \in \mathbb{Z} \land q \in \mathbb{Z}\right\}$ is the set of the **rational numbers**.
- $\mathbb{R}$ is the set of the **real numbers**.
- $\mathbb{B}$ is the set of the booleans $\{0, 1\}$.

**Example 4.5:** Even Numbers

The sets

$$\begin{aligned} E &:= \{n \mid n \in \mathbb{N} \land n \text{ is an even number}\}, \\ E' &:= \{n \in \mathbb{N} \mid n \text{ is divisible by } 2\}, \qquad \text{and} \\ E'' &:= \{0, 2, 4, 6, 8, 10, 12, \ldots\} \end{aligned}$$

all denote the set of all even numbers.
Furthermore, we can say that $42 \in E$ and $1337 \notin E$. Everything that is not a natural number is also not an element of $E$.

**Checkpoint 4.6:** Set Basics

Make sure that you understand the definition of sets and membership, as well as all set notations. Here are some exercises:

- Write the set that contains all powers of two in every possible notation, including predicate notation.

  *Hint:* Keep in mind that mathematical equations can also be predicates, e.g.

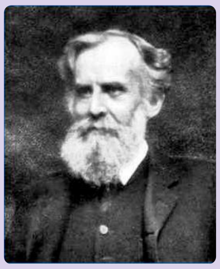
  $$\varphi(x) := x \in \mathbb{N} \land x = 0$$

  is $\top$ if and only if $x$ is a natural number equal to 0.
- Take a look back at the set $E$ of the even numbers from the previous example. There are actually infinitely many notations of that particular set. Can you give more examples and argue why this is the case?
- Give the empty set $\emptyset$ in predicate notation.
- Can you express every set in every notation?

### Visualization with Venn diagrams

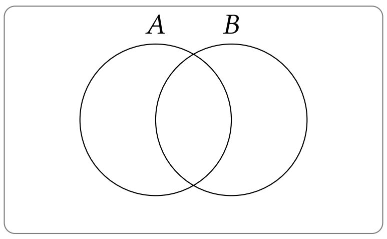
There is another important way how to describe sets, and that is graphically. Especially when dealing with set manipulations, like in the next chapter, meaningful visualizations will help us better understand what different operators do with multiple sets.

**Important Individual:** John Venn



John Venn (1834-1923) was an English mathematician, philosopher, and priest. During his time in Cambridge, he studied and taught logic (especially mathematical and modal logic) and probability theory. In the search for graphical visualizations of sets, he refined the set diagrams previously established by Euler into Venn diagrams, as we call them today.

The whole area of the diagram represents the universe of all possible elements. Then, we draw a circle or an ellipse for each set to be displayed such that they overlap.



In contrast to the other options we have seen, Venn diagrams are not a notation, as they do not represent concrete sets with concrete elements and instead represent them rather abstractly. But this is not the point of the visualization; its function is to represent the *relation* of two or more sets, whatever the elements may be.

**Going Beyond:** Backus–Naur Forms Are Sets?

In the first chapter about formal languages, we got to know BNFs which specify a language. We can regard these languages as sets, namely as the set containing all words that can be derived with the respective language. For example, the simple BNF

$$\mathcal{L}_1 \ni \varphi ::= a \mid b \mid c$$

that does not make use of the meta-variable $\varphi$ corresponds to the set

$$L_1 = \{a, b, c\} .$$

However, this also works when actually using meta-variables:

$$\mathcal{L}_2 \ni \varphi ::= a\varphi a \mid b$$

This gives us the set

$$L_2 = \{b, aba, aabaa, aaabaaa, \ldots\}$$

which has infinitely many elements since with the BNF having rules that use $\varphi$, there are infinitely many words you can produce.

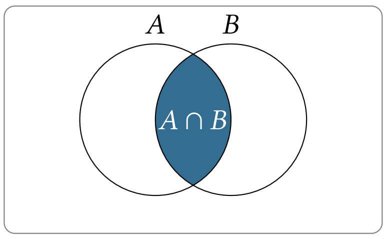
### 4.1.2 Set Operators

#### Basic Definitions

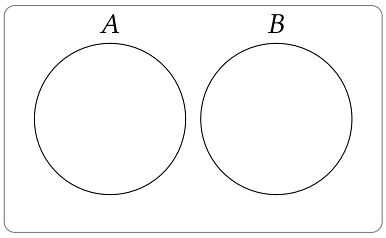
Now that we have a solid foundation to work with, let's take a look at what we can actually do with sets and in which ways we can transform them. All operators will be illustrated using Venn diagrams where the color gray marks the result of the respective operation.

**Definition 4.7** (Intersection)**.** *Let $A$ and $B$ be sets. Then $A \cap B$ denotes the set that contains all elements that are both in $A$ and in $B$. We call this the* ***intersection*** *of $A$ and $B$.*

$$A \cap B = \{x \mid x \in A \land x \in B\}$$

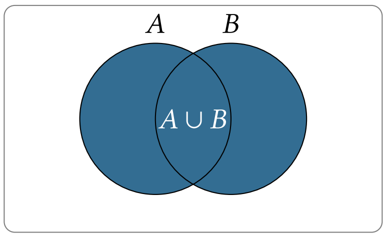


*If the intersection of $A$ and $B$ is empty, that is $A \cap B = \emptyset$, we say that they are* ***disjoint****. Sometimes, you will see this represented by non-overlapping circles in a Venn diagram:*



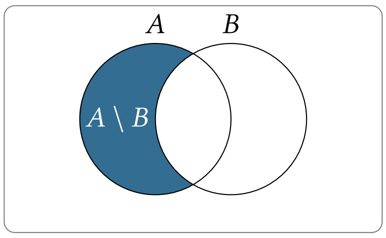
**Definition 4.8** (Union). *Let $A$ and $B$ be sets. Then $A \cup B$ is the set of all elements that are in $A$, $B$, or both. This is called the* ***union*** *of $A$ and $B$.*

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$



**Definition 4.9** (Difference). *Let $A$ and $B$ be sets. Then $A \setminus B$ contains all elements of $A$ that are not in $B$. This is called the* ***difference*** *of $A$ and $B$.*

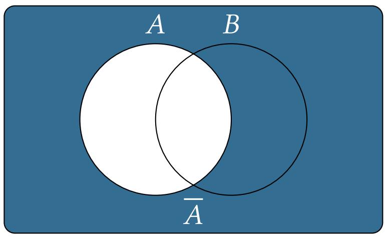
$$A \setminus B = \{x \mid x \in A \wedge x \notin B\}$$



**Definition 4.10** (Complement). *Let $A$ be a sets Then $\overline{A}$ represents the set that contains all elements that are not in $A$, which we also call the* ***complement*** *of $A$.*

$$\overline{A} = \{x \mid x \notin A\}$$

*Another common notation for the complement that you often find in literature is $A^c$.*



That was a lot of definitions all at once, so let's have a look at some concrete examples before we continue.

**Example 4.11:** Set Operations

First, we consider finite sets. Let $A := \{1, 2, 3, 4, 42\}$ and $B := \{3, 4, 5, 42\}$ in our universe $\mathbb{N}$.

- $A \cap B = \{3, 4, 42\}$, $A$ and $B$ are not disjoint.
- $A \cup B = \{1, 2, 3, 4, 5, 42\}$
- $A \setminus B = \{1, 2\}$
- $\overline{A} = \{5, 6, 7, \ldots, 41, 43, 44, \ldots\}$

Everything we have seen so far naturally also applies to infinite sets. Consider $A := \{n \in \mathbb{N} \mid (2 \mid n)\}$ as the set of even numbers and $B := \{n \in \mathbb{N} \mid \neg(2 \mid n)\}$ as the set of odd numbers in the universe $\mathbb{N}$.

- $A \cap B = \emptyset$, so $A$ and $B$ are disjoint.
- $A \cup B = \mathbb{N}$
- $A \setminus B = A$ (as the sets have no shared elements)
- $\overline{A} = B$ and $\overline{B} = A$

Looking at the Venn-diagrams above, we can observe an interesting connection between the last two operators:

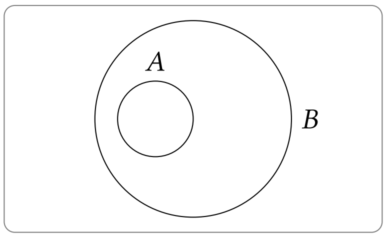
**Theorem 4.12** (Difference as Complement)**.** *Let $A$ and $B$ be sets. Then*

$$A \setminus B = A \cap \overline{B}.$$

Of course, just because it looks like this in the diagram does not mean that is actually true. Therefore, we should prove it—but if we try now, we will find that we do not have the necessary tools yet. So, keep the statement in mind while we lay the foundations for a proof:

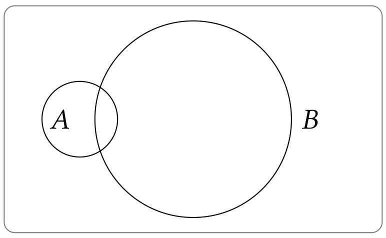
**Definition 4.13** ([Proper] Subset, Superset)**.** *Let $A$ and $B$ be sets. Then we call $A$ a **subset** of $B$ and $B$ a **superset** of $A$ if and only if all elements of $A$ are also elements of $B$, that is*

$$A \subseteq B := (\forall x : x \in A \rightarrow x \in B)\,.$$



*If $A$ is not a subset of $B$, we write*

$$A \not\subseteq B.$$



*Furthermore, we call $A$ a* ***proper subset*** *(or* ***strict subset****) of $B$ (and likewise $B$ a* ***proper superset*** *of $A$) if and only if $A$ is a subset of $B$ and there are elements of $B$ that are not elements of $A$. In short, we say*

$$A \subsetneq B :\iff (A \subseteq B \land \exists x : x \in B \land x \notin A) \iff (A \subseteq B \land A \neq B)\,.$$

*The first Venn diagram depicts a proper subset relation. The second one shows two sets that are not subsets of each other.*

**Example 4.14:** Sets of Card Suits

- The set of the black card suits, $\{♣, ♠\}$ is a subset of the set of all card suits, $\{♡, ♢, ♣, ♠\}$. Therefore, the latter is also a superset of the former.
- In fact, the set of black card suits is a proper subset of the set of all card suits because there exist further suits, ♡ and ♢, that are not black card suits.
- $\{♡, ♢, ♣, ♠\}$ is a subset, but not a proper subset of $\{♡, ♢, ♣, ♠\}$ because there is no element that only occurs in one of the sets.
- The set of all card suits is not a subset of the set containing only the red ones because, for example, $♠ \notin \{♡, ♢\}$.

**Checkpoint 4.15:** Sub- and Supersets

At this point, it makes sense to pause for a moment and think about these definitions. Are there any special cases? Are there any statements that always hold? In particular, given any set $A$, are there other sets that are always (proper) subsets of $A$?

### Equality

When we encounter sets in the wild, we might like to know whether two sets are the same. But what does that even mean? Sets can be very abstract, so how might we compare them? The following definition should give a reasonable notion of what we understand under the term equality in natural language.

**Definition 4.16** (Set Equality)**.** *Two sets $A$ and $B$ are equal if and only if they both contain the same elements, formally $\forall x : x \in A \leftrightarrow x \in B$. Then we write*

$$A = B.$$

*If this condition does not hold, we call them unequal and write*

$$A \neq B.$$

However, if we want to actually prove that two sets are equal or not, we usually use a more formal approach which is described by the following theorem.

**Theorem 4.17** (Set Equality with Subsets)**.** *Two sets are equal if and only if they are subsets of each other. In other words, if $A$ and $B$ are sets, then*

$$A = B \leftrightarrow A \subseteq B \land B \subseteq A.$$

*Proof.* We prove the two directions of the equivalence.

$\longleftarrow$: Let $A \subseteq B$, $B \subseteq A$ and furthermore, $x \in A$. Then by definition, we have $x \in B$ because $A \subseteq B$. Likewise, if $y \in B$, then because of $B \subseteq A$, it follows that $y \in A$. With this, we have shown that every element of $A$ is also in $B$ and every element of $B$ is also in $A$. Therefore, $A$ and $B$ must have the same elements.

$\longrightarrow$: Let $A = B$. Then they both contain the same elements. It follows that if $x \in A$, it is also $x \in B$ and vice versa. With this, we get $A \subseteq B$ and $B \subseteq A$, as both $x \in A \longrightarrow x \in B$ and $x \in B \longrightarrow x \in A$ hold. □

**In Other Words:** (Dis-) Proving Equality Using Mutual Inclusion

As said before, the statement of the previous theorem helps us to prove that two specific sets are equal. Namely, we can show in two steps that they are subsets of each other. This is analogous to proving an equivalence (like above): first, show $A \subseteq B$, then $B \subseteq A$.
It is much easier to show that two sets are unequal: you just have to give a counterexample of an element being in one but not in the other set.

**Going Beyond:** Subset vs. Implication

From this fact, you can get the intuition that $\subseteq$ works like $\longrightarrow$, but on a set level. This is useful when you, for example, look at how *events* (which are nothing other than sets) are defined in probability theory. Namely, if an event $A$ is a subset of another event $B$, then the occurrence of $A$ implies that $B$ also occurs.
Imagine a random experiment where a die is thrown once and the rolled number is checked. We can define events

$$E := \{2, 4, 6\} \text{ and } S := \{6\}$$

as the events that occurs when the number is even or is a 6.
We can clearly see, that $S \subseteq E$. What this means, is that whenever a 6 is rolled and event $S$ occurs, the 6 is also even, meaning $E$ also occurs. In other words, the occurrence of $S$ *implies* the occurrence of $E$.

There is another way to characterize set equality without needing the notion of subsets.

**Theorem 4.18** (Set Equality with Predicates)**.** *Let $A = \{x \mid \varphi(x)\}$ and $B = \{x \mid \psi(x)\}$ be two sets that are produced by predicates $\varphi$ and $\psi$ respectively. Then*

$$A = B \leftrightarrow \forall x : \varphi(x) \leftrightarrow \psi(x).$$

*Proof.* Again, we prove the two directions of the equivalence.

$\longrightarrow$: Let $A = B$. Then every $x \in A$ satisfies both $\varphi$ and $\psi$ because it is also an element of $B$. Every $x \notin A$ is also not in $B$ and therefore $\varphi(x) = \psi(x) = \bot$ in this case. From $\varphi$ and $\psi$ always evaluating to the same value for every $x$, we can conclude that $\varphi \equiv \psi$.

$\longleftarrow$: Let $\varphi$ and $\psi$ be equivalent predicates. Then for every $x$ it holds $\varphi(x) \leftrightarrow \psi(x)$. So, if $x$ satisfies both of them, it is also an element of both sets $A$ and $B$ and in the other case, it is neither element of $A$ nor of $B$. Therefore, $x$ is either in both sets or none of them. Thus, $A = B$. □

**In Other Words:** (Dis-) Proving Equality Using Predicates

This approach can be especially helpful if the predicates of the two sets you are examining are similar or easily transformed into one another. In this case, you would use the laws that you encountered in the previous chapters.

If you remember, we still have a proof to catch up on which we now finally can do. The claim was that $A \setminus B = A \cap \overline{B}$ (Theorem 4.12).

*Proof: Difference as complement.* We use the first approach from above and prove two directions.

$\subseteq$: Let $x \in A \setminus B$. Then, by definition of set difference, we get $x \in A \land x \notin B$. From the latter, we can deduce $x \in \overline{B}$ which, in turn, together with $x \in A$ yields $x \in A \cap \overline{B}$.

$\supseteq$: Let $x \in A \cap \overline{B}$. Then, by definition of intersection, we get $x \in A \land x \in \overline{B}$. From the latter, we can deduce $x \notin B$. Again, combining $x \in A$ and $x \notin B$, we get $x \in A \setminus B$. □

**Checkpoint 4.19:** (Dis-) Proving Equality of Sets

Make sure that you understand both approaches of how to show that two sets are equal. Try it out for yourself:

- Show that $\{n \in \mathbb{N} \mid n < 43\} = \{n \in \mathbb{N} \mid n \leq 42\}$.
- Show that $\mathbb{Z} \neq \mathbb{N}$.

### Operators for More than Two Sets

Sometimes we have large expressions with unions or intersections that can contain any number of or even infinitely many sets. Therefore, we need an easy way to express such terms which we want to give with the next definitions.

**Definition 4.20** (Arbitrary Intersection and Union). *We have different ways of abbreviating intersections and unions.*

(1) *Let $I$ be a set containing sets, that is, every $A \in I$ is a set. Then*

$$\bigcap_{A \in I} A := \{x \mid \forall A \in I : x \in A\} \qquad \textit{and} \qquad \bigcup_{A \in I} A := \{x \mid \exists A \in I : x \in A\}$$

*denote the intersection or the union, respectively, of all sets that are in $I$.*

(2) *Let $I$ be a subset of the integer numbers, that is $I \subseteq \mathbb{Z}$ (It also works with the natural numbers $\mathbb{N}$). Furthermore, let $A_i$ be a set for every $i \in I$. In this case, we call $I$ an* ***index set*** *because it contains the possible indices for the set sequence. Then*

$$\bigcap_{i \in I} A_i = \bigcap \{A_i \mid i \in I\} \qquad \textit{and} \qquad \bigcup_{i \in I} A_i = \bigcup \{A_i \mid i \in I\}$$

*denote the intersection or the union, respectively, of all sets $A_i$ with $i \in I$.*

(3) *If, in the above case, $I$ is an ongoing sequence of numbers starting at $a \in \mathbb{Z}$ and ending at $b \in \mathbb{Z}$, we can also write*

$$\bigcap_{i=a}^{b} A_i = \bigcap \{A_i \mid a \le i \le b\} \qquad \textit{or} \qquad \bigcup_{i=a}^{b} A_i = \bigcup \{A_i \mid a \le i \le b\}$$

**Example 4.21:** Arbitrary Intersections and Unions

For each of the three ways, we take a look at notations in action.

(1) Let $I = \{\{\heartsuit, \spadesuit, \diamondsuit\}, \{\diamondsuit, \clubsuit\}, \{\heartsuit, \clubsuit, \diamondsuit\}\}$. Then

$$\bigcap_{A \in I} A = \{\heartsuit, \spadesuit, \diamondsuit\} \cap \{\diamondsuit, \clubsuit\} \cap \{\heartsuit, \clubsuit, \diamondsuit\} = \{\diamondsuit\},$$

$$\bigcup_{A \in I} A = \{\heartsuit, \spadesuit, \diamondsuit\} \cup \{\diamondsuit, \clubsuit\} \cup \{\heartsuit, \clubsuit, \diamondsuit\} = \{\heartsuit, \spadesuit, \diamondsuit, \clubsuit\}$$

(2) Let $I = \{1, 2, 3\}$ and $A_i = \{i, i+1, i+2\}$ (for example, $A_1 = \{1, 2, 3\}$). Then we have

$$\bigcap_{i \in I} A_i = \{1, 2, 3\} \cap \{2, 3, 4\} \cap \{3, 4, 5\} = \{3\},$$

$$\bigcup_{i \in I} A_i = \{1, 2, 3\} \cup \{2, 3, 4\} \cup \{3, 4, 5\} = \{1, 2, 3, 4, 5\}$$

(3) Let $A_i$ be the same sets as in case (2). Then

$$\bigcap_{i=1}^{3} A_i = \{3\} \qquad \text{and} \qquad \bigcup_{i=1}^{3} A_i = \{1, 2, 3, 4, 5\}$$

**Going Beyond:** Membership of Set Union

Can you find a formal definition for $x \in \bigcup_{i \in I} A_i$ that only uses first-order logic with set membership?

### 4.1.3 Tuples

We have now encountered sets as constructs that can hold any number of distinct and unordered objects. This is useful in many cases; however, sometimes, the order of elements is important.

For example, consider the set containing the last six presidents of the United States of America.[1] That would be

$$\{\text{Obama}, \text{Biden}, \text{Bush}, \text{Trump}, \text{Clinton}\}\ .$$

As the order of elements in a set does not matter, this is a correct representation.

But hopefully, you will notice two things: first, that is not the order they were in office. However, in the current context, where the set should describe *the last* six US presidents, we might want that information preserved in the notation.

The second problem is that the set only has *five* elements which, at first glance, is remarkable for a set containing the last *six* American presidents. So who is missing? Correct, it is the second Bush. And we have also seen the reason for this before: in our notation, we cannot distinguish the Bushes, meaning Bush = Bush, which is why the name only occurs once in set notation. But this is also not a suitable solution for our problem, we want to write down the *six* last presidents *in order*.

Luckily, mathematicians have come up with a clever way to achieve this:

**Definition 4.22** (Tuple). *A* ***tuple*** *is an* ordered *collection of any* finite number *of objects. In contrast to sets, these objects do not need to be distinct. We write*

$$(a_1, a_2, \ldots, a_n)$$

*for a tuple with the elements $a_1$ to $a_n$ for $i = 1$ to $n \in \mathbb{N}$.*

*A tuple with two elements is often called a* ***pair*** *and one containing three elements is called a* ***triple****.*

Finally, we have a tool to write down the six last US presidents:

$$(\text{Biden}, \text{Trump}, \text{Obama}, \text{Bush}, \text{Clinton}, \text{Bush})$$

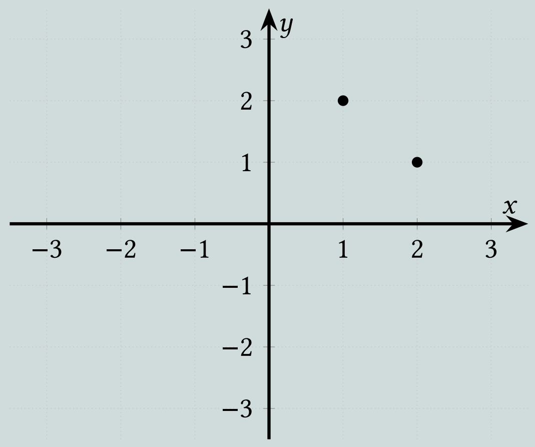
This time, the order is correct and the tuple contains both Bushes.

To reiterate the properties of tuples, we take another look, this time in a more mathematical context.

[1] https://en.wikipedia.org/wiki/List_of_presidents_of_the_United_States

**In Other Words:** Tuples and Coordinates

Consider the coordinate system of the real plane $\mathbb{R}^2$. In school, you may have encountered it when drawing function graphs. It contains *points* which are, in fact, nothing other than tuples.



As you can see in the figure above, the tuple $(1, 2)$ is not the same as $(2, 1)$. The meaning of the numbers (whether they are the $x$ or $y$ value) depends on their position in the tuple. Also, $(1, 1)$ is not the same as $(1)$ as it would be in a set context. A point with only one coordinate does not make sense here; they always need two coordinates.

Having defined tuples, we can now take a look at another very important set operator:

**Definition 4.23** (Cartesian Product)**.** *Let $A$ and $B$ be sets. Then we define*

$$A \times B := \{(a, b) \mid a \in A, b \in B\}$$

*as the **cartesian product** of $A$ and $B$.*

That definition might look a little overwhelming but if we try to describe what it does in words, it becomes rather harmless.

$A \times B$ combines every single element $a \in A$ with every element $b \in B$ and puts those combinations into a tuple. That makes $A \times B$ a set containing only tuples where the first component is an element of $A$ and the second one is an element of $B$.

**Example 4.24:** Playing Cards

Earlier, we talked about card faces $F = \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\}$. Using the cartesian product we now can expand to get a representation of whole cards. We only need one additional set, namely the card values $V := \{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\}$. Herewith, we can now produce

$$\begin{aligned} C &:= F \times V \\ &= \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\} \times \{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\} \\ &= \{(\heartsuit, 2), \ldots, (\heartsuit, A), (\diamondsuit, 2), \ldots, (\diamondsuit, A), (\spadesuit, 2), \ldots, (\spadesuit, A), (\clubsuit, 2), \ldots, (\clubsuit, A)\} \end{aligned}$$

where every tuple in $C$ can be interpreted as a card with the face at its first position and the value at its second position.

However, the cartesian product is not only able to combine two sets. We can extend it in such a way that it can combine an arbitrary but finite amount of sets (as we demand that tuples only have finitely many elements). How does this work?

Look at the expression

$$A \times B \times C$$

where $A$, $B$, and $C$ are sets. All other operators you have encountered until now had an implicit precedence rule when parentheses were omitted. For example, $1000 + 300 + 30 + 7$ would actually be $(((1000 + 300) + 30) + 7)$, as we defined $+$ to be left associative. In the case of $\times$, we do not want that kind of rule as it would disable us from stating the following definition:

**Definition 4.25** (Cartesian Product for Multiple Sets)**.** *Let $A_0, \ldots, A_n$ be sets for $n \in \mathbb{N}$. We define*

$$A_0 \times \cdots \times A_n := \{(a_0, \ldots, a_n) \mid a_0 \in A_0, \ldots a_n \in A_n\}$$

*as the **cartesian product** of $A_0, \ldots, A_n$.*

This means that the operator is not binary but rather $n$-ary, where $n$ is the number of sets to combine.

Finally, we want to introduce a little bit of syntactic sugar in case we use the cartesian product on only one set multiple times: if $A$ is a set, then we can simply write

$$A^n \text{ instead of } \underbrace{A \times \cdots \times A}_{n \text{ times}}.$$

You might now be wondering how many tuples are in such a cartesian product, and you might even have an idea what the answer is. Unfortunately, though, we do not yet have the tools to tackle this, so we need to postpone this until a later point.

**Example 4.26:** Subjects

Let's say we have the following three sets and for your final exam, you need one subject out of every one of these. What are all possible combinations for how to take your exams? (For the sake of simplicity, we also assume that we are interested in the temporal order of the exams.)

$$\begin{aligned}\text{NaturalSciences} &= \{\text{Maths}, \text{CS}\}, \\ \text{Languages} &= \{\text{German}, \text{English}, \text{French}\}, \\ \text{SocialSciences} &= \{\text{Geography}, \text{History}\}\end{aligned}$$

We can get them by finding out

$$\text{NaturalSciences} \times \text{Languages} \times \text{SocialSciences}$$

which is equal to the set

$$\begin{gathered}\{(\text{Maths}, \text{German}, \text{Geography}), (\text{Maths}, \text{German}, \text{History}), \\ (\text{Maths}, \text{English}, \text{Geography}), (\text{Maths}, \text{English}, \text{History}), \\ (\text{Maths}, \text{French}, \text{Geography}), \dots\}\end{gathered}$$

Just like before, we combine all elements of a set with every element of the remaining sets. This is hugely different if we were to consider the same expression but with parentheses:

$$\begin{gathered}(\text{NaturalSciences} \times \text{Languages}) \times \text{SocialSciences} \\ = \{((\text{Maths}, \text{German}), \text{Geography}), ((\text{Maths}, \text{English}), \text{Geography}), \\ ((\text{Maths}, \text{French}), \text{Geography}), ((\text{CS}, \text{German}), \text{Geography}), \\ ((\text{CS}, \text{English}), \text{Geography}), ((\text{CS}, \text{French}), \text{Geography}), \dots\},\end{gathered}$$

where we first get the cartesian product NaturalSciences × Languages and then combine those tuples with all social sciences. This gives us tuples which have tuples and social sciences as elements. In fact, every tuple of the operation with parentheses has these two elements, whereas without them, as we have seen above, the tuples have three elements each.

### 4.1.4 Laws of Set Theory

Now that we know sets as well as their operators, we will want to find some useful statements about them and prove them. This is relatively straightforward, as our definitions rely on the logic operators we introduced in Lemma 2.22.

The following theorem states all the properties that you have already seen in a propositional logic context.

**Theorem 4.27** (Laws of Set Theory). *Let $\mathcal{U}$ be the set containing all elements, also called the universe. Furthermore, let A, B and C be sets. The following statements hold:*

**Commutativity**
$A \cup B = B \cup A$
$A \cap B = B \cap A$

**Associativity**
$(A \cup B) \cup C = A \cup (B \cup C)$
$(A \cap B) \cap C = A \cap (B \cap C)$

**Distributivity**
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

**Identity**
$A \cup \emptyset = A$
$A \cap \mathcal{U} = A$

**Complement laws**
$A \cup \overline{A} = \mathcal{U}$
$A \cap \overline{A} = \emptyset$

**Idempotence laws**
$A \cup A = A$
$A \cap A = A$

**Domination laws**
$A \cup \mathcal{U} = \mathcal{U}$
$A \cap \emptyset = \emptyset$

**Absorption laws**
$A \cup (A \cap B) = A$
$A \cap (A \cup B) = A$

**De Morgan's laws**
$\overline{A \cup B} = \overline{A} \cap \overline{B}$
$\overline{A \cap B} = \overline{A} \cup \overline{B}$

**Double complement law**
$\overline{\overline{A}} = A$

*Proof.* Over the course of this proof, let $A = \{x \mid \varphi(x)\}$, $B = \{x \mid \psi(x)\}$ and $C = \{x \mid \chi(x)\}$ for the respective predicates $\varphi$, $\psi$ and $\chi$. Please remember that we defined $x \in A \equiv \varphi(x)$ as we will use this quite frequently.

**Commutativity**
We know that $\wedge$ and $\vee$ are commutative and therefore we get

$$A \cup B = \{x \mid \varphi(x) \vee \psi(x)\} = \{x \mid \psi(x) \vee \varphi(x)\} = B \cup A$$
$$A \cap B = \{x \mid \varphi(x) \wedge \psi(x)\} = \{x \mid \psi(x) \wedge \varphi(x)\} = B \cap A$$

**Associativity**
We know that $\wedge$ and $\vee$ are also associative which yields

$$(A \cup B) \cup C = \{x \mid (\varphi(x) \vee \psi(x)) \vee \chi(x)\} = \{x \mid \varphi(x) \vee (\psi(x) \vee \chi(x))\} = A \cup (B \cup C)$$
$$(A \cap B) \cap C = \{x \mid (\varphi(x) \wedge \psi(x)) \wedge \chi(x)\} = \{x \mid \varphi(x) \wedge (\psi(x) \wedge \chi(x))\} = A \cap (B \cap C)$$

**Distributivity**
Follows similarly to (a) and (b).

**Identity**

As the predicate describing $\emptyset$ is $\bot$, we can write

$$A \cup \emptyset = \{x \mid \varphi(x) \vee \bot\} = \{x \mid \varphi(x)\} = A.$$

This follows by the identity law of $\vee$.

Similarly, we show the second equation. The predicate of $\mathcal{U}$ is simply $\top$, as it represents the set that contains all objects and $\top$ is the predicate that is true for every object. So we get

$$A \cap \mathcal{U} = \{x \mid \varphi(x) \wedge \top\} = \{x \mid \varphi(x)\} = A.$$

Again, we use the identity law for $\wedge$.

**Complement laws**

We can write

$$A \cup \overline{A} = \{x \mid \varphi(x) \vee \neg\varphi(x)\} = \{x \mid \top\} = \mathcal{U}$$

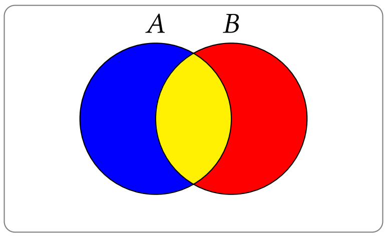
and

$$A \cap \overline{A} = \{x \mid \varphi(x) \wedge \neg\varphi(x)\} = \{x \mid \bot\} = \emptyset$$

using the negation laws for $\wedge$ and $\vee$.

The remaining proofs are left to you as an exercise. □

Another useful fact concerns the union of two sets. Namely, we aim to separate the union set into disjoint subsets that enable us to make further observations.



From this diagram, we can easily identify three subsets that must be disjoint. This leads us to the following statement:

**Theorem 4.28** (Disjoint Decomposition of Union)**.** *Let A and B be sets. Then*

$$A \cup B = (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$$

*is a disjoint union, meaning that the three sets are pairwise disjoint.*

First, what does “pairwise” mean?

“pairwise” is an expression that is commonly used to say that a property holds *for each possible pair* in a range of specified objects. If we consider our current context, we (claim to) have three *pairwise disjoint* sets, so each possible pair of those sets is disjoint: $(A\setminus B)\cap(B\setminus A) = \emptyset$, $(A\setminus B)\cap(A\cap B) = \emptyset$, and $(B \setminus A) \cap (A \cap B) = \emptyset$.

*Proof.* We first show equality and then disjointness.

$\subseteq$: Let $x \in A \cup B$. Then, by definition, $x \in A$ or $x \in B$. We look at both cases. First, let $x \in A$. Then it is possible that $x \in B$ or $x \notin B$. In the first case, we can conclude $x \in A \cap B$ because then $x \in A \land x \in B$. In the second case, we get $x \in A \setminus B$ because $x \in A \land x \notin B$. Now let $x \in B$. Again, we determine whether $x \in A$ or $x \notin A$. If $x \in A$, then by $x \in B \land x \in A$ we conclude $x \in A \cap B$; if $x \notin A$ then by definition we have $x \in B \setminus A$. Therefore, $A \cup B \subseteq (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$.

$\supseteq$: Let $x \in (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$. Then $x$ is an element of at least one of those three sets. If $x \in A \setminus B$, then $x \in A$ and therefore $x \in A \cup B$. If, on the other hand, $x \in B \setminus A$, then by definition $x \in B$ and therefore $x \in A \cup B$. For the case $x \in A \cap B$ we can also conclude $x \in A \cup B$ because $x$ is both in $A$ and $B$. Hence, $(A \setminus B) \cup (B \setminus A) \cup (A \cap B) \subseteq A \cup B$.

Another way of proving is to use the laws that were introduced before. You are encouraged to try this out for yourself. We have now proven that the sets are equal, but we still need to show that the three sets are pairwise disjoint. In the definition of “pairwise” above, we have already seen our three proof obligations. We prove these by contradiction.

- We assume $(A \setminus B) \cap (B \setminus A) \neq \emptyset$, meaning that $\exists x \in (A \setminus B) \cap (B \setminus A)$.

  Then by the definitions of the operators, we get $x \in A \setminus B \land x \in B \setminus A$. If we further simplify this, we can follow $x \in A \land x \notin B \land x \in B \land x \notin A$ ↯. There lies a contradiction (even two contradictions), because neither $x \in A \land x \notin A$ nor $x \in B \land x \notin B$ is possible. Therefore, the assumption must have been false, hence $(A \setminus B) \cap (B \setminus A) = \emptyset$.

  The remaining two cases follow similarly.

- We assume $(A \setminus B) \cap (A \cap B) \neq \emptyset$, meaning that $\exists x \in (A \setminus B) \cap (A \cap B)$.

  Again, we use the definitions of the operators, which yields $x \in A \land x \notin B \land x \in A \land x \in B$ ↯. This time, the contradiction is $x \in B \land x \notin B$, meaning the converse of the assumption must hold, giving us $(A \setminus B) \cap (A \cap B) = \emptyset$.

- We assume $(B \setminus A) \cap (A \cap B) \neq \emptyset$, meaning that $\exists x \in (B \setminus A) \cap (A \cap B)$.

  As this case is analogous to the previous one, you are encouraged to try it out for yourself.

Now we showed that the three sets are pairwise disjoint. □

As you might have noticed, we used a lightning symbol ↯ wherever we can derive a contradiction. This is to make clear, where exactly we arrived at a false statement. Be careful though, this is *not* to be used when proving a statement by contraposition as there is no contradiction there.

### 4.1.5 Cardinality of Finite Sets

There is one question about sets that we have completely ignored so far and that is the question about the number of elements a set contains. For infinite sets, we need to push this question further aback, because we first need some more prerequisites. But we are already able to examine finite sets.

**Definition 4.29.** *Let $A$ be a set with finitely many elements. We call the number of elements in $A$ the* ***cardinality*** *of $A$ and denote it with*

$$|A|.$$

*Sometimes, you will also see the notation $\#A$ used instead.*

If there are only finitely many elements in a set, we can simply count them in order to get the cardinality.

**Example 4.30:** Cardinality of Finite Sets

- $|\{1,2,3\}| = 3$ as it contains 3 distinct elements.
- $|\{1,2,3,2\}| = 3$ as every element is counted only once, resulting in the same set as before.
- $|\{\{1,2,3\}\}| = 1$ as the set contains one set. The cardinality of the inner set would be 3.
- $|\emptyset| = 0$ as it contains 0 elements.

As we have defined several operators above, we are interested in how they influence the cardinality of the resulting sets. For finite sets, we can make a few interesting observations, starting with the following rule.

**Theorem 4.31** (Cardinality of Disjoint Sets)**.** *Let $A$ and $B$ be disjoint sets, that is, $A \cap B = \emptyset$. Then*

$$|A \cup B| = |A| + |B|.$$

*Proof.* Let $x \in A \cup B$. Then there are two cases. First, if $x \in A$, then $x \notin B$. The same holds vice versa, namely $x \in B \to x \notin A$. Therefore, no element gets lost when creating the union $A \cup B$ as all elements are distinct. So the cardinality of the union is the sum of the distinct elements of $A$ and $B$ from which follows that $|A \cup B| = |A| + |B|$. □

**Theorem 4.32** (Cardinality Properties)**.** *Let $A$ and $B$ be finite sets. Then these statements hold:*

*(a)* $|A \cup B| + |A \cap B| = |A| + |B|$

*(b)* $A \cap B = \emptyset \leftrightarrow |A \cup B| = |A| + |B|$

*(c)* $A \subseteq B \leftrightarrow |B \setminus A| = |B| - |A|$

*(d)* $A \subseteq B \to |A| \leq |B|$

*(e)* $A \subsetneq B \to |A| < |B|$

*Proof.* We prove each statement separately:

(a) With the theorem of the disjoint decomposition (Theorem 4.28) and by applying Theorem 4.12 we get $A \cup B = (A \cap \overline{B}) \cup (B \cap \overline{A}) \cup (A \cap B)$. As the three sets are disjoint, we can follow that

$$|A \cup B| = |A \cap \overline{B}| + |B \cap \overline{A}| + |A \cap B|.$$

We now add $|A \cap B|$ on both sides, yielding

$$|A \cup B| + |A \cap B| = |A \cap \overline{B}| + |A \cap B| + |B \cap \overline{A}| + |A \cap B|.$$

The left side already looks promising, leaving us with the right side of the equation to focus on. If you look closely, you will notice that there are two pairs you can separate. On the one hand, we have $|A \cap \overline{B}| + |A \cap B|$ and on the other hand, there is $|B \cap \overline{A}| + |A \cap B|$.

The important fact here is that the sets both sums concern are disjoint, meaning $(A \cap \overline{B}) \cap (A \cap B) = \emptyset = (B \cap \overline{A}) \cap (A \cap B)$. Therefore, we can again apply Theorem 4.31 and get

$$|A \cup B| + |A \cap B| = |(A \cap \overline{B}) \cup (A \cap B)| + |(B \cap \overline{A}) \cup (A \cap B)|.$$

Now we are almost finished, we only need to show that $(A \cap \overline{B}) \cup (A \cap B) = A$ and $(B \cap \overline{A}) \cup (A \cap B) = B$. We do this by using the laws from the previous section.

$$\begin{aligned} (A \cap \overline{B}) \cup (A \cap B) &= A \cup (\overline{B} \cap B) && \mid \text{Distributivity} \\ &= A \cup \emptyset && \mid \text{Complement law} \\ &= A && \mid \text{Identity} \end{aligned}$$

and we conclude the same for $B$ analogously. Applied to the last equation, this yields

$$|A \cup B| + |A \cap B| = |A| + |B|.$$

(b) We prove the two directions.

$\longrightarrow$: Let $A \cap B = \emptyset$. Then by (a), it follows directly that

$$\begin{aligned} |A| + |B| &= |A \cup B| + |A \cap B| \\ &= |A \cup B| + |\emptyset| && \mid A \cap B = \emptyset \\ &= |A \cup B| + 0 \\ &= |A \cup B| \end{aligned}$$

$\longleftarrow$: Let $|A| + |B| = |A \cup B|$. Then by (a), we can follow

$$|A| + |B| = |A \cup B| + |A \cap B|$$

which, in turn, gives us

$$|A \cup B| = |A \cup B| + |A \cap B|.$$

This can only be true if $|A \cap B| = 0$, so $A \cap B = \emptyset$.

For the remaining statements, only a short proof sketch is layed out that is able to give an intuition for how to proof basically works. It is left to you to work out the formalities.

(c) Let $A \subseteq B$. So, $B \setminus A$ contains every element of $B$ that is not in $A$. As it does not contain any elements that were not in $B$ in the first place, the number of elements can be counted as the elements of $B$ and then subtracting the elements that are also in $A$. Thus, $|B \setminus A| = |B| - |A|$. This only works, because both sets are finite.

(d) Let $A \subseteq B$. We prove this statement by contraposition, so we show $|A| > |B| \longrightarrow A \nsubseteq B$. So, let $|A| > |B|$. Then, even if $A$ contains every element of $B$ there must be at least one more element in $A$ that is not in $B$. Therefore, $A \nsubseteq B$.

(e) $A \subsetneq B \longrightarrow |A| < |B|$ We only need to show $A \subsetneq B \longrightarrow |A| \neq |B|$ as $|A| \leq |B|$ already follows from (c) as $A \subsetneq B \longrightarrow A \subseteq B$.

We prove this by contradiction. Let $A \subsetneq B$ and $|A| = |B|$. Then, because of the subset relationship, $A$ must contain every element of $B$ ↯. But this violates the property of the proper subset because there is no element unique to $B$. Therefore, $|A| \neq |B|$. □

In a previous section, we have dealt with the cartesian product and raised the question how many elements, that is, tuples, it contains. Now having defined cardinality, we can finally formalize the problem and find an answer.

First, let's consider only a binary cartesian product, that is, combining two sets. Let's further assume that the first one, $A$, has 3 and the second one, $B$, has 4 elements. Now, in order to get the cartesian product $A \times B$, we have to combine each of the 3 elements of $A$ with each of the four elements of $B$. This would give us 12 elements for $A \times B$, meaning $|A \times B| = 12$.

Before you read on, try to verify the intuition explained in the paragraph above for products consisting of three, four, or even more sets and write down a statement. Hopefully, you will arrive at the same result as the following theorem.

**Theorem 4.33** (Cardinality of the Cartesian Product). *Let $n \in \mathbb{N}$ and $A_0, \ldots, A_n$ be sets. Then*

$$|A_0 \times \cdots \times A_n| = |A_0| \times \cdots \times |A_n|,$$

*where on the right-hand side, $\times$ denotes the multiplication of natural numbers.*

Unfortunately, we do not have the means to prove this yet. For this, you need a new proof concept which is described in the last chapter.

### 4.1.6 Power Sets

Previously, we have introduced the notion of subsets. An interesting question you might come up with is: what are all the possible subsets any set can have? And how many of them can exist? You could even go further: how many subsets with a given number of elements does any set have?

To answer all these questions, we need another important term from set theory which we define in the following.

**Definition 4.34** (Power Set)**.** *Let $A$ be a set. Then we define*

$$\mathcal{P}(A) := \{M \mid M \subseteq A\}$$

*as the **power set** of $A$. Other common notations are $2^A$ or $\mathfrak{P}(A)$.*

What this means is that $\mathcal{P}(A)$ is a set that contains every subset of $A$ and nothing else.

Having defined the power set, we first want to look at some of its simple properties.

**Theorem 4.35.** *Let $A$ be a set. Then*

$$\emptyset \in \mathcal{P}(A) \quad \text{and} \quad A \in \mathcal{P}(A).$$

You might have stumbled over this statement before, namely in Checkpoint 4.15 after the definition of the subset property. Now, with this theorem, we give an answer to the question that was asked back there.

*Proof.* In order to prove for a set to be in the power set of $A$, it is sufficient to show that it is a subset of $A$.

So first, take $\emptyset$. For every $x$, by definition, $x \in \emptyset$ is false, so the implication $x \in \emptyset \longrightarrow x \in A$ is true. Thus, it follows $\emptyset \subseteq A$ and, therefore, $\emptyset \in \mathcal{P}(A)$.

Now we take a look at the set $A$ itself. For every $x \in A$ it holds that $x \in A \longrightarrow x \in A$. So, we can conclude that $A \subseteq A$ and, therefore, $A \in \mathcal{P}(A)$. □

**Example 4.36:** Power Sets

- $\mathcal{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\ \{1,2,3\}\}$
- $\mathcal{P}(\{1,2\}) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$
- $\mathcal{P}(\{\{1,2\}\}) = \{\emptyset, \{1,2\}\}$
- $\mathcal{P}(\{1\}) = \{\emptyset, \{1\}\}$
- $\mathcal{P}(\emptyset) = \{\emptyset\}$

Now, we can take care of the questions about the cardinality of power sets from before. Unfortunately, we still lack the strategies to prove this, however, we can get an intuition and formulate the statement. We postpone the proof to Section 5.1.

If you take a look at the examples above, you might notice a pattern. Before you read on, please try to find it for yourself. If the examples are not sufficient, try to write down the power set of a set with a cardinality of four (for example, the set $F$ of all card faces).

We are looking for a relationship between the cardinality of any set to the cardinality of its power set. So, according to the example above, we can make the following observations:

- If a set has a cardinality of 3, its power set seems to have 8 elements.
- If a set has a cardinality of 2, its power set seems to only have 4 elements.
- If a set has only one element, its power set seems to have 2 elements.
- The set containing 0 elements, that is, the empty set $\emptyset$ has a power set of cardinality 1.

This calls for the powers of 2: they give us

- $|\{1,2,3\}| = 3$, hence $|\mathcal{P}(\{1,2,3\})| = 2^3 = 8$
- $|\{1,2\}| = 2$, hence $|\mathcal{P}(\{1,2\})| = 2^2 = 4$
- $|\{1\}| = 1$, hence $|\mathcal{P}(\{1\})| = 2^1 = 2$
- $|\emptyset| = 0$, hence $|\mathcal{P}(\emptyset)| = 2^0 = 1$

It is important to add that this kind of intuition cannot replace a formal proof in any case. The ideas above are considerations that you might come up with and should formulate yourself when you are on your way to exploring a problem for yourself. Developing such an intuition can be a good first step before you formalize and then, at last, prove a statement.

Having completed this step now, we want to formalize the claim that we have derived from our observations above.

**Theorem 4.37.** *Let $A$ be a set of cardinality $n \in \mathbb{N}$, that is $|A| = n$. Then*

$$|\mathcal{P}(A)| = 2^n.$$

We might not be able to prove this yet, but at least the statement gives an intuition why the power set of a set $A$ is also commonly noted as $2^A$.

**Going Beyond:** Exploring Further

The second question we asked at the beginning of this paragraph is a bit more tricky to answer: given a set of cardinality $n \in \mathbb{N}$, how many subsets with a given number of elements $i \in \mathbb{N}$ are there?
When trying to solve this problem, try to proceed as described above. First, explore. Run the numbers on concrete examples. Then, order your thoughts and think of possible patterns or relations. Finally, try to formulate the claim you came up with. The proof can wait until the last chapter.

**Going Beyond:** Power Sets in the Wild

Power sets are an important mathematical term, and you will most likely run across them quite often. Here is a brief (and thus by far not complete) overview of where they are present.

- The power set is relevant to probability theory. It can define the space of possible events of a random experiment. But on a closer look, this needs some refining. Therefore, the branch of measure theory concerns itself with constructing algebras on the basis of the power set in order to be able to define a probability measure on the event space.
- It also plays a crucial role in answering the question of the cardinality of infinite sets: does every infinite set have the same cardinality, or are there still differences, that is, different levels of infinity? In particular, Cantor made an important statement on the connection between power sets and infinite cardinality. We look at this problem again in Section 4.2.5.
- It is also central to mathematical relations as they are completely based on the concept of the power set. In the next chapter, we will dive deeper into this matter.

## 4.2 Relations

**Chapter Goals**

In this chapter, we discuss the fundamental concepts of relations. You will learn about

- What relations are and how to formally write them down
- Important properties of relations
- Using relations to classify the cardinality of infinite sets

In computer science, we often work with objects that are somehow connected or related to one another. One of the best-known examples for this are two arbitrary numbers, for which we can say that they are equal or one of them is greater than the other. Consider 13 and 37: obviously, $13 \leq 37$ holds while $37 \leq 13$ is wrong. Mathematically speaking, $\leq$ is a relation which connects 13 to 37 but not the other way around.

Another way of looking at relations: think of a set $W$ which contains different words (e.g. "apple," "beaver," "car," …). We can construct a relation that connects each word to the number of letters it consists of. This relation contains pairs like $(\text{"apple"}, 5)$, $(\text{"beaver"}, 6)$ and $(\text{"car"}, 3)$. As you can see, relations don't necessarily connect objects from one set to objects from the same set, but can also connect objects from one set to objects from a different set.

To keep things simple, we only focus on binary relations in this chapter even though other kinds of relations exist. Let's take a look at their mathematical definition:

**Definition 4.38** (Binary Relation)**.** *Let $A$ and $B$ be arbitrary sets.*

*A subset $\mathcal{R} \subseteq A \times B = \{(a, b) \mid a \in A, b \in B\}$ is called* ***binary relation*** *on $A$ and $B$.*
*Elements of the relation $(a, b) \in \mathcal{R}$ are often written in so-called* ***infix notation*** *as $a\mathcal{R}b$.*

*The set $A$ is referred to as* ***source set****, $B$ is referred to as* ***target set****.*
*If $A = B$ holds, $\mathcal{R}$ is said to be defined on the* ***universal set*** *$A$.*

As you can see, relations are defined as subsets of the Cartesian product of their source and target set. This means that any binary relation is a set of ordered pairs where the first component stems from the source and the second component stems from the target set. If a pair of elements $(a, b)$ is contained in a relation $\mathcal{R}$, we can say "$a$ is related to $b$ regarding $\mathcal{R}$."

You probably have already used the infix notation for relations without even noticing: $13 \leq 37$ is one example where it's common to put the relation as an operator in between the two related elements.

**Going Beyond:** $n$-ary Relations

Apart from binary relations, there are also relations connecting more than just two objects. A relation defined on $n$ sets $M_1, M_2, \ldots, M_n$ is called $n$-ary relation. It is a subset of the Cartesian product $M_1 \times M_2 \times \cdots \times M_n$, therefore containing $n$-tuples.
In practice, such relations are often used in the context of databases (you might have already heard of the term "relational databases"): For example, we can model a person as an ordered tuple containing its name, age, address and phone number. A set of such 4-tuples is called 4-ary (*quaternary*) relation.

Up to this point, our definitions did not consider the actual connections contained in the relation: $\{(1,-1)\} \subseteq \mathbb{N} \times \mathbb{Z}$ connects 1 and $-1$ only, however the source set is not $\{1\}$ but $\mathbb{N}$ and the target set is not $\{-1\}$ but $\mathbb{Z}$. Next up, we define some general properties that actually depend on the elements of the relation.

**Definition 4.39** (Domain, Image). *Let $\mathcal{R} \subseteq A \times B$ be a relation on some sets A, B.*

*The* ***domain*** *of $\mathcal{R}$ $\operatorname{dom}(\mathcal{R})$ is defined as follows:*

$$\operatorname{dom}(\mathcal{R}) := \{a \in A \mid \exists b \in B : (a,b) \in \mathcal{R}\}$$

*The* ***image*** *of $\mathcal{R}$ (also called* ***range*** *of $\mathcal{R}$) $\operatorname{im}(\mathcal{R})$ is defined as follows:*

$$\operatorname{im}(\mathcal{R}) := \{b \in B \mid \exists a \in A : (a,b) \in \mathcal{R}\}$$

The domain of a relation is a (not necessarily proper) subset of the relation's source set. It contains all elements from which a connection to some element of the target set starts. Similarly, the image of a relation is the subset of its target set that contains all elements that some source set element connects to.

Make sure not to confuse *source set* with *domain*: The *source set* is part of a relation's definition, to be precise the set from which all connections originate, while the *domain* is the subset of the *source set* that only contains elements from which at least one connection originates.

**Example 4.40:** Binary Relations

Let's consider a group of students: Alice, Bob, Charly, Dieter and Emily. Mathematically, we can think of this group as a set $S$ containing every student (names abbreviated):

$$S = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}\}$$

Every student likes him-/herself (except explicitly said differently), but not every student likes every other student. Below is a list of all students and whom they like apart from themselves:

- Alice likes Bob and Dieter
- Bob likes Alice
- Charly likes Bob, Dieter and Emily
- Dieter likes Alice, Charly and Emily
- Emily likes nobody, not even herself

We can model this mathematically as a relation $\mathcal{R} = \{(x,y) \mid x \text{ likes } y\} \subseteq S^2$ on the set of students. This relation then contains the following elements:

$$\mathcal{R} = \{(\text{A},\text{A}), (\text{A},\text{B}), (\text{A},\text{D}), (\text{B},\text{B}), (\text{B},\text{A}), (\text{C},\text{C}), (\text{C},\text{B}), \\ (\text{C},\text{D}), (\text{C},\text{E}), (\text{D},\text{D}), (\text{D},\text{A}), (\text{D},\text{C}), (\text{D},\text{E})\}$$

As we defined $\mathcal{R}$ on the set $S$, $S$ is both the *source* and *target set* and therefore also the *universal set* of $\mathcal{R}$. The *image* of $\mathcal{R}$ is also equal to $S$, as every student is liked by at least one other student. Hence, $\mathcal{R}$ for each element of $S$ contains at least one connection ending at the respective element. Still, the *domain* is not equal to $S$ as there is no outgoing connection from Emily. Therefore, the domain is

$$\operatorname{dom}(\mathcal{R}) = S \setminus \{\text{E}\} = \{\text{A}, \text{B}, \text{C}, \text{D}\}$$

Having understood those definitions, we can take a look at four important relations:

**Definition 4.41** (Empty Relation)**.** *The relation* $\emptyset \subseteq A \times B$ *is called* ***empty relation****. It does not contain any pairs of source and target set elements, hence not connecting any source set elements to any target set elements. Therefore,* $\operatorname{dom}(\emptyset) = \operatorname{im}(\emptyset) = \emptyset$ *holds.*

**Definition 4.42** (Identity Relation)**.** *The relation* $\mathrm{Id}_A \coloneqq \{(x, x) \mid x \in A\} \subseteq A^2$ *on the universal set* $A$ *is called the* ***identity relation*** *on* $A$*. It connects each element in* $A$ *to itself. Therefore,* $\operatorname{dom}(\mathrm{Id}_A) = \operatorname{im}(\mathrm{Id}_A) = A$ *holds.*

**Definition 4.43** (Universal Relation)**.** *The relation* $\mathrm{U}_{A,B} \coloneqq A \times B \subseteq A \times B$ *is called the* ***universal relation*** *on* $A$ *and* $B$*. It connects each element from* $A$ *to each element in* $B$*. Therefore, both* $\operatorname{dom}(A \times B) = A$ *and* $\operatorname{im}(A \times B) = B$ *hold.*

**Definition 4.44** (Inverse Relation)**.** *The relation* $\mathcal{R}^{-1} \coloneqq \{(b, a) \mid (a, b) \in \mathcal{R}\} \subseteq B \times A$ *is called the* ***inverse relation*** *of* $\mathcal{R} \subseteq A \times B$*. It contains all connections from* $\mathcal{R}$ *with start and end swapped. Therefore, both* $\operatorname{dom}(\mathcal{R}^{-1}) = \operatorname{im}(\mathcal{R})$ *and* $\operatorname{im}(\mathcal{R}^{-1}) = \operatorname{dom}(\mathcal{R})$ *hold.*

**Example 4.45:** Important Relations

Let $X = \{1, 2, 3\}$ and $Y = \{a, b, c\}$.
The identity relation on $X$ is as follows:

$$\mathrm{Id}_X = \{(1, 1), (2, 2), (3, 3)\}$$

The universal relation on $X$ and $Y$ is as follows:

$$\mathrm{U}_{X,Y} = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c)\} = X \times Y$$

We can get the inverse of $\mathrm{U}_{X,Y}$ by swapping the components of every pair in the relation. This results in the following inverse relation:

$$\mathrm{U}_{X,Y}^{-1} = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2), (a, 3), (b, 3), (c, 3)\} = Y \times X = \mathrm{U}_{Y,X}$$

### 4.2.1 Notation

In this section, we take a look at different ways of writing down relations. For this, consider the following relation:

$$\mathcal{R} = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$$

This way of writing down a relation by explicitly naming every contained pair is called **enumerative notation**. We already saw this kind of notation in Section 4.1.1 where sets were denoted by explicitly naming every element. As a relation's elements are ordered pairs, we are just reapplying the same idea from set theory to the topic of relations.

Another notation we can carry over from set theory is the **predicate notation**. The relation $\mathcal{R}$ we just took a look at is the $\leq$ relation on the universal set $A = \{1, 2, 3\}$. Therefore, we can also denote $\mathcal{R}$ as follows:

$$\mathcal{R} = \left\{(x, y) \in A^2 \,\middle|\, x \leq y\right\}$$

One advantage of predicative over enumerative notation is that we are often able to write down large relations (those that contain many connections) compactly. If we consider the $\leq$ relation over $\mathbb{N}$, it is no longer possible to list all contained connections without abbreviations as the relation contains infinitely many connections.

Apart from notations that rely on the fact that relations are just a special kind of sets, there are also several relation-specific notations which can be helpful in characterizing relations. The first one we look at are **logical matrices**. These work by having a row for each source set element and a column for each target set element. At each intersection of a row and column, a 1 is placed if the pair of represented source and target set elements is contained in the relation. We can represent our example relation $\mathcal{R}$ like this:

$$\begin{array}{c|ccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{1} & 1 & 1 & 1 \\ \mathbf{2} & 0 & 1 & 1 \\ \mathbf{3} & 0 & 0 & 1 \end{array} \quad \text{or also commonly written as} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

If you want to see if a source set element $x$ is related to a target set element $y$, you can look for the row that represents $x$ and for the column that represents $y$. If and only if there is a 1 at the crossing, $x$ and $y$ are related.

A more space-efficient way of writing down relations is the **list notation**. It works by listing all source set elements followed by every target set element they relate to. This is how our example $\mathcal{R}$ looks in list notation:

$$\begin{array}{l} \mathbf{1}: \ 1 \ \ 2 \ \ 3 \\ \mathbf{2}: \ 2 \ \ 3 \\ \mathbf{3}: \ 3 \end{array}$$

Another way of visualizing relations are so-called **arrow diagrams**. These work by listing all source set elements vertically on the left side and all target set elements similarly on the right side. After doing so, we can draw arrows from each source set element to its related target set elements.

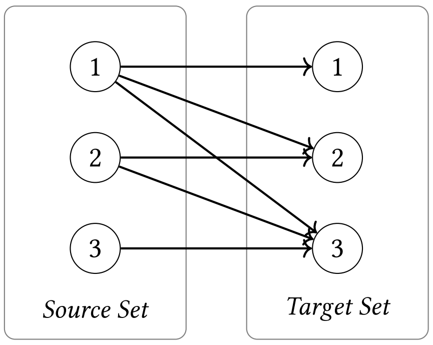


Figure 4.46: Arrow diagram for $\mathcal{R} = \{(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}$

You may notice that such arrow diagrams quickly get cluttered. If a relation is defined on an identical source and target set—therefore defined on a universal set—we can also represent it as a directed graph. A **graph** is a collection of **nodes** (also called **vertices**) and **edges**. Edges always connect two (not necessarily different) nodes. If for each edge it is defined on which side it starts and on which it ends, the graph is called **directed**.

A relation on a universal set can be represented as a directed graph by first drawing nodes for each element of the universal set. Afterwards, each pair contained in the relation is drawn as an arrow that starts at the first component of the pair and ends at the second component.

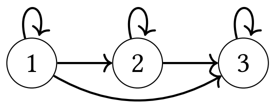
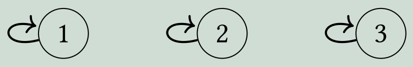


Figure 4.47: Directed graph representing $\mathcal{R} = \{(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}$

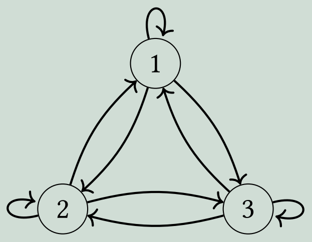
As it is usually clear that graphs in the context of relations are directed, from now on we only say “graphs” when referring to “directed graphs.”

**Example 4.48:** Graphs of Important Relations

Let again $X = \{1, 2, 3\}$. The graph of any identity relation only consists of arrows that start and end at the same node. Therefore, the graph of $\mathrm{Id}_X$ looks like this:



In the graph of any universal relation defined on a universal set, there are outgoing arrows from every node going towards every node (including the one where the arrow started). Therefore, the graph of $\mathrm{U}_{X,X}$ looks like this:



Remember that you can only draw a graph for a universal relation if the source and target set match.

**Checkpoint 4.49:** Relation Notations

- In Example 4.40, we defined a relation on a group of students. Which kinds of relation notations were used there? How else can you represent the relation?
- How does the graph of a relation change if the relation is inverted?

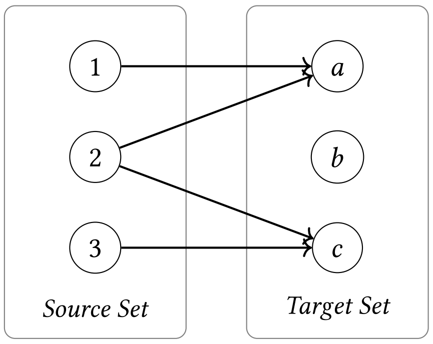
### 4.2.2 Common Properties

So far, we have only covered general properties of relations that can be applied to most, if not all, relations. In this section, we take a look at some properties of relations that can be used to classify different kinds of relations.

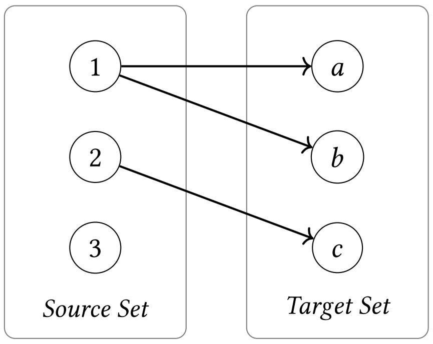
**Definition 4.50** (Left-Total/Serial Relation)**.**
*Let $\mathcal{R} \subseteq A \times B$ be a relation on arbitrary sets $A$ and $B$.*

*$\mathcal{R}$ is **left-total** or **serial** if $\forall a \in A : \exists b \in B : a\mathcal{R}b$. This is equivalent to* $\operatorname{dom}(\mathcal{R}) = A$.

If a relation is *left-total*, every source set element is connected to *at least* one element in the target set. Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *left-total*



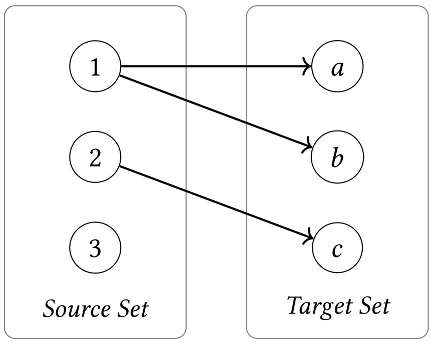
✗ *not left-total*

The relation depicted on the right is not left-total, as there is no connection from 3.

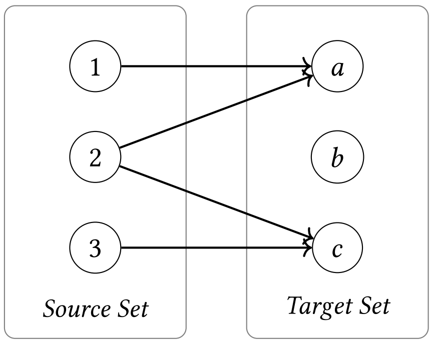
**Definition 4.51** (Right-Total/Surjective Relation)**.**
*Let $\mathcal{R} \subseteq A \times B$ be a relation on arbitrary sets $A$ and $B$.*

*$\mathcal{R}$ is **right-total** or **surjective** if $\forall b \in B : \exists a \in A : a\mathcal{R}b$. Then,* $\operatorname{im}(\mathcal{R}) = B$ *also holds.*

If a relation is *surjective*, every target set element is reached by *at least* one element in the source set. Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *surjective*

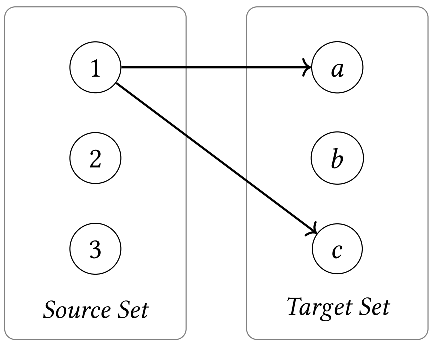


✗ *not surjective*

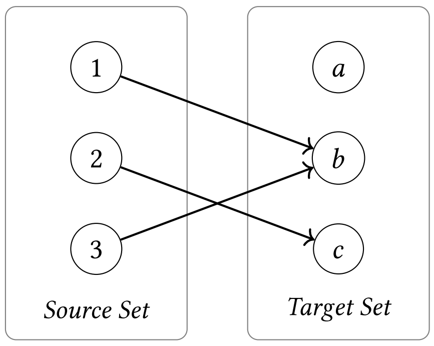
The relation depicted on the right is not surjective, as there is no connection to $b$.

> **Definition 4.52** (Left-Unique/Injective Relation).
> *Let $\mathcal{R} \subseteq A \times B$ be a relation on arbitrary sets $A$ and $B$.*
>
> *$\mathcal{R}$ is **left-unique** or **injective** if $\forall a_1, a_2 \in A : \forall b \in B : a_1\mathcal{R}b \land a_2\mathcal{R}b \rightarrow a_1 = a_2$.*

If a relation is *injective*, each target set element is connected to *at most* one element in the source set. Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *injective*

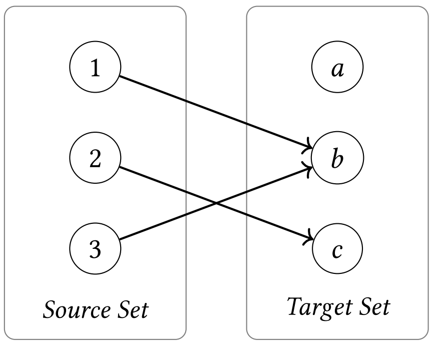


✗ *not injective*

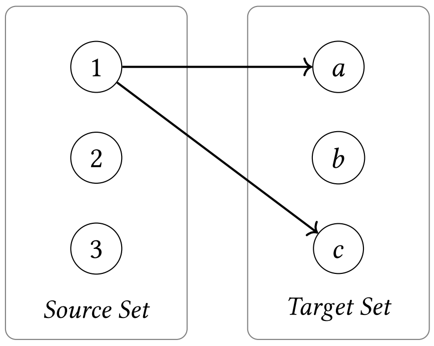
The relation depicted on the right is not injective, as there are two connections to $b$.

> **Definition 4.53** (Right-Unique/Functional Relation).
> *Let $\mathcal{R} \subseteq A \times B$ be a relation on arbitrary sets $A$ and $B$.*
>
> *$\mathcal{R}$ is **right-unique** or **functional** if $\forall b_1, b_2 \in B : \forall a \in A : a\mathcal{R}b_1 \land a\mathcal{R}b_2 \rightarrow b_1 = b_2$.*

If a relation is *functional*, each source set element is connected to at most one element in the target set. Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *functional*



✗ *not functional*

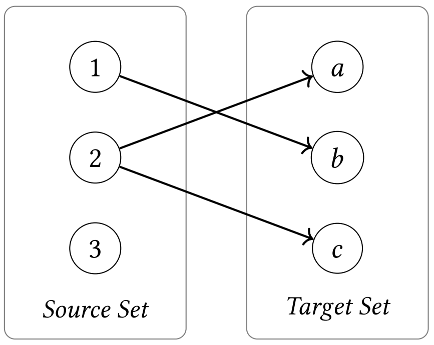
The relation depicted on the right is not functional as there are two outgoing connections from 1.

> **Definition 4.54** (Total and Partial Functions). *A functional relation $f \subseteq A \times B$ on two arbitrary sets $A$ and $B$ is called **function from** $A$ **to** $B$, abbreviated as $f : A \rightarrow B$. If it is left-total, it is called **total function**. Otherwise, it is called **partial function**.* For functions, *the source set is commonly referred to as "domain" and the target set is called "codomain."*
>
> *Note that in the case of partial functions, the source set and domain (the way we defined it in Definition 4.38) actually differ. Nonetheless, "domain" is often used the same way as for total functions if the relation is not mentioned explicitly.*

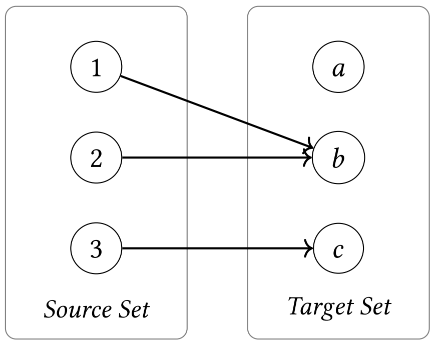
You probably already know functions from school where they were defined to map some parameter values to some other values. Mathematically speaking, functions are a special kind of relations that connect a parameter to a result value. Let's consider the function $f : \mathbb{N} \to \mathbb{N}$ which is defined as $f(x) = x + 1$. If we think of $f$ as a relation, it is a set that contains infinitely many pairs (for example $(1, 2) \in f$ and $(41, 42) \in f$). This relation is functional, as it maps each value for $x$ (the parameter) to exactly one result value (that is $x + 1$).

> **Definition 4.55** (Bijective Relation). *A relation $\mathcal{R}$ is* ***bijective*** *if it is both surjective and injective.*

If a relation is *bijective*, each target set element is reached by exactly one source set element. Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *bijective*



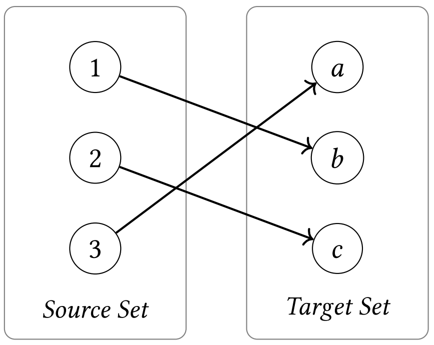
✗ *not bijective*

The relation depicted on the right is not bijective for two reasons: There is no source set element connecting to $a$, and there are two source set elements connecting to $b$. Both reasons would suffice on their own to disprove the right relation being bijective.

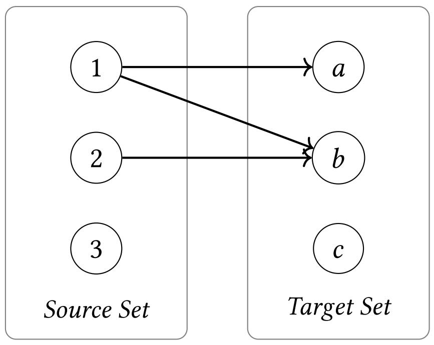
> **Definition 4.56** (Bijection). *A bijective total function is called* ***bijection****. Bijections are left-total, surjective, injective and functional.*

Be careful not to confuse *bijections* with *bijective relations*. The latter ones aren't necessarily left-total or functional but only surjective and injective. Bijections connect each element of the source set to exactly one element of the target set and vice-versa. Therefore, bijections are sometimes also called **one-to-one relations**.

Consider the following two relations defined on the source set $\{1, 2, 3\}$ and target set $\{a, b, c\}$:



✓ *a bijection*



✗ *not a bijection*

The relation depicted on the right is not a bijection for multiple reasons:

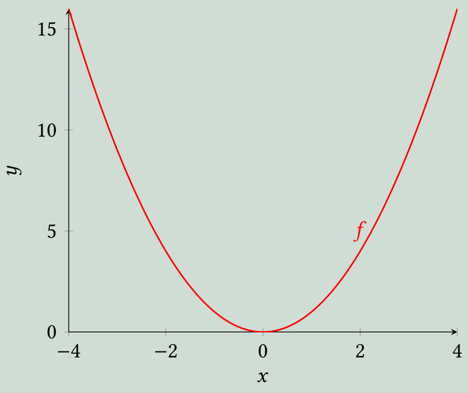
- It is not left-total, as 3 is not connected to any target set element.
- It is not surjective, as there is no source set element connected to $c$.
- It is not injective, as there are two different source set elements connected to $b$.
- It is not functional, as 1 is connected to two different target set elements.

Any one of those four reasons would suffice on their own to disprove the relation being a bijection.

Let's look at an example, where we prove some relation properties for a concrete relation.

**Example 4.57:** Proving Relation Properties

Let $f \subseteq \mathbb{R}\times\mathbb{R}$ be defined by $\{(x, x^2) \mid x \in \mathbb{R}\}$. You can see a plot of $f$ on the right.



We prove: $f$ is left-total.

*Proof.* Let $x \in \mathbb{R}$ be arbitrary, but fixed. We need to find a $y \in \mathbb{R}$ such that $(x, y) \in f$. We choose $y := x^2$, and are done since $(x, x^2) \in f$ by definition. Thus, $f$ is left-total. □

We prove: $f$ is not surjective.

*Proof by contradiction.* Assume $f$ is surjective. Then, in particular, there is $x$ such that $(x, -1) \in f$, which means that $x^2 = -1$. Because $\sqrt{-1}$ is not defined on the real number line, there is no $x \in \mathbb{R}$ so that $x^2 = -1$. This is a contradiction. Therefore, $f$ is not surjective. □

*Intuitively:* One can easily see that $f$ is not surjective as the function plot never reaches values below the $x$-axis.

We prove: $f$ is not injective.

*Proof by contradiction.* Assume $f$ is injective. Let $x_1 = 1$ and $x_2 = -1$. In particular, $x_1 \neq x_2$ Then $(x_1, 1) \in f$ and $(x_2, 1) \in f$. Since $f$ was assumed injective, we have $x_1 = x_2$, a contradiction. Thus $f$ is not injective. □

*Intuitively:* One can easily see that $f$ is not injective as the function plot reaches some values on the $y$-axis multiple times.

We prove: $f$ is functional.

*Proof.* Let $x \in \mathbb{R}$ be arbitrary, but fixed. We must show that for any two numbers $y_1, y_2$ such that $(x, y_1) \in f$ and $(x, y_2) \in f$, $y_1 = y_2$. By definition of $f$, we know that $y_1 = x^2 = y_2$. Therefore, $f$ is functional. □

**Checkpoint 4.58:** Bijections and Cardinality

Consider a bijection on two finite sets. Which property regarding the cardinality of the source set and target set does always hold?

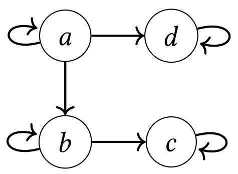
### 4.2.3 Properties of Relations on Universal Sets

Most relations computer scientists work with are defined on universal sets (meaning the source and target set are equal). In this section, we introduce several properties of such relations that allow us to further classify them. From now on, we no longer explicitly name relations defined on a universal set $A$ as such; instead, we say "relations on $A$" and refer to relations defined on different a source set $X$ and target set $Y$ as "relation on $X$ and $Y$."

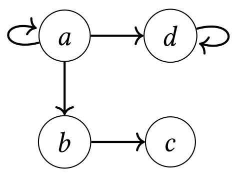
> **Definition 4.59** (Reflexive Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set $A$.*
>
> *$\mathcal{R}$ is **reflexive** if $\forall x \in A : x\mathcal{R}x$ holds.*
>
> *If a relation is reflexive, each element of $A$ is connected to itself.*

> **Definition 4.60** (Irreflexive Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set $A$.*
>
> *$\mathcal{R}$ is **irreflexive** if $\forall x \in A : (x, x) \notin \mathcal{R}$ holds.*
>
> *If a relation is irreflexive, no element of $A$ is connected to itself.*

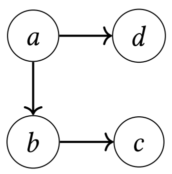
Consider the following three relations defined on $\{a, b, c, d\}$:



✓ *reflexive*
✗ *not irreflexive*



✗ *not reflexive*
✗ *not irreflexive*



✗ *not reflexive*
✓ *irreflexive*

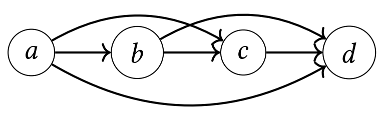
The relation depicted left is reflexive but not irreflexive as every element of $A$ is self-connected.

The relation in the middle is neither reflexive nor irreflexive, as only $a$ and $d$ but not $c$ and $d$ have reflexive edges. As you can see, reflexivity is not the opposite of irreflexivity.

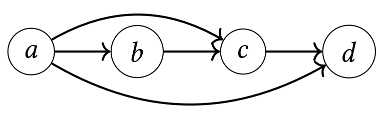
The relation shown on the right has no reflexive edges, therefore it is not reflexive but irreflexive.

> **Definition 4.61** (Transitive Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set $A$.*
>
> *$\mathcal{R}$ is **transitive** if $\forall x, y, z \in A : x\mathcal{R}y \land y\mathcal{R}z \rightarrow x\mathcal{R}z$ holds.*
>
> *A relation being transitive means that there is always a direct connection from $x$ to $z$ if a $y$ exists, which $x$ is connected to and which connects to $z$.*

Consider the following two relations defined on $\{a, b, c, d\}$:



✓ *transitive*



✗ *not transitive*

The relation depicted on the right is not transitive, as $b$ is connected to $c$ and $c$ is connected to $d$, but there is no direct connection from $b$ to $d$. Intuitively, transitivity allows you to get directly from one node to another if there is a path over another node to reach it.

**Definition 4.62** (Symmetric Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set A.*

*$\mathcal{R}$ is **symmetric** if $\forall x, y \in A : x\mathcal{R}y \rightarrow y\mathcal{R}x$ holds.*

*If a relation is symmetric, for each connection from one node to another there also exists a connection between the two nodes in the opposite direction.*

**Definition 4.63** (Antisymmetric Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set A.*

*$\mathcal{R}$ is **antisymmetric** if $\forall x, y \in A : x\mathcal{R}y \land y\mathcal{R}x \rightarrow x = y$ holds.*

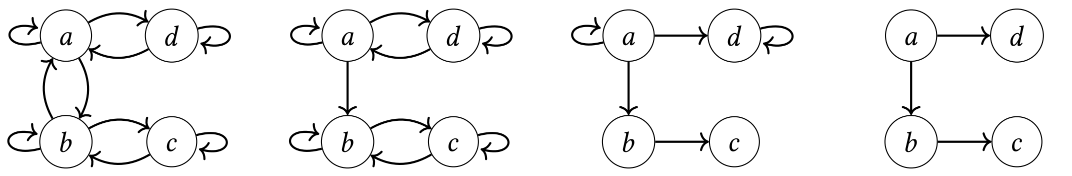
*If a relation is antisymmetric, for each connection from one node to another there is no connection between the two nodes in the opposite direction.*

**Definition 4.64** (Asymmetric Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set A.*

*$\mathcal{R}$ is **asymmetric** if $\forall (x, y) \in \mathcal{R} : (y, x) \notin \mathcal{R}$ holds.*

*If a relation is asymmetric, for each connection from one node to another there is no connection between the two nodes in the opposite direction. Additionally, there also are no reflexive edges.*

Consider the following four relations defined on $\{a, b, c, d\}$:



| | | | |
|---|---|---|---|
| ✓ *symmetric* | ✗ *not symmetric* | ✗ *not symmetric* | ✗ *symmetric* |
| ✗ *not antisymmetric* | ✗ *not antisymmetric* | ✓ *antisymmetric* | ✓ *antisymmetric* |
| ✗ *not asymmetric* | ✗ *not asymmetric* | ✗ *not asymmetric* | ✓ *asymmetric* |

The first relation is symmetric as for each edge there exists an edge going in the opposite direction.

The second relation is not symmetric, as there is an edge from $a$ to $b$ but none from $b$ to $a$. It also isn't antisymmetric or asymmetric as there are symmetric edges $(b, c)$ and $(c, b)$. As shown in this example, symmetry, antisymmetry and asymmetry are not mutual opposites.

The third relation is antisymmetric but not asymmetric as there are no symmetric edges between different nodes but reflexive nodes at $a$ and $d$.

The fourth relation is both antisymmetric and asymmetric as no reflexive or symmetric edges exist. Note that every asymmetric relation is also antisymmetric.

**Definition 4.65** (Connected Relation). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set A.*

*$\mathcal{R}$ is **connected** or **total** if $\forall x, y \in A : x \neq y \rightarrow x\mathcal{R}y \lor y\mathcal{R}x$ holds.*

*$\mathcal{R}$ is **strongly connected** if $\forall x, y \in A : x\mathcal{R}y \lor y\mathcal{R}x$ holds.*

*In a connected relation, each element of A is connected with each other element of A in at least one direction. If a relation is strongly connected, each element of A is additionally connected to itself. Therefore, strongly connected relations are always reflexive.*

**Example 4.66:** Properties of $=$, $\neq$, $<$ and $\leq$ Relations

In this example, we take a look at the $=$, $\neq$, $<$ and $\leq$ relations on $\mathbb{N}$.

$=$ reflexive, as $x = x$ holds for every $x \in \mathbb{N}$
transitive, as from $x = y$ and $y = z$ it follows that $x = z$
symmetric, as from $x = y$ we can follow $y = x$
not connected, as neither $1 = 2$ nor $2 = 1$ hold

$\neq$ irreflexive, as $x \neq x$ holds for no $x \in \mathbb{N}$
not transitive, as from $x \neq y$ and $y \neq z$ does not imply $x \neq z$ ($x = z$ is possible)
symmetric, as from $x \neq y$ we can follow $y \neq x$
connected, as for any distinct $x$ and $y$ it holds that $x \neq y$
not strongly connected, as $x = x$ and $x \neq x$ can never be true at the same time

$<$ irreflexive, as $x < x$ holds for no $x \in \mathbb{N}$
transitive, as $x < y$ and $y < z$ implies $x < z$
asymmetric, as $x < y$ and $y < x$ can never be true at the same time
connected, as $x \neq y$ implies $x < y$ or $y < x$
not strongly connected, as $x = x$ and $x < x$ can never be true at the same time

$\leq$ reflexive, as $x \leq x$ holds for every $x \in \mathbb{N}$
transitive, as $x \leq y$ and $y \leq z$ implies $x \leq z$
antisymmetric, as $x \leq y$ and $y \leq x$ can only be true at the same time if $x = y$
connected, as $x \neq y$ implies $x \leq y$ or $y \leq x$
strongly connected, as $x = x$ implies $x \leq x$

**Checkpoint 4.67:** All Satisfied

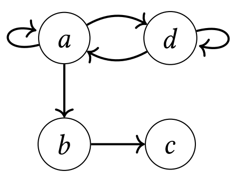
There is a relation which fulfills all the properties introduced in this section. Which one is it? For all other relations, which properties can never occur at the same time?

**Definition 4.68** (Relation Closures). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set $A$.*

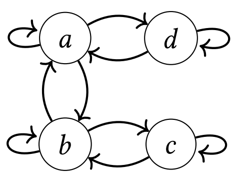
*The* ***reflexive closure*** *of $\mathcal{R}$ is the smallest reflexive relation $\mathcal{R}'$ for which $\mathcal{R} \subseteq \mathcal{R}'$ holds. The* ***symmetric*** *and* ***transitive closures*** *are defined analogously.*

*The closure $\mathcal{R}'$ of a relation $\mathcal{R}$ regarding one or more properties is the smallest superset of $\mathcal{R}$ which contains all necessary connections to fulfill the considered properties.*

Consider the following relations defined on $\{a, b, c, d\}$:



✗ *not reflexive*
✗ *not symmetric*



✓ *reflexive*
✓ *symmetric*

The relation on the right is the reflexive-symmetric closure of the left relation. This means that we have added the least amount of edges to make the left relation become reflexive and symmetric.

**Definition 4.69** (Composition). *Let $A, B, C$ be sets, and $\mathcal{R}_1 \subseteq B \times C$ and $\mathcal{R}_2 \subseteq A \times B$ two relations. The **composition** $\mathcal{R}_1 \circ \mathcal{R}_2$ of two relations $\mathcal{R}_1$ and $\mathcal{R}_2$ is defined as follows:*

$$\mathcal{R}_1 \circ \mathcal{R}_2 := \{(a, c) \in A \times C \mid \exists b \in B : a\mathcal{R}_2 b \wedge b\mathcal{R}_1 c\}$$

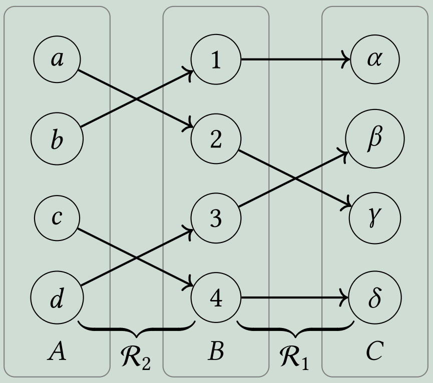
*The composition of two functions $f : B \to C$ and $g : A \to B$ is usually written as follows:*

$$(f \circ g)(x) := f(g(x))$$

Be careful when using compositions: They are not always defined in the same way. In some contexts, the order in which the functions or relations are written is swapped.

**Example 4.70:** Relation Composition

Let $A = \{a, b, c, d\}$, $B = \{1, 2, 3, 4\}$ and $C = \{\alpha, \beta, \gamma, \delta\}$ be sets. Additionally, let $\mathcal{R}_1 = \{(1, \alpha), (2, \gamma), (3, \beta), (4, \delta)\} \subseteq B \times C$ and $\mathcal{R}_2 = \{(a, 2), (b, 1), (c, 4), (d, 3)\} \subseteq A \times B$ be relations on those sets. We visualize $\mathcal{R}_1$ and $\mathcal{R}_2$ in the following arrow diagram:



Intuitively, we can get the composition $\mathcal{R}_1 \circ \mathcal{R}_2$ by tracing the outgoing arrows from each element in $A$ to see which elements in $C$ can be reached. For example, from $a$ we can reach $\gamma$ by going over 2. By tracing all arrows that start at some element of $A$, we can see that the composition is $\mathcal{R}_1 \circ \mathcal{R}_2 = \{(a, \gamma), (b, \alpha), (c, \delta), (d, \beta)\}$.

**Definition 4.71** (Alternative Definitions of Relation Properties). *Let $\mathcal{R} \subseteq A^2$ be a relation defined on some set $A$.*

- *$\mathcal{R}$ is reflexive iff $\mathrm{Id}_A \subseteq \mathcal{R}$.*
- *$\mathcal{R}$ is irreflexive iff $\mathcal{R} \cap \mathrm{Id}_A = \emptyset$.*
- *$\mathcal{R}$ is transitive iff $\mathcal{R} \circ \mathcal{R} \subseteq \mathcal{R}$.*
- *$\mathcal{R}$ is symmetric iff $\mathcal{R}^{-1} \subseteq \mathcal{R}$.*
- *$\mathcal{R}$ is antisymmetric iff $\mathcal{R} \cap \mathcal{R}^{-1} \subseteq \mathrm{Id}_A$.*
- *$\mathcal{R}$ is asymmetric iff $\mathcal{R} \cap \mathcal{R}^{-1} = \emptyset$.*

**Checkpoint 4.72:** Alternative Definitions

Take a moment to think about each of the alternative definitions seen in Definition 4.71. Why are these equivalent to the definitions we introduced before?

### 4.2.4 Equivalence and Order Relations

Over the last sections, we introduced the concept of relations as well as several properties that allow us to classify them. In this section, we use these properties to construct so-called equivalence relations.

> **Definition 4.73** (Equivalence Relation). *A binary relation $\mathcal{R}$ on some set $A$ is called* ***equivalence relation*** *if it is reflexive, symmetric and transitive. Two elements $x$ and $y$ are $\mathcal{R}$-equivalent if $x\mathcal{R}y$.*
>
> *The set $[a] := \{x \in A \mid a\mathcal{R}x\}$ is called an* ***equivalence class*** *with* ***representative*** *$a$.*

**In Other Words:** Equivalence Relations

*Equivalence relations* allow us to split up a set of objects into different classes by certain shared properties. As the name suggests, this kind of relation groups equivalent objects and separates them from other groups. Those groups are called *equivalence classes*. Within those groups, all elements are related to all elements (including themselves).
The reflexive property of the relation ensures that each element is equivalent to itself. Additionally, the relation is symmetric to ensure that if an element $x$ is equivalent to another element $y$, then $y$ is also equivalent to $x$. The transitive property ensures that if an element $x$ is equivalent to an element $y$ and $y$ is equivalent to an element $z$, that $x$ is also equivalent to $z$.

You probably already worked with multiple equivalence relations without even noticing. The best-known equivalence relation is $=$, which can be used to relate numbers of the same value. For example, we can relate $0.5$, $\frac{1}{2}$, $\frac{2}{4}$ and $\frac{3}{6}$ as they are all equivalent regarding the $=$ relation. Because $0.5 = \frac{1}{2} = \frac{2}{4} = \frac{3}{6}$ holds, we can follow that all the listed numbers are part of the same equivalence class $\left[\frac{1}{2}\right]$.

Other examples of equivalence relations include the logical equivalence $\equiv$ on the set of logical expressions, and the relation "Person $x$ was born in the same year as person $y$." on a set of persons.

If we stick to equivalence relations on the set of natural numbers, the remainder of a division by some number $m$ might come to mind. This equivalence is called "modular congruence."

> **Definition 4.74** (Modular Congruence). *Let $a$, $b$ and $n > 1$ be integer values.*
> *$a$ and $b$ are* ***congruent modulo*** *$n$ if the remainder of integer division by $n$ is equal for $a$ and $b$. Then, $n$ is called the* ***modulus*** *and the congruence is written as $a \equiv b \pmod{n}$. The resulting equivalence classes are called* ***residue classes****.*

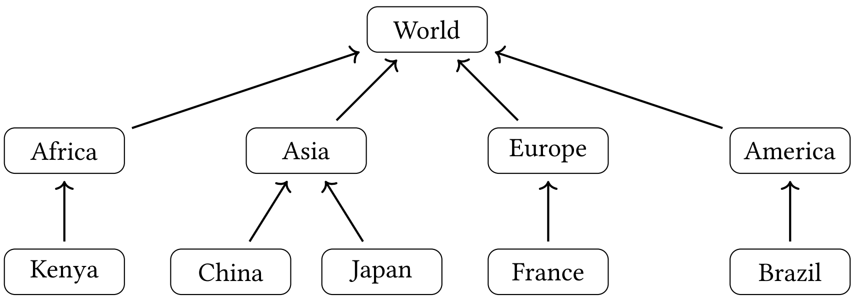
**Example 4.75:** Modular Congruence

Let's consider the modulus 3. It splits the natural numbers up into three residue classes: One for numbers that are divisible by 3, one for those numbers that can be divided with remainder 1, and one for numbers with division remainder 2. The most obvious representatives for those classes are 0, 1 and 2. Still, each of the residue classes can be represented by any number that is contained in it. Therefore, we can name the class for numbers that are divisible by 3 $[0], [3], [6], [42], \ldots$.

Now that we know of a way to express the equivalence or equality of mathematical objects using relation, let's think of a way to express comparisons between different objects. This allows us to say that a mathematical object is "greater than" or "less than" another object. "Greater" or "less" can, depending on the context, have other meanings than expected: Take for example the relation "$x$ is geographically contained within $y$" on the following set of regions:

$$L = \{\text{World, Africa, Asia, Europe, America, Brazil, China, France, Japan, Kenya}\}$$

We can visualize this relation in a graph:



Notice that the transitive and reflexive edges are missing in this graph even though every country lies within the world and every region is also geographically contained in itself. We left them out as those would make reading the graph significantly harder.

By our definition, the relation is antisymmetric, reflexive and transitive. Those properties can be used to order all regions: For example, we can say that France is contained in Europe but not the other way around. Therefore, we could say that Europe is greater than France in terms of this specific relation.

> **Definition 4.76** (Order Relation).
> *If a relation $\mathcal{R} \subseteq A^2$ is antisymmetric, transitive and reflexive, it is called an* ***order relation*** *on $A$.*
>
> *If $\mathcal{R}$ is irreflexive instead of reflexive, it is called a* ***strict order relation*** *on $A$.*

Two more commonly known order relations are $\leq$ and $<$ on $\mathbb{N}$. The latter is a strict order relation, as it is irreflexive because no number is less than itself.

> **Definition 4.77** (Partial and Linear Order). *An order relation that is connected is called* ***linear order*** *or* ***total order*** *while a non-connected order relation is called* ***partial order****. If a linear order is irreflexive instead of reflexive, it is also called* ***strict linear order*** *or* ***strict total order****.*

In contrast to our relation on geographic regions, the $\leq$ and $<$ on $\mathbb{N}$ relations are also connected as every number is related to all other numbers in one direction. This allows us to order all numbers in one line as can be seen below:

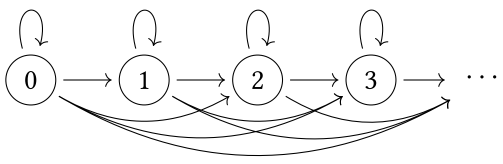


Figure 4.78: $\leq$ relation on $\mathbb{N}$

You might notice that there is no edge pointing to 0 except for the reflexive one. Instead, 0 has outgoing edges to every other element in $\mathbb{N}$. Intuitively, we say 0 is the smallest element or minimum of $\mathbb{N}$ as it is smaller than every other element of $\mathbb{N}$.

> **Definition 4.79** (Minimum and Minimal Element)**.**
> *Let $\mathcal{R} \subseteq A^2$ be an order on $A$ and $A' \subseteq A$ a non-empty subset of $A$.*
>
> *An element $y \in A'$ is called **minimal element** in $A'$ if $\forall x \in A' : x\mathcal{R}y \to x = y$.*
>
> *An element $x \in A'$ is called **minimum** of $A'$ if $\forall y \in A' : x\mathcal{R}y$.*

**In Other Words:** Minimal Elements and Minima

Let $\mathcal{R} \subseteq A^2$ be an order on $A$.
You can think of *minimal elements* as all elements of $A$ that have no incoming edges except for reflexive ones. *Minima* (singular: minimum) are a special kind of minimal elements and have the additional property that they have outgoing edges to all other elements.
Minima can only exist on linear orders which by their linear nature never have more than one minimal element. If there exist multiple minimal elements, none of them can be a minimum because that minimum would then have to have outgoing edges to the other minimal elements (which lets them lose their minimal property).

Recall the relation on geographic regions we introduced in this section. We can find multiple minimal elements (the countries) which have no incoming edges except for reflexive ones. Still, there is no minimum as none of the minimal elements is related to all other regions (e.g. China and Japan are not related at all).

**Checkpoint 4.80:** Max

How would you define a maximum and maximal elements for order relations?

To finish up this section on order relations, let's consider the $\leq$ relation on $\mathbb{Z}$. It does not have a minimum as it is not limited by any smallest number: For every integer, you can find another even smaller integer.

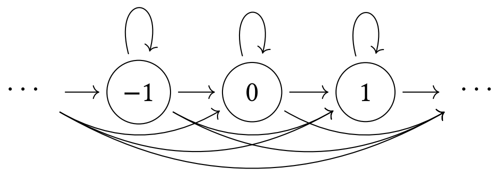


Figure 4.81: $\leq$ relation on $\mathbb{Z}$

> **Definition 4.82** (Well-Founded and Well-Ordered Sets)**.** *Let $\mathcal{R} \subseteq A^2$ be an order on $A$.*
>
> *$\mathcal{R}$ is **well-founded** if for any non-empty subset of $A$ there exists a minimal element. A well-founded total order is also called **well-order**.*

Intuitively, an order relation is well-founded if there is no infinitely descending chain. If you think of the graph of a well-founded relation, you can start at any node and walk over every incoming edge back until you at some point hit a minimal element. This also means that any incoming path leading to some element in a well-founded relation only consists of a finite number of edges.

### 4.2.5 Cardinality of Infinite Sets

In the chapter on sets, we already defined the cardinality of finite sets to be the number of elements contained within them. Next up, we discuss the cardinality of infinite sets. Take the set of natural numbers $\mathbb{N}$ for example. Obviously, $\mathbb{N}$ consists of infinitely many elements, therefore one might assume $|\mathbb{N}| = \infty$. The same seems to hold for the set of real numbers $\mathbb{R}$: $|\mathbb{R}| = \infty$. Don't let this fool you into believing that these equations actually hold[2] or that $\mathbb{N}$ and $\mathbb{R}$ are equal in size. In this section, we see why both sets have different sizes even though both contain infinitely many elements. To prove this, we first have to define what it means for two infinite sets to be "equal in size"—or, more formally, "equinumerous."

> **Definition 4.83** (Equinumerous Sets)**.**
> *Two sets $A$ and $B$ are* ***equinumerous*** *iff a bijection $f : A \to B$ exists.*

**In Other Words:** Bijections and Cardinality

Recall the definition of bijections: A bijection is a relation which connects each element of its source set to exactly one element of its target set and vice-versa. We can say that two sets $A$ and $B$ are equal in size if we can find such a one-to-one relation that maps each element of $A$ to exactly one element of $B$ and the other way around.
This works both for finite and infinite sets. If $A$ and $B$ are finite, finding a bijection is trivial as we can just define orders on both sets and then construct a relation which maps the first element of $A$ to the first element of $B$, the second element of $A$ to the second element of $B$, and so on for every pair of elements.

We can use this definition to show that the set of natural numbers $\mathbb{N}$ and the set of integers $\mathbb{Z}$ are equal in size. To do so, we have to find a bijection $f : \mathbb{N} \to \mathbb{Z}$ mapping each integer uniquely to one natural number.

**Theorem 4.84.** *$\mathbb{N}$ and $\mathbb{Z}$ are equinumerous, meaning a bijection $f : \mathbb{N} \to \mathbb{Z}$ exists.*

*Proof.* Let $f : \mathbb{N} \to \mathbb{Z}$ be defined as follows:

$$f(x) = \begin{cases} \frac{x}{2}, & \text{if } x \text{ is even} \\ -\frac{x+1}{2}, & \text{if } x \text{ is odd} \end{cases}$$

$f$ then yields the following results:

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | $\cdots$ |
|---|---|---|---|---|---|---|---|---|
| $f(x)$ | 0 | $-1$ | 1 | $-2$ | 2 | $-3$ | 3 | $\cdots$ |

We can also represent the results of $f$ as a spiral:

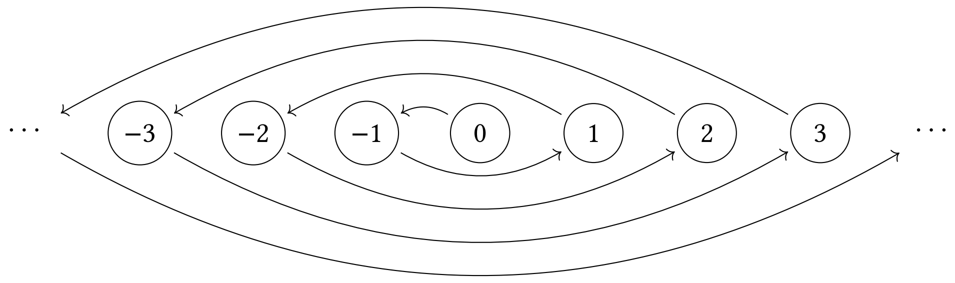


Figure 4.85: Graphical representation of results of $f$

[2] There is no clear definition on what $\infty$ is supposed to mean. Therefore, the equation $\infty = \infty$ doesn't make sense.

We show that $\mathbb{N}$ and $\mathbb{Z}$ are equinumerous by proving that $f$ is a bijection on $\mathbb{N}$ and $\mathbb{Z}$. This can be done by proving $f$ is left-total, surjective, injective and functional.

*We show:* $f$ is left-total, so $\forall a \in \mathbb{N} : \exists b \in \mathbb{Z} : f(a) = b$.

Let $a \in \mathbb{N}$ be arbitrary, but fixed. We differentiate between two cases:

- Case 1: *$a$ is even.*
  Then $\exists k \in \mathbb{N} : a = 2k$ and $f(a) = \frac{a}{2} = \frac{2k}{2} = k$. Therefore, $f(a) \in \mathbb{Z}$.
- Case 2: *$a$ is odd.*
  Then $\exists k \in \mathbb{N} : a = 2k + 1$ and $f(a) = -\frac{a-1}{2} = -\frac{2k+1-1}{2} = -k$. Therefore, $f(a) \in \mathbb{Z}$.

*We show:* $f$ is surjective, so $\forall b \in \mathbb{Z} : \exists a \in \mathbb{N} : f(a) = b$.

Let $b \in \mathbb{Z}$ be arbitrary, but fixed. We differentiate between two cases:

- Case 1: $b \geq 0$.
  Let $a = 2b$. Then $a$ is even and $f(a) = \frac{a}{2} = \frac{2b}{2} = b$.
- Case 2: $b < 0$.
  Let $a = -2b - 1$. Then $a$ is odd and $f(a) = -\frac{a+1}{2} = -\frac{-2b-1+1}{2} = b$.

*We show:* $f$ is injective, so $\forall b \in \mathbb{Z} : \forall a_1, a_2 \in \mathbb{N} : f(a_1) = b \land f(a_2) = b \rightarrow a_1 = a_2$.

Let $b \in \mathbb{Z}$ and $a_1, a_2 \in \mathbb{N}$ be arbitrary, but fixed with $f(a_1) = b$ and $f(a_2) = b$. We differentiate between four cases:

- Case 1: *$a_1$ and $a_2$ are even.*
  Then $f(a_1) = \frac{a_1}{2} = b = \frac{a_2}{2} = f(a_2)$ and by multiplying 2: $a_1 = a_2$.
- Case 2: *$a_1$ and $a_2$ are odd.*
  Then $f(a_1) = -\frac{a_1+1}{2} = b = -\frac{a_2+1}{2} = f(a_2)$ and by multiplying 2 and subtracting 1: $a_1 = a_2$.
- Case 3: *$a_1$ is even and $a_2$ is odd.*
  Then $f(a_1) = \frac{a_1}{2} = b = -\frac{a_2+1}{2} = f(a_2)$ and by multiplying 2 and adding 1: $a_1 + 1 = -a_2$. As $a_1 \in \mathbb{N}$, $a_1 + 1 > 0$ has to hold and therefore $a_2 < 0$ and $a_2 \notin \mathbb{N}$ $\lightning$. Hence, this cannot occur.
- Case 4: *$a_1$ is odd and $a_2$ is even.*
  Analogous to the previous case, this cannot occur.

*We show:* $f$ is functional, so $\forall a \in \mathbb{N} : \forall b_1, b_2 \in \mathbb{Z} : f(a) = b_1 \land f(a) = b_2 \rightarrow b_1 = b_2$.

Let $a \in \mathbb{N}$ and $b_1, b_2 \in \mathbb{Z}$ be arbitrary, but fixed with $f(a) = b_1$ and $f(a) = b_2$. We differentiate between two cases:

- Case 1: *$a$ is even.*
  Then $b_1 = f(a) = \frac{a}{2} = f(a) = b_2$.
- Case 2: *$a$ is odd.*
  Then $b_1 = f(a) = -\frac{a-1}{2} = f(a) = b_2$.

By proving all bijection properties of $f$ we have shown that $\mathbb{N}$ and $\mathbb{Z}$ are equinumerous. $\square$

Now that we have shown that $\mathbb{N}$ and $\mathbb{Z}$ are equinumerous, the question arises if there are other infinite sets that are also equinumerous to $\mathbb{N}$. The answer to this question is yes, as, for example, the set of even and the set of uneven numbers are both also equinumerous to $\mathbb{N}$.

> **Definition 4.86** (Countable Set). *A set is* ***countable*** *if it is finite or equinumerous to* $\mathbb{N}$*. If a countable set is infinite, it is also called* ***countably infinite set****. An infinite set that is not countable is called* ***uncountably infinite set****.*

Intuitively, each infinite set for which it is possible to uniquely assign a natural number to every element is countable. In contrast to these countable sets, there also are uncountably infinite sets for which this is not possible. A prominent example for such uncountable sets is the set of real numbers $\mathbb{R}$.

**Going Beyond:** Cantor's Diagonal Argument

We can prove that $\mathbb{R}$ is uncountable by using Cantor's diagonal argument. For this, we only consider real numbers in the interval $(0, 1)$ (those that are $> 0$ and $< 1$). Showing that the set of these numbers already is uncountable is enough to prove that $\mathbb{R}$ is uncountable. Cantor's diagonal argument proves that no bijection $f : \mathbb{N} \longrightarrow (0, 1)$ can exist. Assume any function $f : \mathbb{N} \longrightarrow (0, 1)$. We can write its results in a list like this:

$$\begin{array}{ccccccccc}
f(1) & = & 0. & \underline{a_{1,1}} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots \\
f(2) & = & 0. & a_{2,1} & \underline{a_{2,2}} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots \\
f(3) & = & 0. & a_{3,1} & a_{3,2} & \underline{a_{3,3}} & a_{3,4} & a_{3,5} & \cdots \\
f(4) & = & 0. & a_{4,1} & a_{4,2} & a_{4,3} & \underline{a_{4,4}} & a_{4,5} & \cdots \\
f(5) & = & 0. & a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & \underline{a_{5,5}} & \cdots \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Here, $a_{m,n}$ represents the digit at the $n$-th decimal place of $f(m)$. If we take all digits along the main diagonal (underlined in the visualization) and change them to some other digit (for example by adding 1 and overflowing to 0 from $9 + 1$), we can construct a new number which is contained in $(0, 1)$ but is never reached by $f$ because each for each number reached by $f$, one digit was changed to no longer match it. Therefore, $f$ cannot be surjective. This leads to the conclusion that the set of real numbers is uncountable.

Before ending this chapter, we still have one proof left:

**Theorem 4.87** (Cantor)**.** *The cardinality of an arbitrary set $A$ differs from the one of its power set $\mathcal{P}(A)$, that is*

$$|A| \neq |\mathcal{P}(A)|.$$

*Proof by contradiction.* Let $A$ be an arbitrary set. We assume that the cardinality of $A$ is equal to the one of $\mathcal{P}(A)$. Therefore, a bijection $f : A \to \mathcal{P}(A)$ exists.

We define $\mathcal{S} := \{x \in A \mid x \notin f(x)\}$. As $\mathcal{S}$ only consists of elements of $A$, both $\mathcal{S} \subseteq A$ and therefore also $\mathcal{S} \in \mathcal{P}(A)$ hold.

Since $f$ is surjective and $\mathcal{S} \in \mathcal{P}(A)$, an $a \in A$ has to exist so that $f(a) = \mathcal{S}$.

We differentiate two cases:

- Case 1: $a \in f(a)$. Then by definition $a \notin \mathcal{S} = f(a)$ ↯.
- Case 2: $a \notin f(a)$. Then by definition $a \in \mathcal{S} = f(a)$ ↯.

Since both cases lead to a contradiction, our assumption has to be wrong. Therefore, a bijection $f : A \to \mathcal{P}(A)$ cannot exist, so $A$ and $\mathcal{P}(A)$ are not equinumerous. □

**In Other Words:** Cantor's Proof

The proof you have just seen is pretty abstract, so there is no need to worry if you did not understand it the first time you read it. It works by showing that no bijection $f$ between any set $A$ and its power set $\mathcal{P}(A)$ can exist.
To show this, we define a set $\mathcal{S}$ that contains all elements of $A$ that do not appear in the result of $f$ when applied to that specific element. Consider for example the set $A = \{1, 2, 3, 4\}$ and assume the following mappings for $f$: $f(1) = \{1, 2\}$ and $f(2) = \{3\}$. In this specific example, 1 does not appear in $\mathcal{S}$ because it is contained in $f(1) = \{1, 2\}$, but 2 does appear in $\mathcal{S}$ as it is not contained in $f(2) = \{3\}$.
We then use the fact that every bijection is surjective, meaning that there has to exist an input $a \in A$ for $f$ such that it returns $\mathcal{S}$. This leads to a contradiction: If $a$ is contained in $f(a)$, then by definition of $\mathcal{S}$, $a \notin \mathcal{S}$ and if $a$ is not contained in $f(a)$, then $a \in \mathcal{S}$. Thus, no $a$ can exists for which $f(a) = \mathcal{S}$.
This proves that our initial assumption that a bijection between $A$ and $\mathcal{P}(A)$ exists is wrong. Therefore, $A$ and $\mathcal{P}(A)$ have to have different cardinalities.

## 4.3 Problems With Naive Set Theory

To conclude this chapter, we would like to take a look at some problems posed by the very first definition we gave for sets. As previously stated, we use Cantor's naive definition of a set which, if you remember Definition 4.1, was only loosely defined as a collection of some distinct objects.

We also saw quite early that those objects can also be sets themselves, but we did not think further about this. However, this fact poses some interesting problems with important consequences which we want to investigate now along with some short insights into the history of set theory.

In the 19th century, sets were first used by the German mathematicians Gottlob Frege and Richard Dedekind in a similar manner to what we have seen with Cantor, but not "formalized". Cantor published his definition in 1895 in his works "Beiträge zur Begründung der transfiniten Mengenlehre" (contributions to the founding of the theory of transfinite numbers).

In 1901, British philosopher and mathematician Bertrand Russel discovered a severe problem which later became known as **Russel's paradox** or **Russel's antinomy**. He asked the following question:

Is there a set
that contains all sets
that do not contain themselves?

We can also formulate this question as a statement in our language of propositional logic:

$$\exists A : A = \{B \mid B \notin B\}?$$

You might ask yourself now: why should this be a paradox? Why should this set not exist? Until now, we just wrote down any sets with any predicates and it just worked out fine. But here, things are different.

We have to differentiate between two cases here, namely whether $A$ does contain itself. Let's look at them both and see where exactly the problem lies.

- First, let's assume that, in fact, $A$ *contains itself* or formally $A \in A$.

  What are the consequences? If $A \in A$, we can follow $A \notin A$ ↯, since that is the very definition of $A$, that is, the predicate all elements of $A$ fulfill. But this right there is a contradiction as from Definition 4.3 we know that either $A \in A$ or $A \notin A$.

  So we can conclude that, if $A \in A$, $A$ cannot exist.

- Now we take a look at the alternative, namely $A$ *does not contain itself*, for short $A \notin A$.

  Our conclusion goes similarly to the above case: if $A$ does not contain itself, it is by definition an element of $A$, meaning $A \in A$ ↯.

  Again, because of the contradiction, the set $A$ cannot exist if $A \notin A$.

As either $A \in A$ or $A \notin A$ and in both cases we come to a contradiction, the set $A$ cannot exist. To reiterate, if it would exist, we were able to derive $A \in A \leftrightarrow A \notin A$ with the above reasoning which is a false statement. Therefore, we have found a concrete set that we can write down but that cannot exist. That is the core of the paradox.

**Going Beyond:** Cantor's Antinomies

Cantor himself has also found two more paradoxes (or antinomies) in his set theory and they are therefore called Cantor's first and second antinomies. The first one involves cardinal numbers, a mathematical concept beyond what we are going to cover here. However, the result is quite similar, as he found that the set containing all cardinal numbers is, in fact, not a set.
Cantor's second antinomy concerns the set that contains everything, particularly all sets. We can write this out as

$$U := \{X \mid \top\}.$$

But why can't this be a set? This is hard to prove without the necessary prerequisites. However, we would still like to give you an intuition and therefore line out a proof concept. We need one important result that is named after Cantor: Cantor's theorem. Fortunately, we have already seen something that is almost the theorem we need, namely Theorem 4.87. The statement can actually be strengthened to

$$|A| < |\mathcal{P}(A)|.$$

Now, if there were said set $U$, then $\mathcal{P}(U) \in U$, as $U$ contains all sets. Additionally, and for the same reason, $U$ contains all $P \in \mathcal{P}(U)$. So we come to the conclusion that $U$ contains all elements of $\mathcal{P}(U)$ plus $\mathcal{P}(U)$ itself, meaning

$$|U| \geq |\mathcal{P}(U)|. \lightning$$

This is a contradiction to Cantor's theorem, which is why our assumption, that the set of all sets $U$ exists, is false.

The core construct that enabled the previous paradox was *unrestricted quantification*, which describes the concept of just taking any property (like $x \notin x$) and constructing the set of all objects fulfilling this property. Russel's paradox showed that this construction principle leads to a contradiction.

After these paradoxes were discovered, mathematicians began to come up with axiomatized set theories, meaning that they gave a few axioms for how sets must behave, and then built their set theories on top of them. There are a few different theories around today. One of them is the Zermelo–Fraenkel set theory (ZF for short), named after mathematicians Ernst Zermelo and Abraham Fraenkel. This set theory has several different construction principles, which allow us to do almost all of the operations we want to do, like constructing power sets or unions, or constructing a new set by removing some elements from an already existing set. Since there is no unrestricted quantification in this set theory, the set of all sets that do not contain themselves can not be constructed. Even further, every set is required to be well-founded, so that no set can contain itself.

The way ZF deals with "sets" like discovered by Russel and Cantor is by creating another construct called *classes* which simply are collections of sets. If a class itself is not a set, like the ones discussed previously, it is called a *proper class*. Thus, there is no paradox, since the class of all sets that do not contain themselves is not a set, and thus we can not even ask whether it contains itself, since sets can not contain classes. However, the notion of classes is again only loosely defined, whereas, for example, in the von-Neumann–Bernays–Gödel set theory (NBG for short) they are properly axiomatized.

**What can we learn from all this?** If you think back to the very beginning of chapter 4, we introduced sets with a simple metaphor, namely a box into which you can put anything you like. From that we formalized sets and came up with all sorts of things to do with them. Was all this for nothing if the foundation we used is able to produce such paradoxes?

In the real world, mathematicians and computer scientists actually don't care too much about this; naive set theory is still fine to use as long as you do not construct weird sets that contain every object of a certain property. Especially for our purposes, the set theory taught in this chapter is sufficient to tackle most problems during your course of studies. And you will need it: set theory is one of the foundation of mathematics and therefore leaks into almost every subject. Even in computer science, it shows up whenever want to formally study a concept, like the theory of computability, or in database theory, or really anywhere, as soon as we want to describe all objects having a certain property.

*This page is intentionally left blank.*

# 5 | Inductive Proofs

Proving that a proposition holds for a (small) finite number of elements is rather easy. Consider the set $\mathcal{S} := \{\heartsuit, \diamondsuit, \clubsuit, \spadesuit\}$ and the following function $f : \mathcal{S} \to \mathbb{N}$:

$$f(\heartsuit) := 256 \qquad f(\diamondsuit) := 42 \qquad f(\clubsuit) := 128 \qquad f(\spadesuit) := 1337$$

If we want to prove that $\forall s \in \mathcal{S} : f(s) \geq 42$, we can simply evaluate the $f$ for every suit and check whether the resulting number is at least 42. Proving such propositions becomes much harder if we have to deal with infinitely many elements. Consider the recursively defined **Fibonacci function** $\mathit{fib} : \mathbb{N} \to \mathbb{N}$:

$$\begin{aligned} \mathit{fib}(0) &:= 0 \\ \mathit{fib}(1) &:= 1 \\ \mathit{fib}(n+2) &:= \mathit{fib}(n) + \mathit{fib}(n+1) \end{aligned}$$

How fast does this function grow? Let's check:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $1.5^n$ | 1 | 1.5 | 2.25 | 3.38 | 5.06 | 7.59 | 11.39 | 17.09 | 25.63 | 38.44 | 57.67 | 86.50 | 129.75 |
| $\mathit{fib}(n)$ | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 |
| $2^n$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 |

It seems obvious that $2^n$ grows faster than $\mathit{fib}(n)$. After all, $2^n$ is a chain of multiplications $2 \cdot 2 \cdot \ldots$ while $\mathit{fib}(n)$ is a chain of additions, so we'd expect this result, right? But wait, $1.5^n$ is a chain of multiplications as well and $1.5^{11}$ is definitely less than $\mathit{fib}(11)$. So our intuition is wrong.[1] It seems like $\forall n \in \mathbb{N}_{\geq 11} : \mathit{fib}(n) > 1.5^n$ and $\forall n \in \mathbb{N} : \mathit{fib}(n) < 2^n$. But how can we actually be sure about this? Is there a way to formally prove it?

**Chapter Goals**

In this chapter, we discuss a proof technique called induction, which enables us to prove propositions about infinite sets. We will discover different variants of induction and see their use cases.

**In Other Words:** The Essence of Induction

Writing inductive proofs might seem a bit complicated at first and it definitely requires some training. But the idea behind induction can be briefly summarized. Just imagine that the following two conditions hold:

- There is a person, Giuseppe, who knows induction.
- Every person that knows induction teaches induction to another person who does not know induction yet.

Then finally, everybody knows induction![a]

[a]Actually, infinitely many people do!

[1] *fib* does not build a chain but an entire tree of additions with significantly more operations than in $2^n$.

## 5.1 Natural Induction

Before we prove the two propositions about *fib* from the introduction, we start with a simpler example. You might have heard about the **Gaussian sum**: the sum of all natural numbers up to $n$ is just $\frac{n(n+1)}{2}$, i.e.

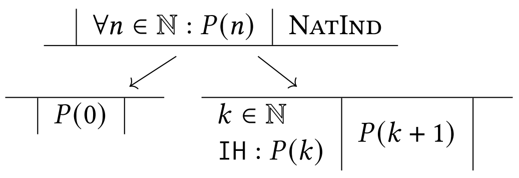
$$\forall n \in \mathbb{N} : 0 + 1 + \ldots + (n-1) + n = \frac{n(n+1)}{2}.$$

If we naively tried to prove this, we would probably start like this:

- For $n = 0$ we have $0 = \frac{0 \cdot (0+1)}{2}$.
- For $n = 1$ we have $0 + 1 = 1 = \frac{2}{2} = \frac{1 \cdot (1+1)}{2}$.
- For $n = 2$ we have $0 + 1 + 2 = \ldots$ To shortcut this here, we might reuse the result from $n = 1$, so we have $0 + 1 + 2 = \frac{1 \cdot (1+1)}{2} + 2 = \frac{(1+2) \cdot 2}{2} = \frac{2 \cdot (2+1)}{2}$.
- For $n = 3$ we have $0 + 1 + 2 + 3 = \frac{2(2+1)}{2} + 3 = \frac{(2+2) \cdot 3}{2} = \frac{3 \cdot (3+1)}{2}$, reusing the result from $n = 2$.
- For $n = 4$ we have $0 + 1 + 2 + 3 + 4 = \frac{3(3+1)}{2} + 4 = \frac{(3+2) \cdot 4}{2} = \frac{4 \cdot (4+1)}{2}$, reusing the result from $n = 3$.
- …

You might observe that the latter cases all follow the same pattern: For any $k \in \mathbb{N}$, we have $0 + \cdots + k + (k+1) \overset{(*)}{=} \frac{k(k+1)}{2} + (k+1) = \frac{(k+2)(k+1)}{2} = \frac{(k+1)(k+2)}{2}$. For $(*)$, we assume that $0 + \cdots + k = \frac{k(k+1)}{2}$ holds. But this assumption—let's call it $P(k)$ for short—is really fair: we already showed $P(0)$ at the very top. Now with our general rule, we may show $P(1)$ given $P(0)$, so we have both $P(1)$ and $P(0)$. And given $P(1)$, we may show $P(2)$. We may repeat this process *arbitrary but finitely many times*,[2] and this way we show that $\forall n \in \mathbb{N} : 0 + \ldots + n = \frac{n(n+1)}{2}$.

The reasoning principle we used here is called **natural induction**. As a proof rule, it looks like this:



To show that a proposition $P(n)$ holds for all natural numbers $n \in \mathbb{N}$, it suffices to show that (1) $P(0)$ holds, and (2) that given $P(k)$ for some arbitrary $k \in \mathbb{N}$, $P(k+1)$ holds. To simplify speaking about these two cases, we call (1) the **base case** and (2) the **induction case**. Furthermore, we call the assumption $P(k)$ in the induction case **induction hypothesis** and usually name it "IH" in proof tables, although it is not really different from other assumptions.

[2]We will come back to what this means at the end of this section

**Checkpoint 5.1:** Different Starting Points

Is it possible to show a proposition for all natural numbers $\geq 42$ using the proof rule above? What about the even integers $\geq -7$?

Now, let us rewrite our proof more formally. First, we want to get rid of the $\cdots$ notation. While the notation is pretty intuitive, it is rather hard to define what it means. A much better approach is to use the $\sum$ **notation** ($\sum$ is the capital Greek letter sigma):

$$\sum_{i=m}^{n} f(i) = f(m) + f(m+1) + \cdots + f(n-1) + f(n)$$

$f$ can be an arbitrary function $\mathbb{Z} \to \mathbb{R}$. This was not really a definition (we still used the $\cdots$ notation), but we can define the $\sum$ notation **recursively** like this:

**Definition 5.2** ($\sum$ Notation)**.**

$$\sum_{i=m}^{n} f(i) := 0 \qquad n < m \qquad \text{"base case"}$$

$$\sum_{i=m}^{n} f(i) := f(n) + \sum_{i=m}^{n-1} f(i) \qquad n \geq m \qquad \text{"recursion case"}$$

Similarly to the NatInd rule, we also have (at least one) **base case** in a recursive definition. Here, we do not refer back to the definition. In contrast, we do so in the **recursion case**. This is a little bit different from the NatInd rule, where we have an inductive case. In principle, recursive definitions may have multiple recursion cases.

As hinted, that there is some similarity between recursion and induction. But while induction is the bottom-up approach—we start by proving or defining[3] the base cases and use inductive cases to successively build upon them—, recursion is top-down: we want to compute a function for some value, but to do that, we first need to compute it for a smaller value. We go into recursion until we (hopefully[4]) reach a base case. You will see that recursion and induction play nicely together.

When introducing new notation, it makes sense to think about precedence rules. $\sum$ has the same operator precedence as +, that is:

$$\sum_{i=0}^{1} i \cdot 2^i + 42 = \left(\sum_{i=0}^{1} (i \cdot 2^i)\right) + 42 = (0 \cdot 2^0 + 1 \cdot 2^1) + 42.$$

Using the $\sum$ notation, our proposition now looks as follows. We are ready for our first formal inductive proof:

[3] Recall that we had inductive definitions when we defined the syntax of a formal language.

[4] If we do not reach a base case for some value, the function is undefined for that value. For the $\sum$ notation we ensured that this cannot happen.

**Theorem 5.3** (Gaussian Sum)**.**

$$\forall n \in \mathbb{N} : \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

*Proof by natural induction on $n \in \mathbb{N}$.* We distinguish the following cases:

**Base case:**

$$\begin{aligned}
\sum_{i=0}^{0} i &= 0 + \sum_{i=0}^{-1} i && \mid \text{Definition of } \sum \\
&= 0 + 0 && \mid \text{Definition of } \sum \\
&= \frac{0 \cdot (0+1)}{2} && \mid \text{Arithmetic}
\end{aligned}$$

**Induction case:** By induction, we know $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$. It remains to show $\sum_{i=0}^{k+1} i = \frac{(k+1)(k+2)}{2}$:

$$\begin{aligned}
\sum_{i=0}^{k+1} i &= (k+1) + \sum_{i=0}^{k} i && \mid \text{Definition of } \sum \\
&= (k+1) + \frac{k(k+1)}{2} && \mid \text{Induction hypothesis} \\
&= \frac{2(k+1) + k(k+1)}{2} && \mid \text{Arithmetic (distributivity)} \\
&= \frac{(2+k)(k+1)}{2} && \mid \text{Arithmetic (distributivity)} \\
&= \frac{(k+1)(k+2)}{2} && \mid \text{Arithmetic (commutativity)}
\end{aligned}$$

Thus we proved that $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$ for any $n \in \mathbb{N}$. □

Another example: we can define the **factorial** $n!$ directly using recursion:

**Definition 5.4** (Factorial)**.**

$$\begin{aligned}
0! &:= 1 \\
(n+1)! &:= (n+1) \cdot n!
\end{aligned}$$

However, we could also use the $\prod$ **notation** (capital Greek letter pi), which is the equivalent to the $\sum$ notation but for multiplication.

**Definition 5.5** ($\prod$ Notation)**.**

$$\begin{aligned}
\prod_{i=m}^{n} f(i) &:= 1 && n < m \\
\prod_{i=m}^{n} f(i) &:= f(n) \cdot \prod_{i=m}^{n-1} f(i) && n \geq m
\end{aligned}$$

$\prod$ has the same operator precedence as $\cdot$, e.g.

$$\prod_{i=0}^{1} i^2 \cdot 42 + 1337 = \left(\prod_{i=0}^{1} i^2\right) \cdot 42 + 1337 = (0^2 \cdot 1^2) \cdot 42 + 1337.$$

Using this notation, the factorial function can be defined as $n! = \prod_{i=1}^{n} i$. It might seem rather obvious that these two definitions are equivalent, but we can also prove it formally using induction:

**Theorem 5.6** (Factorial in $\prod$ Notation)**.**

$$\forall n \in \mathbb{N} : n! = \prod_{i=1}^{n} i$$

*Proof by natural induction on $n \in \mathbb{N}$.* We distinguish two cases:

**Base case:**

$$\begin{aligned} 0! &= 1 && \mid \text{Definition of } ! \\ &= \prod_{i=1}^{0} i && \mid \text{Definition of } \prod \end{aligned}$$

**Induction case:** By induction, we know $k! = \prod_{i=1}^{k} i$.

$$\begin{aligned} (k+1)! &= (k+1) \cdot k! && \mid \text{Definition of } ! \\ &= (k+1) \cdot \prod_{i=1}^{k} i && \mid \text{Induction hypothesis} \\ &= \prod_{i=1}^{k+1} i && \mid \text{Definition of } \prod \end{aligned}$$

Hence, the two definitions of the factorial coincide. □

In Checkpoint 5.1, we posed the question whether we can start the induction at a number different from 0. To make this concrete: it seems that the sequence $n!$ $(1, 1, 2, 6, 24, 120, 720, \ldots)$ grows faster than the sequence $2^n$ $(1, 2, 4, 8, 16, 32, 64, \ldots)$. But $n! > 2^n$ does not hold for $n \leq 3$. So we want to show $\forall n \in \mathbb{N}_{\geq 4} : n! > 2^n$. But obviously, our induction rule does not directly match our proposition. However, we can rewrite it a little: $\forall n \in \mathbb{N} : n \geq 4 \rightarrow n! > 2^n$. In all its detail, the proof looks like this:

**Theorem 5.7** (Factorial vs. Power of 2)**.**

$$\forall n \in \mathbb{N}_{\geq 4} : n! > 2^n$$

*Proof.* We show the equivalent proposition $\forall n \in \mathbb{N} : n \geq 4 \to n! > 2^n$ by natural induction on $n \in \mathbb{N}$. Therefore, we distinguish two cases:

**Base case:** $0 \geq 4$ is false, so there is nothing left to show.

**Induction case:** By induction, we know $k \geq 4 \to k! > 2^k$. It remains to show

$$(k+1) \geq 4 \to (k+1)! > 2^{k+1}.$$

We distinguish three cases:

$k < 3$**:** $k + 1 \geq 4$ is equivalent to $k \geq 3$. This contradicts $k < 3$, so we are done.

$k = 3$**:**

$$\begin{aligned}(k+1)! &= 4! && \mid k = 3, \text{arithmetic}\\ &= 24 && \mid \text{Computation of } 4!\\ &> 16 && \mid \text{Arithmetic}^{5}\\ &= 2^4 && \mid \text{Arithmetic}\\ &= 2^{k+1} && \mid k = 3, \text{arithmetic}\end{aligned}$$

$k > 3$**:**

$$\begin{aligned}(k+1)! &= (k+1) \cdot k! && \mid \text{Definition of } n!\\ &> (k+1) \cdot 2^k && \mid \text{Induction hypothesis, } k \geq 4 \leftrightarrow k > 3\\ &> 2 \cdot 2^k && \mid \text{Arithmetic } (k > 3 \to k \geq 1)\\ &= 2^{k+1} && \mid \text{Definition of } 2^n\end{aligned}$$

Hence, we proved that $\forall n \in \mathbb{N}_{\geq 4} : n! > 2^n$. □

This proof spells out many technical details. Mathematicians would probably consider the base case as well as the case $k < 3$ to be obvious. They might even call our case $k = 3$ the base case and only $k > 3$ the induction case. For the textual proofs you write during this course, you are also allowed to gloss over such technicalities. However, it is good to be aware of what really happens here. There are computer programs that can automatically check proofs, so-called proof assistants. These require you to reason precisely, and write out such details.

**Checkpoint 5.8:** Natural Induction—Your Turn!

- Prove that $\forall n \in \mathbb{N} : 2n \leq 2^n$.
- Come up with a formula for the sum of the first $n$ even numbers. Prove it correct just like we did for the Gaussian sum.

There is still one proof left from the chapter on sets. With natural induction, we have the missing proof strategy to do this proof:

[5] If you are unfamiliar with this style of inequational reasoning, you may want to revisit Section 3.4.1.

**Theorem 5.9** (Power Set Cardinality of Finite Sets)**.** *For every finite set A, we have*

$$|\mathcal{P}(A)| = 2^{|A|}.$$

*Proof by natural induction on the size of A.* We distinguish two cases:

**Base case:** In this case we have $|A| = 0$, so $A = \emptyset$.

$$\begin{aligned}
|\mathcal{P}(A)| &= |\mathcal{P}(\emptyset)| && \mid |A| = 0 \\
&= |\{\emptyset\}| && \mid \text{Definition } \mathcal{P} \\
&= 1 && \mid \text{Definition } |\cdot| \\
&= 2^0 && \mid \text{Arithmetic} \\
&= 2^{|A|} && \mid A = \emptyset
\end{aligned}$$

**Induction case:** By induction we know $|A| = k \rightarrow |\mathcal{P}(A)| = 2^{|A|}$ for every set $A$. We need to show $|\mathcal{P}(A)| = 2^{|A|}$ for an arbitrary but fixed set $A$ with $|A| = k + 1$. Because of $|A| = k + 1$, there must be an $a \in A$. Let $A' := A \setminus \{a\}$. Since $|A'| = k$, we know $|\mathcal{P}(A')| = 2^{|A'|}$ using the induction hypothesis. We may split $\mathcal{P}(A)$ into sets that contain $a$ and such that don't. Formally, this is $\mathcal{P}(A) = \mathcal{P}(A') \cup \bigcup_{X \in \mathcal{P}(A')} \{X \cup \{a\}\}$. Note that the unions are all disjoint. Thus, we have:

$$\begin{aligned}
|\mathcal{P}(A)| &= \left| \mathcal{P}(A') \cup \bigcup_{X \in \mathcal{P}(A')} \{X \cup \{a\}\} \right| && \mid \text{Disjoint decomposition, see above} \\
&= |\mathcal{P}(A')| + \left| \bigcup_{X \in \mathcal{P}(A')} \{X \cup \{a\}\} \right| && \mid \text{Cardinality of Disjoint Sets (Theorem 4.31)} \\
&= |\mathcal{P}(A')| + \sum_{X \in \mathcal{P}(A')} |\{X \cup \{a\}\}| && \mid \text{Cardinality of Disjoint Sets} \\
&= |\mathcal{P}(A')| + \sum_{i=1}^{|\mathcal{P}(A')|} 1 && \mid \text{Definition } |\cdot| \\
&= 2 \cdot |\mathcal{P}(A')| && \mid \text{Arithmetic} \\
&= 2 \cdot 2^{|A'|} && \mid |\mathcal{P}(A')| = 2^{|A'|} \\
&= 2 \cdot 2^k && \mid |A'| = k \\
&= 2^{k+1} && \mid \text{Arithmetic} \\
&= 2^{|A|} && \mid |A| = k + 1
\end{aligned}$$

It follows that the power set of any finite set $A$ has cardinality $2^{|A|}$. □

By writing "Proof by natural induction on the size of $A$" we were not as formal as possible. The formal statement would have been $\forall n \in \mathbb{N} : \forall A : |A| = n \rightarrow |\mathcal{P}(A)| = 2^{|A|}$. However, the textual form is completely fine and a bit easier to read.

Now you might wonder why we put so much stress on things being finite here. Consider the following non-proof:

**Warning:** This “proof” is wrong!

*Proof that every set $A$ is finite by induction over the size of $A$.*

**Base case:** There is only one set of size $0$, $\emptyset$, which clearly is finite.

**Induction case:** By induction, we know that a set of size $k \in \mathbb{N}$ is finite. Every set of size $k + 1$ is finite as well.

Hence, every set is finite. □

While being a bit sloppy, the problem is not the induction itself. The statement we actually prove by induction is just not our original claim. An infinite set simply has no size $\in \mathbb{N}$. Natural induction, however, only proves statements about natural numbers.

Remember how we said that conceptually, we may apply the induction case *arbitrary but finitely many times*. If we repeatedly add one element to the empty set, but do so only finitely often, the resulting set is still finite. To construct an infinite set, we would need to repeat the process infinitely often, but this is not how induction works.

## **Going Beyond:** Why Does Natural Induction Work?

In this course, you might have learned that asking for a proof can help to see why things work. Can we prove natural induction? This question is related to asking: what are the natural numbers? They form such a basic concept that it seems hard to define them. If you are clever, you might say that we can dissect the natural numbers into 0 and **successors** of natural numbers $S(n)$. Now it is rather easy to see that we can give a BNF for this:

$$\mathbb{N} \ni n ::= 0 \mid S(n)$$

But is this the only way to model the natural numbers? Are there other equivalent models? To answer these questions, it helps to characterize the natural numbers by their properties. Giuseppe Peano did this in his "Arithmetices principia, nova methodo exposita," published in 1889. The following is not Peano's original formulation, but one that works in first-order logic with equality. It is commonly known as **Peano Arithmetic** (PA).
The natural numbers $\mathbb{N}$ satisfy the following axioms:

1. $0 \in \mathbb{N}$
2. $\forall n \in \mathbb{N} : S(n) \in \mathbb{N}$ (the natural numbers are closed under $S$)
3. $\forall n, m \in \mathbb{N} : S(n) = S(m) \to n = m$ ($S$ is injective)
4. $\forall n \in \mathbb{N} : S(n) \neq 0$ (constructor disjointness)

Now, these axioms seem fairly reasonable, but we are not done yet. Consider this model:

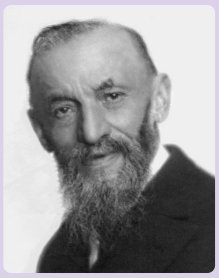
$$\begin{tikzcd}\bullet \arrow[r, bend right] & \bullet \arrow[l, bend right] & 0 \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \cdots\end{tikzcd}$$

The nodes $\bullet$ and 0 all denote natural numbers, an arrow $n \to m$ means $m = S(n)$. While the chain $0 \to \bullet \to \cdots$ is what we intend, there is also this strange cycle. If we defined $\leq$ to be the reflexive-transitive closure of $S$, it would not be well-founded. To get rid of junk like the cycle, we have the last axiom:

5. $(\psi(0) \land \forall n \in \mathbb{N} : \psi(n) \to \psi(S(n))) \to \forall n \in \mathbb{N} : \psi(n)$ (induction axiom)

$\psi$ denotes any predicate-form. It might be tempting to write $\forall \psi : \ldots$ for the last axiom, but quantifying over predicates or predicate-forms is only possible in higher-order logic. So technically, we have infinitely many copies of the last axiom, one for each predicate-form. Our induction rule is actually derived from this fifth axiom. Using induction, we may also prove that $\leq$ is a well-order. Now the model given by the BNF satisfies all these axioms, but only because we assume similar ones for arbitrary inductive definitions. This will lead us to a more general form of natural induction, namely structural induction.
To summarize, we cannot really prove natural induction in context of PA. Here, we basically assume it as an axiom. However, this is not the only possibility. In Section 5.5 we will see an even more general form of induction called well-founded induction, which on its own is provable. The only thing we need to derive natural induction from well-founded induction is a well-order on $\mathbb{N}$. But where do we get the well-order from? We could either assume it, or we can prove it using natural induction. This is kind of a chicken and egg problem …

**Important Individual:** Giuseppe Peano



Giuseppe Peano (1858-1932) was an Italian mathematician and linguist. He was a founder of mathematical logic and set theory. His most famous work is the axiomatization of the natural numbers. Furthermore, some notations such as $\cup$ and $\cap$ for the set union or intersection, respectively, as well as the symbol for the existential quantifier $\exists$ were introduced by him.

## 5.2 Complete Induction

Recall the definition of the **Fibonacci function** from the introduction:

**Definition 5.10** (Fibonacci)**.**

$$\begin{aligned} \mathit{fib} &: \mathbb{N} \to \mathbb{N} \\ \mathit{fib}(0) &:= 0 \\ \mathit{fib}(1) &:= 1 \\ \mathit{fib}(n+2) &:= \mathit{fib}(n) + \mathit{fib}(n+1) \end{aligned}$$

Why didn't we directly prove $\forall n \in \mathbb{N} : \mathit{fib}(n) < 2^n$? Let us try. First, we consider the two base cases of the recursive definition.[6] For $n = 0$, we have $\mathit{fib}(0) = 0 < 1 = 2^0$. For $n = 1$, we have $\mathit{fib}(1) = 1 < 2 = 2^1$. Now for $n = k + 2$, we have $\mathit{fib}(k+2) = \mathit{fib}(k) + \mathit{fib}(k+1)$. By induction, we also know that $\mathit{fib}(k+1) < 2^{k+1}$. But our list of assumptions does not contain $\mathit{fib}(k) < 2^k$. Sadly, we know nothing about $\mathit{fib}(k)$. We are stuck!

If this seems strange to you, you are absolutely right. It is true that our list of assumptions does not contain anything about $\mathit{fib}(k)$, but when we do induction, we conceptually build up a chain: we first show that our proposition $P$ holds for 0, then we show $P(1)$ given $P(0)$ as induction hypothesis, then $P(2)$ given $P(1)$ and so on. But why shouldn't we be able to use $P(0)$ as well when proving $P(2)$? Or more generally, why shouldn't we use $P(l)$ for any $l \in \mathbb{N}$ with $l < k$ when proving $P(k)$? All of this works perfectly fine. And indeed, there is a variant of induction that gives us such a stronger induction hypothesis: **complete induction**. Sometimes, this is called **strong induction** as well. Here is what the proof rule looks like:

| | $\forall n \in \mathbb{N} : P(n)$ | CompInd |
|---|---|---|
| $k \in \mathbb{N}$<br>$\mathtt{IH} : \forall l \in \mathbb{N} : l < k \to P(l)$ | $P(k)$ | |

Interestingly, the rule does not distinguish between a base and an induction case anymore. But still, we usually have to show $P(0)$ separately, since the induction hypothesis is useless for $k = 0$: for any number $l \in \mathbb{N}$, $l < 0$ is always false.

We are now ready to prove our theorem:

[6] Technically, we apply our induction rule first and in the inductive case, we do a case distinction on $n$.

**Theorem 5.11** (Upper Bounds for *fib*).

$$\forall n \in \mathbb{N} : \mathit{fib}(n) < 2^n$$

*Proof by complete induction on $n \in \mathbb{N}$.* By induction, we know $\forall l \in \mathbb{N} : l < k \to \mathit{fib}(l) < 2^l$. It remains to show that $\mathit{fib}(k) < 2^k$. We distinguish three cases:

$k = 0$:

$$\begin{aligned} \mathit{fib}(k) &= \mathit{fib}(0) && \mid k = 0 \\ &= 0 && \mid \text{Definition of } \mathit{fib} \\ &< 1 && \mid \text{Arithmetic} \\ &= 2^0 && \mid \text{Arithmetic} \\ &= 2^k && \mid 0 = k \end{aligned}$$

$k = 1$:

$$\begin{aligned} \mathit{fib}(k) &= \mathit{fib}(1) && \mid k = 1 \\ &= 1 && \mid \text{Definition of } \mathit{fib} \\ &< 2 && \mid \text{Arithmetic} \\ &= 2^1 && \mid \text{Arithmetic} \\ &= 2^k && \mid 1 = k \end{aligned}$$

$k = k' + 2$:

$$\begin{aligned} \mathit{fib}(k) &= \mathit{fib}(k' + 2) && \mid k = k' + 2 \\ &= \mathit{fib}(k') + \mathit{fib}(k' + 1) && \mid \text{Definition of } \mathit{fib} \\ &< 2^{k'} + 2^{k'+1} && \mid \text{Induction hypothesis for } k', k' + 1 \\ &< 2^{k'+1} + 2^{k'+1} && \mid \text{Arithmetic} \\ &< 2^{k'+2} && \mid \text{Arithmetic} \\ &= 2^k && \mid k' + 2 = k \end{aligned}$$

Hence, we have shown that $2^n$ is an upper bound for $\mathit{fib}(n)$. □

**Checkpoint 5.12:** Lower Bounds

Show that $\forall n \in \mathbb{N}_{\geq 11} : \mathit{fib}(n) > \left(\frac{2}{3}\right)^n$. You may use that $\mathit{fib}(11) = 89$, $\left(\frac{2}{3}\right)^{11} < 87$, $\mathit{fib}(12) = 144$, and $\left(\frac{2}{3}\right)^{12} < 130$. Hint: $\left(\frac{2}{3}\right)^k = \frac{2^k}{3^k}$.

Another interesting property we can show using complete induction is that $\leq$ on $\mathbb{N}$ is **well-founded**. Recall that an order $\mathcal{R}$ is well-founded on a set $\mathcal{M}$ if every non-empty subset $\mathcal{U} \subseteq \mathcal{M}$ has a minimal element. Intuitively, this is clear for $\leq$. The minimal element is just $\min \mathcal{U}$ (e.g. $\min\{4, 2, 8\} = 2$). But we haven't proved yet that min on $\mathbb{N}$ is well-defined. Let's do this now.

**Theorem 5.13** ($\leq$ is Well-Founded on $\mathbb{N}$).

$$\forall \mathcal{U} \subseteq \mathbb{N} : \mathcal{U} \neq \emptyset \rightarrow \exists m \in \mathcal{U} : \forall m' \in \mathcal{U} : m' \nless m$$

*Proof.* Let $\mathcal{U} \subseteq \mathbb{N}$ with $\mathcal{U} \neq \emptyset$. It remains to show that $\mathcal{U}$ has a minimal element (that is, $\exists m \in \mathcal{U} : \forall m' \in \mathcal{U} : m' \nless m$). We prove this by contradiction: We assume that $\mathcal{U}$ has no minimal element, i.e. $\forall m \in \mathcal{U} : \exists m' \in \mathcal{U} : m' < m$, and have to derive $\perp$. It suffices to show that there is no natural number in $\mathcal{U}$, formally $\forall n \in \mathbb{N} : n \notin \mathcal{U}$ (together with $\mathcal{U} \subseteq \mathbb{N}$ and $\mathcal{U} \neq \emptyset$ this yields $\perp$, as desired).

We prove the new goal $\forall n \in \mathbb{N} : n \notin \mathcal{U}$ by complete induction on $n \in \mathbb{N}$, so we obtain $\forall k \in \mathbb{N} : k < n \rightarrow k \notin \mathcal{U}$ as induction hypothesis. Our goal, $n \notin \mathcal{U}$, is equivalent to $n \in \mathcal{U} \rightarrow \perp$, so we may assume $n \in \mathcal{U}$ and derive $\perp$. $n \in \mathcal{U}$ in combination with the assumption that $\mathcal{U}$ has no minimal element ($\forall m \in \mathcal{U} : \exists m' \in \mathcal{U} : m' < m$) gives us an $m' \in \mathcal{U}$ with $m' < n$. Finally, we distinguish two cases:

$n = 0$: We have $m' < 0$, a contradiction.

$n > 0$: We may instantiate the induction hypothesis ($\forall k \in \mathbb{N} : k < n \rightarrow k \notin \mathcal{U}$) with $m'$, since $m' \in \mathcal{U} \subseteq \mathbb{N}$ and $m' < n$. This gives us $m' \notin \mathcal{U}$, again a contradiction.

This means that the assumption "$\mathcal{U}$ has no minimal element" is false. So $\mathcal{U}$ has a minimal element and $\leq$ is well-founded on $\mathbb{N}$. □

**Going Beyond:** Is Complete Induction Really Stronger than Natural Induction?

From the application perspective, we clearly have seen that complete induction is more powerful than natural induction. But it turns out that we can prove complete induction using natural induction. Given any predicate-form $\psi(n)$, we can view the proof rule as a formula in first-order logic, which we'll need to prove:

**Lemma 5.14** (Complete induction)**.**

$$(\forall k \in \mathbb{N} : (\forall l \in \mathbb{N} : l < k \rightarrow \psi(l)) \rightarrow \psi(k)) \rightarrow \forall n \in \mathbb{N} : \psi(n)$$

*Proof.* Assume $\forall k \in \mathbb{N} : (\forall l \in \mathbb{N} : l < k \rightarrow \psi(l)) \rightarrow \psi(k)$. We call this assumption $H$. Let $n \in \mathbb{N}$ be arbitrary, but fixed. By applying $H$, it remains to show $\forall l \in \mathbb{N} : l < n \rightarrow \psi(l)$. We prove this by natural induction on $n \in \mathbb{N}$:

**Base case:** We need to show $\forall l \in \mathbb{N} : l < 0 \rightarrow \psi(l)$. Let $l \in \mathbb{N}$ be arbitrary, but fixed. We are done since $l < 0$ is always false.

**Induction case:** By induction, we have $\forall l \in \mathbb{N} : l < n \rightarrow \psi(l)$. We need to show $\forall l \in \mathbb{N} : l < n+1 \rightarrow \psi(l)$. Assume $l \in \mathbb{N}$ such that $l < n+1$. We consider two cases:

$l \neq n$**:** In this case, we know that $l < n$. We are done by applying the induction hypothesis.

$l = n$**:** As $l = n$, showing $\psi(n)$ suffices. By applying $H$ from the very beginning, we are left with $\forall l \in \mathbb{N} : l < n \rightarrow \psi(l)$, which is exactly the induction hypothesis.

Thus, we have proven complete induction. □

This proof is quite technical and pretty abstract. In particular, the order of quantification during the natural induction step is important. If you do not understand what is happening here, it might help to draw the proof tables.

**Checkpoint 5.15:** Fibonacci Without Complete Induction

It is possible to show $\forall n \in \mathbb{N} : \mathit{fib}(n) < 2^n$ without complete induction. Hint: prove

$$\forall n \in \mathbb{N} : \mathit{fib}(n) < 2^n \wedge \mathit{fib}(n+1) < 2^{n+1}$$

by natural induction.

## 5.3 Quantified Inductive Hypotheses

We can also use recursion when programming: We write a function that figures out what case it is in and calls itself as necessary. Unfortunately, if we do this naively, our programs might not be as fast as possible. Recall our definition of the factorial function:

$$0! := 1$$
$$(n+1)! := (n+1) \cdot n!$$

Now consider this computation of 4!:

$$\begin{aligned}
4! &= 4 \cdot 3! \\
&= 4 \cdot (3 \cdot 2!) \\
&= 4 \cdot (3 \cdot (2 \cdot 1!)) \\
&= 4 \cdot (3 \cdot (2 \cdot (1 \cdot 0!))) \\
&= 4 \cdot (3 \cdot (2 \cdot (1 \cdot 1))) \\
&= 4 \cdot (3 \cdot (2 \cdot 1)) \\
&= 4 \cdot (3 \cdot 2) \\
&= 4 \cdot 6 \\
&= 24
\end{aligned}$$

We first build up a chain of multiplications and then evaluate this to a single value. This is exactly what our definition tells us to do. But it also means that our computer has to store this multiplication chain in memory. The computation would be much faster if we were allowed to reassociate the multiplications such that we could directly multiply $4 \cdot 3$, then $12 \cdot 2$ and so on. And indeed, this is possible if we change our definition:

> **Definition 5.16** (Tail-Recursive Factorial)**.**
>
> $$\begin{aligned}
> \mathit{fac} &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\
> \mathit{fac}(a, 0) &:= a \\
> \mathit{fac}(a, n) &:= \mathit{fac}(n \cdot a, n - 1) \qquad n > 0
> \end{aligned}$$

Now the claim is that to compute $n!$, we can also compute $\mathit{fac}(1, n)$. So let's compute $\mathit{fac}(1, 4)$:

$$\begin{aligned}
\mathit{fac}(1, 4) &= \mathit{fac}(4 \cdot 1, 4 - 1) \\
= \mathit{fac}(4, 3) &= \mathit{fac}(3 \cdot 4, 3 - 1) \\
= \mathit{fac}(12, 2) &= \mathit{fac}(2 \cdot 12, 2 - 1) \\
= \mathit{fac}(24, 1) &= \mathit{fac}(1 \cdot 24, 1 - 1) \\
= \mathit{fac}(24, 0) &= 24
\end{aligned}$$

Observe that when going into recursion, there are no other computations left to do. This is in contrast to !, where we have to perform one multiplication after each recursion step. Recursive functions without computations after going into recursion are called **tail recursive**.

Compared to the original version, the tail-recursive *fac* has one additional argument $a$. We use this to carry our intermediate results around. This argument has a special name, we call it the **accumulator**.

After discussing the differences, let's return to what the two functions (should) share: they should in some sense compute the same result. More formally, we claimed that $\forall n \in \mathbb{N} : \mathit{fac}(1, n) = n!$. Can we prove it? The recursive case of $\mathit{fac}(a, n)$ only depends on $\mathit{fac}(n \cdot a, n-1)$, so natural induction should suffice. The base case is simple: $\mathit{fac}(1, 0) = 1 = 1!$. For the induction case, our induction hypothesis is $\mathit{fac}(1, k) = k!$. By definition, we have $\mathit{fac}(1, k+1) = \mathit{fac}((k+1) \cdot 1, (k+1) - 1) = \mathit{fac}(k+1, k)$. But unfortunately, we cannot apply our induction hypothesis here! It requires that the accumulator is 1, but we have $k + 1$ instead. Clearly, we need an induction hypothesis with an arbitrary natural number as the first argument of *fac*, something like $\forall a \in \mathbb{N} : \mathit{fac}(a, n) = \ldots$

But what is $\mathit{fac}(a, n)$ for an $a$ that is not necessarily 1? The base case of our definition gives us $a$ instead of just 1. So we have $a \cdot 0!$. And also $\mathit{fac}(42, 2)$ gives us $42 \cdot 2 \cdot 1 = 42 \cdot 2!$. So we might guess that $\mathit{fac}(a, n) = a \cdot n!$. So let's try to prove the following lemma:

**Lemma 5.17** (Correctness of Tail-Recursive Factorial)**.**

$$\forall n \in \mathbb{N} : \forall a \in \mathbb{N} : \mathit{fac}(a, n) = a \cdot n!$$

*Proof by natural induction on $n \in \mathbb{N}$.* We distinguish the following cases:

**Base case:** Let $a \in \mathbb{N}$ be arbitrary, but fixed.

$$\begin{aligned} \mathit{fac}(a, 0) &= a && \mid \text{Definition of } \mathit{fac} \\ &= a \cdot 1 && \mid \text{Arithmetic} \\ &= a \cdot 0! && \mid \text{Definition of } n! \end{aligned}$$

**Induction case:** By induction, we have $\forall a \in \mathbb{N} : \mathit{fac}(a, k) = a \cdot k!$. Let $a \in \mathbb{N}$ be arbitrary, but fixed.

$$\begin{aligned} \mathit{fac}(a, k+1) &= \mathit{fac}((k+1) \cdot a, (k+1) - 1) && \mid \text{Definition of } \mathit{fac} \\ &= \mathit{fac}((k+1) \cdot a, k) && \mid \text{Arithmetic} \\ &= ((k+1) \cdot a) \cdot k! && \mid \text{Induction hypothesis for } (k+1) \cdot a \\ &= a \cdot ((k+1) \cdot k!) && \mid \text{Arithmetic} \\ &= a \cdot (k+1)! && \mid \text{Definition of } n! \end{aligned}$$

Hence, we have shown that $\mathit{fac}(a, n) = a \cdot n!$ holds for every $a, n \in \mathbb{N}$. □

Now $\forall n \in \mathbb{N} : \forall a \in \mathbb{N} : \mathit{fac}(a, n) = a \cdot n!$ trivially implies $\forall n \in \mathbb{N} : \mathit{fac}(1, n) = n!$ (change the quantifier order and instantiate with 1).

Note that the order of quantifiers in the Lemma is important: we want to apply the induction rule first and have a *quantified induction hypothesis* $\forall a \in \mathbb{N} : \ldots$ If the quantifier order was the other way round, we would have introduced $a$ first. If we applied the induction rule then, our induction hypothesis would have been $\mathit{fac}(a, k) = a \cdot k!$ for some fixed $a$. Observe that our proof would have failed with this induction hypothesis, just as it did when we had $\mathit{fac}(1, k) = k!$ as induction hypothesis.

Let's summarize our procedure. We first had a proposition that we could not prove. Then we made it more general by quantifying over the accumulator. A more general proposition says more (for example $\forall n \in \mathbb{N} : n \geq 0$ is more informative than $1 \geq 0$). Sometimes we also say that a more general proposition is *stronger*: if we have it as an assumption it is more powerful. But in the first place, we make our goal stronger. Would you expect that fighting against a stronger opponent is easier in the end? Remarkably, this was the case. Strengthening our goal also meant that we got a stronger induction hypothesis. This is why our process is sometimes called *strengthening the inductive hypothesis.* And using the stronger induction hypothesis we could show our the stronger proposition. Finally, we could prove our original proposition as the stronger one directly implied it.

**Checkpoint 5.18:** Recursion Transformer

Consider the following recursive functions:

$$\begin{aligned} pow &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} & mul &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ pow(n, 0) &:= 1 & mul(n, 0) &:= 0 \\ pow(n, m) &:= n \cdot pow(n, m-1) \quad m > 0 & mul(n, m) &:= n + mul(n, m-1) \quad m > 0 \end{aligned}$$

Make them tail-recursive and prove that your tail-recursive versions are correct.

## 5.4 Structural Induction

Natural numbers are not the only thing that forms an infinite set, this is also the case for most of the languages we defined in Chapter 1. Can we reason inductively for them as well?

Further, we may also view the natural numbers as a special case of an inductive definition:

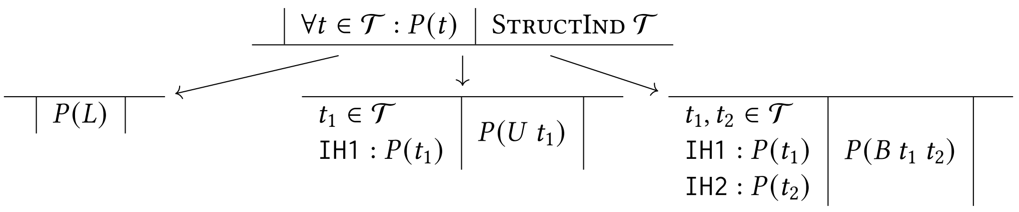
$$\mathbb{N} \ni n ::= 0 \mid S(n)$$

0 is just the smallest natural number. $S$ is the successor function that adds 1 to its argument. So with this definition, we have $4 = S(S(S(S(0))))$. Can we use this to view natural induction as a special case of a more general concept?

The answer to both questions is yes. Let's reconsider our language of binary trees:

$$\mathcal{T} \ni \varphi, \psi ::= L \mid U\ \varphi \mid B\ \varphi\ \psi$$

Up to the choice of names, the BNF is pretty similar to the $\mathbb{N}$-BNF: The base case is now called $L$, and we also have an induction case with one meta-variable, $U$. However, there is now $B$, an induction case with two meta-variables. What does this mean for inductive reasoning? The principle is still the same: We start with the elements in our language, the base cases. Here, we only need to show that our proposition—call it $P$—holds for the tree $L$. Then, we show that constructing any tree from trees for which $P$ holds gives us a tree for which $P$ holds again. Concretely, there we need to (a) show $P(U\ t_1)$ given that $P(t_1)$ holds and (b) prove $P(B\ t_1\ t_2)$ given that $P(t_1)$ and $P(t_2)$ hold. These are our *two* induction cases with their induction hypotheses. So our proof rule looks like this:



As there are infinitely many BNFs, we cannot write down all structural induction rules here. But the principle to create them is always the same: for every case of our BNF, we have one case in our proof. For every meta-variable, we get one inductive hypothesis.

Now let's do an example proof using structural induction. Recall that the **breadth** of a tree is the count of its leafs. The **depth** is the length of the longest path from the root to a leaf. Formally, they were defined as:

$$\begin{aligned} \mathit{breadth} &: \mathcal{T} \to \mathbb{N} & \mathit{depth} &: \mathcal{T} \to \mathbb{N} \\ \mathit{breadth}(L) &= 1 & \mathit{depth}(L) &= 0 \\ \mathit{breadth}(U\ t_1) &= \mathit{breadth}(t_1) & \mathit{depth}(U\ t_1) &= 1 + \mathit{depth}(t_1) \\ \mathit{breadth}(B\ t_1\ t_2) &= \mathit{breadth}(t_1) + \mathit{breadth}(t_2) & \mathit{depth}(B\ t_1\ t_2) &= 1 + \max(\mathit{depth}(t_1), \mathit{depth}(t_2)) \end{aligned}$$

**Theorem 5.19** (Upper Bound for the Breadth of Binary Trees)**.**

$$\forall t \in \mathcal{T} : \mathit{breadth}(t) \leq 2^{\mathit{depth}(t)}$$

*Proof by structural induction on $t \in \mathcal{T}$.* We distinguish three cases:

*L*:

$$\begin{aligned} \mathit{breadth}(L) &= 1 && \mid \text{Definition of } \mathit{breadth} \\ &\leq 2^0 && \mid \text{Arithmetic} \\ &= 2^{\mathit{depth}(L)} && \mid \text{Definition of } \mathit{depth} \end{aligned}$$

$U\ t_1$: By induction, we have $\mathit{breadth}(t_1) \leq 2^{\mathit{depth}(t_1)}$.

$$\begin{aligned} \mathit{breadth}(U\ t_1) &= \mathit{breadth}(t_1) && \mid \text{Definition of } \mathit{breadth} \\ &\leq 2^{\mathit{depth}(t_1)} && \mid \text{Induction hypothesis} \\ &\leq 2^{1+\mathit{depth}(t_1)} && \mid \text{Arithmetic} \\ &= 2^{\mathit{depth}(U\ t_1)} && \mid \text{Definition of } \mathit{depth} \end{aligned}$$

$B\ t_1\ t_2$: By induction, we have $\mathit{breadth}(t_1) \leq 2^{\mathit{depth}(t_1)}$ and $\mathit{breadth}(t_2) \leq 2^{\mathit{depth}(t_2)}$.

$$\begin{aligned} \mathit{breadth}(B\ t_1\ t_2) &= \mathit{breadth}(t_1) + \mathit{breadth}(t_2) && \mid \text{Definition of } \mathit{breadth} \\ &\leq 2^{\mathit{depth}(t_1)} + 2^{\mathit{depth}(t_2)} && \mid \text{Induction hypotheses} \\ &\leq 2 \cdot \max(2^{\mathit{depth}(t_1)}, 2^{\mathit{depth}(t_2)}) && \mid \text{Arithmetic} \\ &= 2 \cdot 2^{\max(\mathit{depth}(t_1), \mathit{depth}(t_2))} && \mid \text{Arithmetic} \\ &= 2^{1+\max(\mathit{depth}(t_1), \mathit{depth}(t_2))} && \mid \text{Arithmetic} \\ &= 2^{\mathit{depth}(B\ t_1\ t_2)} && \mid \text{Definition of } \mathit{depth} \end{aligned}$$

Thus we have shown that $2^{\mathit{depth}(t)}$ is an upper bound for the breadth of any binary tree $t$. □

## 5.5 Going Beyond: Well-founded Induction

If there is complete induction as a stronger variant of natural induction, is there a stronger variant of structural induction as well? As it turns out, there is, and it's called **well-founded induction** or **Noetherian induction** (after Emmy Noether).

**Important Individual:** Emmy Noether

Amalie Emmy Noether (1882-1935) was a German mathematician who made many important contributions to abstract algebra and theoretical physics. When she completed her doctorate in 1907 on invariant theory, she was the second woman to receive a PhD in mathematics from a German university. She was also the first woman in Germany to habilitate in mathematics. Although her work was well-received, she worked for many years without any pay. Several mathematicians and physicists (including Albert Einstein) describe her as the most important woman in the history of mathematics.

Here is what the proof rule looks like:

| Hwf : $\mathcal{R} \subseteq \mathbb{X} \times \mathbb{X}$ *is a well-founded order* | $\forall x \in \mathbb{X} : P(x)$ | WFInd $\mathcal{R}$ |
|---|---|---|
| $x \in \mathbb{X}$<br>IH : $\forall y \in \mathbb{X} : (y, x) \in \mathcal{R} \wedge y \neq x \rightarrow P(y)$ | $P(x)$ | |

When proving $P(x)$, the WFInd rules gives us one assumption for free: for every element $y$ that is "smaller" than $x$ according to $\mathcal{R}$, we may assume that $P(y)$ holds. The rule is quite similar to the one of complete induction, although more general. We know that $\leq$ on $\mathbb{N}$ is a well-order, so we can simply plug it in for $\mathcal{R}$ (and $\mathbb{N}$ for $\mathbb{X}$). Then the induction hypothesis becomes $\forall y \in \mathbb{N} : y \leq x \wedge y \neq x \rightarrow P(y)$. The formula $y \leq x \wedge y \neq x$ is true iff $x$ is smaller than $y$, so the induction hypothesis is equivalent to the one we have for complete induction: $\forall y \in \mathbb{N} : y < x \rightarrow P(y)$.

So well-founded induction is a generalization of complete induction. But as mentioned above, it is a generalization of structural induction as well. For a BNF, there is a direct subexpression relation. By direct we mean e.g. the children of a node in a tree, not arbitrary descendants. In contrast, an indirect subexpression would be any descendant. Put differently, the (indirect) subexpression relation is just the reflexive-transitive closure of the direct subexpression relation. Structural induction allows us to use induction hypotheses for direct subexpressions only. However, it would be really fair to assume that the proposition holds for all indirect subexpressions (excluding the expression itself) as well. This is what well-founded induction allows us to do, by plugging the (indirect) subexpression relation in for $\mathcal{R}$.

In most cases, the other variants of induction we have seen are already enough. Nevertheless, well-founded induction is really useful. For instance, we can show that every natural number $\geq 2$ has a prime factorization:

> **Definition 5.20** (Prime Factorization)**.** *A prime factorization is a decomposition of a natural number into a product that only consists of prime factors.*

**Example 5.21:** Prime Factorization

$10 = 2 \cdot 5$ $\qquad$ $11 = 11$ $\qquad$ $12 = 2 \cdot 2 \cdot 3$

As the example suggests, the prime factorization of any $n \in \mathbb{N}_{\geq 2}$ is also unique, but we won't show this here. We only show that one exists.

Now, what do we know about prime factors? The prime factorization of a prime is the prime itself. If we know a prime factorization of two numbers $x, y$, we also know a prime factorization of $x \cdot y$, it is just the product of the prime factors of $x$ and the prime factors of $y$. So when we want to show that 8 has a prime factorization, this is easy if we have already shown that both 4 and 2 have a prime factorization (since $8 = 4 \cdot 2$). You might already be able to see how we will build up our "chain" of reasoning:

$$\begin{tikzcd}
8 & & & & & & \\
4 \arrow[u] & 6 & 9 & & 10 & & \\
2 \arrow[u] \arrow[ur] \arrow[urrrr] \arrow[uu, bend left] & & 3 \arrow[ul] \arrow[u] & & 5 \arrow[u] & & 7
\end{tikzcd}$$

Figure 5.22: Order of reasoning for the proof

Now, what is the order that we sketched in Figure 5.22? It is just the divisibility relation

$$\left\{(x, y) \in \mathbb{N}_{\geq 2}^2 \;\middle|\; \exists k \in \mathbb{N} : y = k \cdot x\right\}.$$

This will be the well-order we choose for the induction. But first, we have to prove that it really is a well-order:

**Lemma 5.23** (Divisibility Relation is a Well-Order)**.** *The relation*

$$\mathcal{R} := \left\{(x, y) \in \mathbb{N}_{\geq 2}^2 \;\middle|\; \exists k \in \mathbb{N} : y = k \cdot x\right\}$$

*is a well-order on* $\mathbb{N}_{\geq 2}$*.*

*Proof.* We need to show that $\mathcal{R}$ is an order, meaning that it is reflexive, transitive and anti-symmetric, as well as that $\mathcal{R}$ is well-founded.

**Reflexive** We need to show that $\forall x \in \mathbb{N}_{\geq 2} : (x, x) \in \mathcal{R}$, which is equivalent to

$$\forall x \in \mathbb{N}_{\geq 2} : \exists k \in \mathbb{N} : x = k \cdot x.$$

Let $x \in \mathbb{N}_{\geq 2}$ be arbitrary, but fixed. Pick $k := 1$. We are done, since $1 \cdot x = x$.

**Transitive** We need to show $\forall x, y, z : (x, y) \in \mathcal{R} \land (y, z) \in \mathcal{R} \to (x, z) \in \mathcal{R}$, that is

$$\forall x, y, z \in \mathbb{N}_{\geq 2} : (\exists k_1 \in \mathbb{N} : y = k \cdot x) \land (\exists k_2 \in \mathbb{N} : z = k \cdot y) \longrightarrow \exists k \in \mathbb{N} : z = k \cdot x.$$

Let $x, y, z \in \mathbb{N}_{\geq 2}$ be arbitrary, but fixed. Assume that there are $k_1, k_2 \in \mathbb{N}$ such that $y = k_1 \cdot x$ and $z = k_2 \cdot y$. Pick $k := k_1 \cdot k_2$. We have

$$\begin{aligned}
z &= k_2 \cdot y && \mid \text{Assumption} \\
&= k_2 \cdot (k_1 \cdot x) && \mid \text{Assumption} \\
&= (k_2 \cdot k_1) \cdot x && \mid \text{Associativity} \\
&= k \cdot x && \mid \text{Definition of } k
\end{aligned}$$

**Anti-symmetric** We need to show $\forall x, y : (x, y) \in \mathcal{R} \land (y, x) \in \mathcal{R} \to x = y$, i.e.

$$\forall x, y \in \mathbb{N}_{\geq 2} : (\exists k_1 \in \mathbb{N} : y = k_1 \cdot x) \land (\exists k_2 \in \mathbb{N} : x = k_2 \cdot y) \to x = y.$$

Let $x, y \in \mathbb{N}_{\geq 2}$ be arbitrary, but fixed. Assume $k_1, k_2 \in \mathbb{N}$ such that $y = k_1 \cdot x$ and $x = k_2 \cdot y$. By our assumptions, we have $x = k_2 \cdot y = k_2 \cdot k_1 \cdot x$. Now $x = k \cdot x$ holds for arbitrary $x$ only if $k = 1$, which means that we have $k_2 \cdot k_1 = 1$. But since $k_1, k_2 \in \mathbb{N}$, it must be that $k_1 = k_2 = 1$. So finally, $x = k_2 \cdot y = 1 \cdot y = y$.

**Well-founded** We need to show that every non-empty subset of $\mathbb{N}_{\geq 2}$ contains a minimal element with respect to $\mathcal{R}$, i.e.

$$\forall \mathcal{U} \subseteq \mathbb{N}_{\geq 2} : \mathcal{U} \neq \emptyset \to \exists m \in \mathcal{U} : \forall m' \in \mathcal{U} : (m', m) \in \mathcal{R} \to m = m'.$$

Let $\mathcal{U} \subseteq \mathbb{N}_{\geq 2}$ be arbitrary, but fixed. Assume that $\mathcal{U}$ is not empty. Since $\leq$ is a well-order on $\mathbb{N}_{\geq 2}$, there exists a minimal element $\min \mathcal{U}$ with respect to $\leq$. Pick $m := \min \mathcal{U}$. Let $m' \in \mathcal{U}$ be arbitrary, but fixed. Unfolding the definition of $\mathcal{R}$, it remains to show that

$$(\exists k \in \mathbb{N} : \min \mathcal{U} = k \cdot m') \to \min \mathcal{U} = m'.$$

Assume a $k \in \mathbb{N}$ such that $m' = k \cdot \min \mathcal{U}$. We distinguish three cases:

$\min \mathcal{U} = m'$**:** We are done since this is exactly what we needed to show.

$\min \mathcal{U} > m'$**:** Contradiction: $\min \mathcal{U}$ is a minimal element with respect to $\leq$.

$\min \mathcal{U} < m'$**:** Contradicts $\min \mathcal{U} = k \cdot m'$ and $k \in \mathbb{N}$.

This means that $\mathcal{R}$ is well-founded on $\mathbb{N}_{\geq 2}$.

Since all these properties hold, the divisibility relation $\mathcal{R}$ is a well-order on $\mathbb{N}_{\geq 2}$. □

**Checkpoint 5.24:** Wrong Relation

What would have gone wrong if we had chosen $\mathcal{R} := \left\{(x, y) \in \mathbb{N}^2 \;\middle|\; \exists k \in \mathbb{N} : y = k \cdot x\right\}$?

Now we are ready for the induction:

**Theorem 5.25** (Existence of the Prime Factorization)**.** *Every natural number $n \in \mathbb{N}_{\geq 2}$ has a prime factorization.*

*Proof.* By Lemma 5.23, we know that $\mathcal{R}$ is a well-order, so we can perform well-founded induction. Now we need to show that an arbitrary $x \in \mathbb{N}_{\geq 2}$ has a prime factorization. By induction, we know that any strict divisor $y \in \mathbb{N}_{\geq 2}$ of $x$ has a prime factorization. We distinguish two cases:

***n*** **is a prime:** A prime factorization of $n$ is just $n$ (i.e. the unary product, if you want to).

***n*** **is not a prime:** In this case there are two factors $k_1, k_2 \notin \{1, n\}$ such that $n = k_1 \cdot k_2$. By applying the induction hypothesis we obtain a prime factorization $k_1$ and one for $k_2$. A prime factorization of $n$ is just the product of these prime factorizations.

So we have shown that there exists a prime factorization for every natural number $\geq 2$. □

After showing that $\mathcal{R}$ is a well-order, the proof was amazingly short. Now you might ask: can I really trust this? Is there a proof for well-founded induction? Or is it again just something we assume? It turns out that we can really prove well-founded induction. To show that our proof rule is correct, we need to show that what is below the line implies what is above the line, i.e. $\forall x \in \mathbb{X} : P(x)$. So more formally, the lemma looks like this:

**Lemma 5.26** (Well-Founded Induction)**.** *Let $\mathbb{X}$ be a set and $\mathcal{R} \subseteq \mathbb{X}^2$ a well-order. Assuming*

$$\forall x \in \mathbb{X} : (\forall y \in \mathbb{X} : (y, x) \in \mathcal{R} \land y \neq x \rightarrow P(y)) \rightarrow P(x),$$

$\forall x \in \mathbb{X} : P(x)$ *holds.*

*Proof.* Let $\mathbb{X}$ be a set and $\mathcal{R} \subseteq \mathbb{X}^2$ a well-order with

$$\forall x \in \mathbb{X} : (\forall y \in \mathbb{X} : (y, x) \in \mathcal{R} \land y \neq x \rightarrow P(y)) \rightarrow P(x).$$

We call this assumption $H$. Now assume $\forall x \in \mathbb{X} : P(x)$ did not hold. Then there is a non-empty set of "troublemakers" $S := \{x \in \mathbb{X} \mid \neg P(x)\}$. Since $\mathcal{R}$ is well-founded, there is a minimal element $x \in S$. Our goal is to derive a contradiction, concretely that there are no troublemakers. To that end, it suffices to show that there is no minimal element in the set of troublemakers, i.e. that $P(x)$ holds.

Now remember that $H$ corresponds to the proof that one has to write when applying well-founded induction. It says that if $P(y)$ holds for all elements $y \in \mathbb{X}$ that are "smaller" than $x$, then $P(x)$ holds as well. So when we apply $H$, we only have to show

$$\forall y \in \mathbb{X} : (y, x) \in \mathcal{R} \land y \neq x \rightarrow P(y).$$

Let $y \in \mathbb{X}$. Assume $(y, x) \in \mathcal{R}$ and $y \neq x$, i.e. that $y$ is "smaller" than $x$. Now if $P(y)$ holds, then we also have shown that $P(x)$ holds. If instead $P(y)$ does not hold, then $y$ is a troublemaker as well, formally $y \in S$. But since $y$ is "smaller" than $x$, $x$ cannot be a minimal element of $S$. Thus, we have shown that there are no troublemakers and $P(x)$ holds for all $x \in \mathbb{X}$. □

**Checkpoint 5.27:** Well-founded Induction Only Works If . . .

What is wrong with the following proof?

*Proof that $\forall z \in \mathbb{Z} : z > 42$.* By well-founded induction with $\leq$, we know that

$$\forall z' : z' \leq z \land z' \neq z \rightarrow z' > 42.$$

We instantiate the induction hypothesis with $z - 1$ and obtain

$$z - 1 \leq z \land z - 1 \neq z \rightarrow z - 1 > 42.$$

Since $z - 1 \leq z$ and $z - 1 \neq z$ hold trivially, we have $z - 1 > 42$. But since $z > z - 1$, we ultimately have $z > 42$. □

## 5.6 Summary

In this chapter, we have seen induction as an extremely useful technique to prove properties of infinite sets. We considered four variants of induction, where well-founded induction is the most-general. The other ones can be viewed as specializations thereof. In the following diagram, there are our four variants with their proof rules (only the obligations as a formula in first-order logic) as well as their relationship. Of course, the proof rule of structural induction depends on the inductive definition we consider, here it's just the natural numbers.

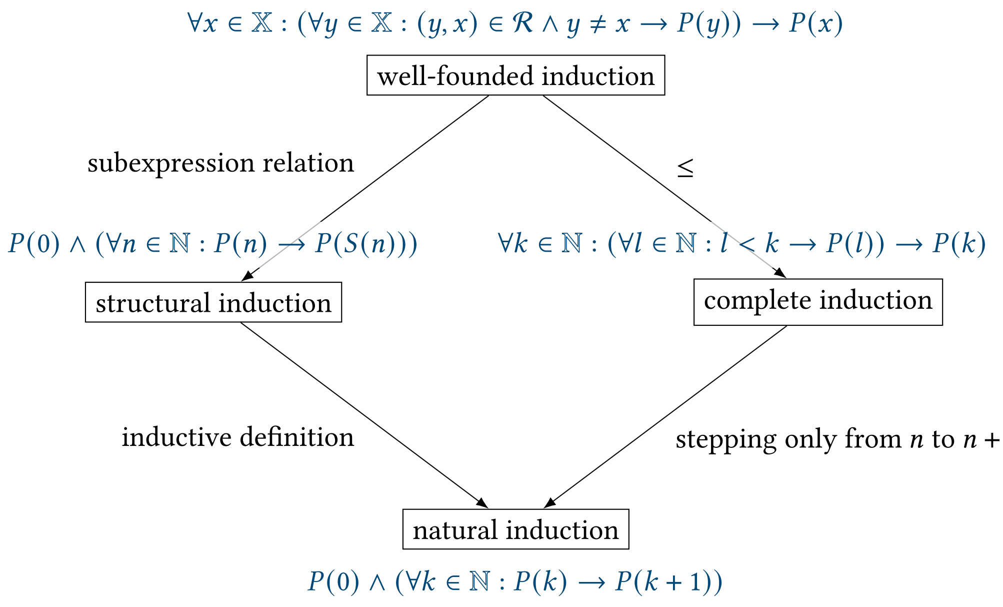

$\forall x \in \mathbb{X} : (\forall y \in \mathbb{X} : (y, x) \in \mathcal{R} \land y \neq x \to P(y)) \to P(x)$
well-founded induction
subexpression relation
$\leq$
$P(0) \land (\forall n \in \mathbb{N} : P(n) \to P(S(n)))$
$\forall k \in \mathbb{N} : (\forall l \in \mathbb{N} : l < k \to P(l)) \to P(k)$
structural induction
complete induction
inductive definition
stepping only from $n$ to $n + 1$
natural induction
$P(0) \land (\forall k \in \mathbb{N} : P(k) \to P(k + 1))$

Figure 5.28: Overview of the induction variants

Furthermore, you know that sometimes a proposition cannot be proved directly but a stronger one can. In this context, we discussed tail-recursive functions and quantified inductive hypotheses.

*You have now reached the end of the mathematics preparatory course. Take a moment to think about all the different things that we discussed: we started with formal languages in order to define expressions and assign meaning to them. Then, we learned to reason using propositional and first-order logic, as well as how to write correct proofs. After that, we took a tour through the land of sets and relations before concluding with another proof technique, induction.*

*You will very likely encounter many of these topics again during your studies, some earlier, some later. Feel free to come back to this book whenever you need a refresher on any of these topics.*

*Finally, we would like to take this moment to thank all the people that made the prep course possible, beyond the writing of this book. All the work that goes into each iteration, from organizing rooms to preparing lectures and creating materials, is conducted voluntarily by the team members, who give up a large part of their spare time before and during the course to provide you with the best possible experience. If you liked this preparatory course, we will all greatly appreciate if you consider helping out in the next iterations. But for now, we wish you a good start into your studies!*

*This page is intentionally left blank.*

# A | Natural Numbers

Natural numbers have been around you since you learned to count. By now, you probably know how to use them for calculating basic things. But having formalized a logic and a proof system, how can we use these new tools to prove claims about (natural) numbers?

One possibility is to formalize them as well. There is a rigorous axiomatic approach to this, first proposed by the mathematician Giuseppe Peano in 1889, see Page 164. While this rigorous approach is rather beautiful, it can be hard to work with, since all the properties we would expect to hold need to be proven from first principles, which takes a long time and thus is outside the scope of this book.

Since we still want to be able to handle numbers in our proofs, we cheat a little bit and only define the properties that we specifically need for the tasks in this prep course. Furthermore, we presume some knowledge about the very basic laws natural numbers follow.

> **Definition A.1** (Natural Numbers). *For our needs, **natural numbers** simply are distinct objects labelled* $0, 1, 2, 3, \ldots$ *There are infinitely many natural numbers, and together they form the set* $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

There always is the discussion whether $0$ is a natural number or if $1$ is the first one. Both can have their advantages because they make certain definitions a little cleaner but, in general, this does not matter too much. In computer science, the natural numbers typically include $0$, as they do in this book.

Next, we introduce the basic operations on natural numbers.

**Addition of Natural Numbers** The operator $+$ denotes addition.

**Fact A.2** (Properties of Addition). *The addition of natural numbers has the following two basic properties:*

*(a)* $+$ *is **commutative**, that is, for all natural numbers* $a, b \in \mathbb{N}$ *it holds that*

$$a + b = b + a.$$

*(b)* $+$ *is **associative**, that is, for all natural numbers* $a, b, c \in \mathbb{N}$ *it holds that*

$$(a + b) + c = a + (b + c)$$

Furthermore, $0$ is the **neutral element** with respect to addition:

**Fact A.3** (Neutral Element of Addition). $\forall n \in \mathbb{N} : n + 0 = n$

The proofs are omitted here as it would require deeper formalization of natural numbers. For this prep course, it suffices to simply *know* these properties.

Similarly, we can define the second important operation on natural numbers:

**Multiplication of Natural Numbers** The operator $\cdot$ denotes multiplication. Oftentimes, when working with variables, we simply omit the dot, such that $a \cdot b$ is the same as $ab$.

**Fact A.4** (Properties of Multiplication)**.** *Multiplication of natural numbers has the following two basic properties:*

*(a)* $\cdot$ *is* ***commutative****, that is, for all natural numbers* $a, b \in \mathbb{N}$ *it holds that*

$$ab = ba.$$

*(b)* $\cdot$ *is* ***associative****, that is, for all natural numbers* $a, b, c \in \mathbb{N}$ *it holds that*

$$(ab)c = a(bc).$$

The **neutral element** of multiplication is 1:

**Fact A.5** (Neutral Element of Multiplication)**.** $\forall n \in \mathbb{N} : n \cdot 1 = n$

Now, + and $\cdot$ together fulfill the **distributive property**:

**Fact A.6** (Distributivity)**.** *Let* $a, b, c \in \mathbb{N}$ *natural numbers.*

*(a)* $a(b + c) = ab + ac.$

*(b)* $(a + b)c = ac + bc$

Additionally, all the other rules you happen to know from school hold (e.g. $\forall n \in \mathbb{N} : n \cdot 0 = 0$) and whenever we use them, we do so implicitly.

You might ask yourself now, what's up with $-$ and $\div$, why are they not introduced the same way? You can subtract natural numbers, or at least, some of them, right? Well, yes, but actually no. In order to properly define subtraction, you would need to expand to the integers ($\mathbb{Z}$) to account for negative numbers, for example when subtracting 6 from 5. Similarly, you need the rational numbers ($\mathbb{Q}$) to give a proper definition of division. Furthermore, the concept of equivalence relations (see Section 4.2.4) is required to formally construct these numbers.

Another important aspect when you talk about numbers, in general, is their order. We want to be able to say that a number is greater, less than, or equal to another number.

**Definition A.7** (Less-Equal)**.** *The natural numbers are ordered by the total order* $\leq$*, defined as follows:*

$$a \leq b :\equiv \exists k \in \mathbb{N} : b = a + k$$
$$a < b :\equiv a \leq b \land a \neq b$$

*When* $a < b$*, we say that* $a$ *is* ***less than*** $b$*. Similarly, if* $a \leq b$*, then* $a$ *is* ***less or equal to*** $b$*. When we flip the order, we get* $>$*, called* ***greater than****.*

In proofs, one sometimes needs to do a case distinction on how two numbers relate to each other (i.e. if they are the same or one is less or greater than the other). This is what **trichotomy** is about. In general, a trichotomy is a splitting into three parts, and this is what we do here as well:

**Fact A.8** (Trichotomy)**.** *Let $a, b \in \mathbb{N}$ be two natural numbers. Then* exactly one *of the following statements holds:*

*(a)* $a < b$

*(b)* $a = b$

*(c)* $a > b$

Finally, we introduce the concept of factors and divisors. Again, note that this alone does not enable us to divide natural numbers arbitrarily.

**Definition A.9** (Factor, Divisor)**.** *Let $n \in \mathbb{N}$ be a natural number. We call a number $x \in \mathbb{N}, x \leq n$ a* ***factor*** *or a* ***divisor*** *of $n$ if and only if there is another natural number that, multiplied with $x$, results in $n$. Formally, this means*

$$x \mid n \iff \exists k \in \mathbb{N} : xk = n.$$

*We also say that $n$ is divisible by $x$. In the case that $x$ is not a factor of $n$, we write*

$$x \nmid n \quad \textit{or} \quad \neg x \mid n.$$

From this definition, we can derive two simple properties of natural numbers that you are probably already familiar with.

**Definition A.10** (Even, Odd)**.** *Let $n \in \mathbb{N}$ be a natural number.*

*(a) We call $n$* **even** *if and only if $2 \mid n$, that is, $2$ is a factor of $n$.*

*(b) We call $n$* **odd** *if and only if $2 \mid n + 1$, that is, $2$ is a factor of $n + 1$.*

*We can also give the respective predicates, that is*

$$\begin{aligned} \mathit{even}(n) &:= 2 \mid n, \\ \mathit{odd}(n) &:= 2 \mid n + 1. \end{aligned}$$

At last, we state some useful facts that you might use when reasoning about natural numbers. You are encouraged to prove these yourself as an exercise.

**Fact A.11** (Properties of even and odd numbers)**.** *Let $n \in \mathbb{N}$ be a natural number. Then the following holds:*

*(a)* $\mathit{even}(n) \lor \mathit{odd}(n)$,

*(b)* $\mathit{even}(n) \leftrightarrow \neg \mathit{odd}(n)$.

*This page is intentionally left blank.*

# Index