

Mathematics 2 (Formal Proving)

Wolfgang Schreiner
Engineering for Computer-based Learning
University of Applied Sciences at Hagenberg

Wolfgang.Schreiner@fh-hagenberg.at

June 6, 2005

Abstract

We present the rules for rigorous reasoning (i.e., formal proving) in the language of logic. Our discussion starts with reasoning about recursively defined domains, such as the natural numbers, by the principle of induction. We then proceed to general predicate-logic proofs with two basic strategies (direct and indirect proofs) and the two directions (the “top-down” decomposition of the goal to simpler goals and the “bottom-up” extension of the knowledge base by additional knowledge). The rules are demonstrated by examples from the first part of these lecture notes.

Contents

I	Recursive Definitions and Induction Proofs	5
1	Recursive Definitions	6
1.1	Recursive Function Definitions	7
1.2	Recursive Predicate Definitions	12
2	Numbers and Such	17
2.1	The Natural Numbers	17
2.1.1	Arithmetic Operations	20
2.1.2	Arithmetic Laws	21
2.1.3	Ordering	22
2.1.4	More Operations	23
2.1.5	Even More Operations	26
2.2	Minimum and Maximum	27
2.3	Sum Quantifier	29
2.4	Product Quantifier	32
2.5	Binomials	34
2.6	Matrix Operations	36
3	Induction	39
3.1	Inductive Definitions	39
3.2	Induction as a Proof Technique	43
3.3	Induction on Sets	50

II	Predicate Logic Proofs	57
4	Introduction	58
5	Preliminaries	61
5.1	Proof Situations	61
5.2	Proof Rules	62
5.3	Proof Termination	63
6	General Strategies	65
6.1	Direct Proofs	65
6.2	Indirect Proofs	66
6.3	Proof Directions	70
7	Decomposing the Goal	74
7.1	Quantifiers	74
7.1.1	Universal Quantifiers	74
7.1.2	Existential Quantifiers	78
7.2	Connectives	82
7.2.1	Equivalences	82
7.2.2	Implications	84
7.2.3	Conjunctions	87
7.2.4	Disjunctions	88
7.3	Defined Constants	90
7.3.1	Predicates	90
7.3.2	Functions	91
8	Deriving New Knowledge	95
8.1	Case Distinctions	95
8.2	Universal Formulas in Knowledge	98
8.3	Existential Formulas in Knowledge	100

8.4	Inferring Additional Knowledge	102
8.4.1	Propositional Consequences	104
8.4.2	Quantifier Consequences	106
8.4.3	Substitutions	108
9	Example	111
10	Further Proving Techniques	115
A	Logic Evaluator Definitions	117
A.1	Natural Numbers	117

Part I

Recursive Definitions and Induction Proofs

Chapter 1

Recursive Definitions

To proceed further with defining new notions, we need more powerful techniques. The definitions discussed so far were constrained by the fact that the new notion has to be defined in terms of already existing notions only. However, there are also situations in which it is necessary to reduce the value of a predicate or functions to other values of the same predicate or function. Such a *recursive* definition makes sense if this reduction does not continue forever, i.e., if eventually a situation is reached where the value can be defined directly in terms of other (previously introduced) notions.

Example The *factorial function* (*Faktorielle*) $fact : \mathbb{N} \rightarrow \mathbb{N}$ is defined as follows:

$$\begin{aligned} fact(n) &:= \\ &\mathbf{if} \ n = 0 \\ &\quad \mathbf{then} \ 1 \\ &\quad \mathbf{else} \ n * fact(n - 1) \end{aligned}$$

This definition is constructive in the sense that, for any concrete argument n , the function value $fact(n)$ can be determined by a finite number of “unfoldings” of the definition.

$$\begin{aligned} fact(4) &= 4 * fact(4 - 1) \\ &= 4 * (3 * fact(3 - 1)) \\ &= 4 * (3 * (2 * fact(2 - 1))) \\ &= 4 * (3 * (2 * (1 * fact(1 - 1)))) \\ &= 4 * (3 * (2 * (1 * 1))) = 24 \end{aligned}$$

However, not every recursive definition behaves in this way.

Example Take the function $foo : \mathbb{N} \rightarrow \mathbb{N}$ defined as follows:

$$\begin{aligned}
 foo(n) &:= \\
 &\mathbf{if} \ n = 0 \\
 &\quad \mathbf{then} \ 1 \\
 &\quad \mathbf{else} \ n * foo(n + 1)
 \end{aligned}$$

For $n > 0$, the value of $foo(n)$ can not be computed by a finite number of unfoldings of the definition:

$$\begin{aligned}
 &foo(4) \\
 &= 4 * foo(4 + 1) \\
 &= 4 * (5 * foo(5 + 1)) \\
 &= 4 * (5 * (6 * foo(6 + 1))) \\
 &= \dots
 \end{aligned}$$

It is therefore hard to see, which function (if any), this definition introduces.

The key for “well-behaved” recursive definitions is the fact that the argument gets “smaller” in every recursive invocation (in order to determine $fact(n) := n * fact(n - 1)$, for $n > 0$) and that the function value for the “smallest” argument is defined without recursion ($fact(0) := 1$). Both conditions together ensure that the total number of recursive unfoldings is bounded. The same idea can be applied to recursive predicate definitions.

1.1 Recursive Function Definitions

To generalize the concept of recursion to arbitrary argument domains (not just natural numbers), we introduce the following concept.

Definition 1 (Well-Founded Ordering) A binary relation \prec is well-founded if there is no infinitely decreasing chain with respect to \prec :

$$\prec \text{ is well-founded } :\Leftrightarrow \neg \exists S, s : \mathbb{N} \rightarrow S : \forall i \in \mathbb{N} : s_{i+1} \prec s_i.$$

$a \succ b :\Leftrightarrow b \prec a$.

Intuitively, in a well-founded ordering a chain $s_{i+2} \succ s_{i+1} \succ s_i \succ \dots \succ s_0$ is always terminated by a minimum element s_0 .

We now drop the constraint that the definiens must not refer to the definiendum.

Definition 2 (Recursive Function Definition) Let f be a function constant of arity n that is not yet in use, x_0, \dots, x_{n-1} be n distinct variables, and T be a term with no other free variables than x_0, \dots, x_{n-1} . Then

$$f(x_0, \dots, x_{n-1}) := T$$

is a *recursive function definition (rekursive Funktionsdefinition)*, if a term of the form $f(T_0, \dots, T_{n-1})$ occurs in T and we have a well-founded ordering \prec with the following property: for all x_0, \dots, x_{n-1} and every occurrence $f(T_0, \dots, T_{n-1})$ in T , it holds that

$$\langle T_0, \dots, T_{n-1} \rangle \prec \langle x_0, \dots, x_{n-1} \rangle$$

if the value of $f(x_0, \dots, x_{n-1})$ depends on the value of $f(T_0, \dots, T_{n-1})$.

This definition introduces an n -ary function constant f with the axiom

$$\forall x_0, \dots, x_{n-1} : f(x_0, \dots, x_{n-1}) = T.$$

Above definition requires the existence of a well-founded ordering such that in every recursive call $f(T_0, \dots, T_{n-1})$ the argument tuple is smaller than the argument tuple of the original invocation $f(x_0, \dots, x_{n-1})$.

The predicate “depends on” in above definition is a bit vague; for our purposes it suffices to state that the value of a term T depends on the value of every subterm of T , *unless* T is of the form

if F then T_0 else T_1 .

In this case, it only depends on the value of T_0 if F is true, and only on the value of T_1 , otherwise. A “definition” like

$$f(x) := 1 + f(x)$$

can therefore not make sense because the function result depends on the value of every subterm of the definiens.

The existence of a well-founded ordering ensures that for every argument only a finite number of recursive “unfoldings” of a function definition is required to determine the function value. Frequently, such an ordering is constructed by defining a corresponding *termination function* (*Terminationsfunktion*) r such that, for every x_0, \dots, x_{n-1} ,

$$r(x_0, \dots, x_{n-1}) \in \mathbb{N}.$$

Then the well-founded ordering is defined as

$$x \prec y :\Leftrightarrow r(x) < r(y)$$

where $<$ is the usual ordering of the natural numbers.

Example We define

$\text{any}(S) := \mathbf{such} \ x : x \in S$
 $\text{rest}(S) := \{x \in S : x \neq \text{any}(S)\}$
 $\#S := \text{the number of elements in a finite set } S.$

- The recursive definition

```

sum : FiniteSet → ℕ
sum(S) :=
  if S = ∅
  then 0
  else any(S) + sum(rest(S))

```

computes the sum of the elements of a finite set of numbers S ; a corresponding termination function is

$$r(S) := \#S.$$

For finite S , $\text{sum}(S) \in \mathbb{N}$. If $S = \emptyset$, no other application of ‘sum’ is needed to determine $\text{sum}(S)$. Otherwise, we need $\text{sum}(\text{rest}(S))$ with $r(\text{rest}(S)) = r(S) - 1 < r(S)$.

- The recursive definition

$$\begin{aligned}
 &* : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\
 &x * y := \\
 &\quad \mathbf{if} \ y = 0 \\
 &\quad \quad \mathbf{then} \ 0 \\
 &\quad \quad \mathbf{else} \ x + x * (y - 1)
 \end{aligned}$$

introduces multiplication over the natural numbers; a corresponding termination function is

$$r(x, y) := y.$$

For $x \in \mathbb{N}$ and $y \in \mathbb{N}$, $r(x, y) \in \mathbb{N}$. If $y = 0$, no other application of $*$ is needed to determine $x * y$. If $y \neq 0$, we need $x * (y - 1)$ with $r(x, y - 1) = y - 1 < y = r(x, y)$.

- The function defined as

$$\begin{aligned}
 &\text{dsum} : \text{FiniteSet} \times \text{FiniteSet} \rightarrow \mathbb{N} \\
 &\text{dsum}(A, B) := \\
 &\quad \mathbf{if} \ A = \emptyset \ \wedge \ B = \emptyset \ \mathbf{then} \\
 &\quad \quad 0 \\
 &\quad \mathbf{else if} \ A = \emptyset \ \mathbf{then} \\
 &\quad \quad \text{any}(B) + \text{dsum}(A, \text{rest}(B)) \\
 &\quad \mathbf{else if} \ B = \emptyset \ \mathbf{then} \\
 &\quad \quad \text{any}(A) + \text{dsum}(\text{rest}(A), B) \\
 &\quad \mathbf{else} \\
 &\quad \quad \text{any}(A) + \text{any}(B) + \text{dsum}(\text{rest}(A), \text{rest}(B))
 \end{aligned}$$

computes the sum of the elements of two finite sets A and B . A corresponding termination function is

$$r(A, B) := \#A + \#B.$$

Logic Evaluator We may define a function recursively by a statement

$$\mathbf{fun} \ f(x_0, \dots, x_{n-1}) \ \mathbf{recursive} \ R = T;$$

What function is defined here?

where R is a term that denotes the value of a termination function.

```

option silent = true; read set;

fun count(e, i: Nat) = +(i, 1);
fun #(S: Set) = reduce(count, S, 0);

fun size(S: Set) recursive #(S) =
  if(=(S, {}), 0,
    let(e = such(x in S: true, x): +(1, size(--(S, {})(e)))));

fun S(n) = set(x in nat(1, n): true, x);

term #(S(10));
> 10.
term size(S(10));
> 10.

term size(--(S(12), S(5)));

```

The evaluator checks that in every recursive invocation function this term is appropriately decreased (unless the potentially dangerous statement `option check = false` is executed). Some more definitions are shown below.

```

fun f(x) recursive x = +(1, f(x));
> function f/1.
term f(3);
> ERROR: new induction value 3 of function f/1 is not less than old
value 3.

fun *(x: Nat, y: Nat) recursive y =
  if(=(y, 0), 0, +(x, *(x, -(y, 1))));
> function */2.
term *(7, 8);
> 56.

read set;
> file 'set.txt' read.
fun any(S: Set) = such(x in S: true, x);
> function any/1.
fun rest(S: Set) =
  let(e = any(S): set(x in S: not(=(e, x)), x));
> function rest/1.
fun prod(S: Set) recursive #(S) =
  if(=(S, {}), 1, *(any(S), prod(rest(S))));
> function prod/1.
term prod(set(x in nat(1, 5): true, x));
> 120.

term prod(set(x in nat(1, 5): true, -(x, 1)));

```

1.2 Recursive Predicate Definitions

Also a predicate can be defined in a recursive way.

Definition 3 (Recursive Predicate Definition) Let p be a predicate constant of arity n that is not yet in use, x_0, \dots, x_{n-1} be n distinct variables, and F be a formula and with no other free variables than x_0, \dots, x_{n-1} . Then

$$p(x_0, \dots, x_{n-1}) :\Leftrightarrow F$$

is a *recursive predicate definition* (*rekursive Prädikatsdefinition*) if a formula of the form $p(T_0, \dots, T_{n-1})$ occurs in F and we have a well-founded ordering \prec with the following property: for all x_0, \dots, x_{n-1} , we have

$$\langle T_0, \dots, T_{n-1} \rangle \prec \langle x_0, \dots, x_{n-1} \rangle.$$

for every occurrence of $p(T_0, \dots, T_{n-1})$ in F on which the truth value of $p(x_0, \dots, x_{n-1})$ depends.

This definition introduces an n -ary predicate constant p with the axiom

$$\forall x_0, \dots, x_{n-1} : p(x_0, \dots, x_{n-1}) \Leftrightarrow F.$$

As for functions, the predicate “depends on” needs some refinement. We introduce a new kind of formula.

Definition 4 (Conditional Formula) For every formula F and terms F_0 and F_1 , the phrase

$$(\mathbf{if } F \mathbf{ then } F_0 \mathbf{ else } F_1)$$

is a formula whose truth value is the value of F_0 , if F holds, and the value of F_1 , otherwise. We omit the parentheses, if F_1 is clear.

By above definition, we have the relationship

$$(\mathbf{if } F \mathbf{ then } F_0 \mathbf{ else } F_1) \Leftrightarrow ((F \Rightarrow F_0) \wedge (\neg F \Rightarrow F_1)).$$

We now state that the truth value of a formula F depends on the truth value of every subformula of F , *unless* F is of the form

if F **then** F_0 **else** F_1 .

In this case, it only depends on the truth value of F_0 if F is true, and only on the truth value of F_1 , otherwise.

Example

- The definition

$$\begin{aligned} \text{iseven} &\subseteq \mathbb{N} \\ \text{iseven}(x) &:\Leftrightarrow \\ &\mathbf{if} \ x = 0 \\ &\quad \mathbf{then} \ \mathbf{T} \\ &\quad \mathbf{else} \ \neg\text{iseven}(x - 1) \end{aligned}$$

introduces a unary predicate on natural numbers; a corresponding termination function is

$$r(x) := x.$$

- The definition

$$\begin{aligned} \leq &\subseteq \mathbb{N} \times \mathbb{N} \\ x \leq y &:\Leftrightarrow \\ &\mathbf{if} \ x = 0 \ \mathbf{then} \ \mathbf{T} \\ &\quad \mathbf{else} \ \mathbf{if} \ y = 0 \ \mathbf{then} \ \mathbf{F} \\ &\quad \mathbf{else} \ x - 1 \leq y - 1 \end{aligned}$$

introduces the usual ordering on natural numbers; a corresponding termination function is

$$r(x, y) := x.$$

Logic Evaluator Similar to recursive function definitions, predicates may be recursively defined by a statement

$$\text{pred } p(x_0, \dots, x_{n-1}) \text{ recursive } R \Leftrightarrow F;$$

where R is a term that denotes the value of a termination function. For writing recursive predicates, we also need the conditional formula

$$\text{if}(F, F_0, F_1)$$

whose value is **(if F then F_0 else F_1)**.

```

pred iseven(x: Nat) recursive x <=>
  if(=(x, 0), true, not(iseven(-(x, 1))));
> predicate iseven/1.
formula iseven(0);
> true.
formula iseven(1);
> false.
formula iseven(17);
> false.
pred <=(x: Nat, y: Nat) recursive x <=>
  if(=(x, 0), true,
    if(=(y, 0), false,
      <=(-(x, 1), -(y, 1))));
> predicate <=/2.
formula <=(0, 3);
> true.
formula <=(10, 5);
> false.
formula <=(5, 10);

```

Reduction of Sets Frequently functions over sets are recursively defined as follows:

$$\begin{aligned}
 g(S) &:= \\
 &\text{if } S = \emptyset \text{ then } b \\
 &\text{else let } e = \text{such } x : x \in S : \\
 &\quad f(e, g(S - \{e\})).
 \end{aligned}$$

We call this a *reduction (Reduktion)* of S by f with base b . If f is commutative and associative, the result of a reduction is uniquely defined.

Example Let $S := \{1, 2, 3\}$. Then the reduction of S by $+$ with base 0 is

$$6 = (1 + (2 + 3)) + 0 = (2 + (1 + 3)) + 0 = (2 + 1) + (3 + 0).$$

The reduction of S by $*$ with base 1 is

$$6 = (1 * (2 * 3)) * 1 = (2 * (1 * 3)) * 1 = ((1 * 1) * 2) * 3.$$

The Logic Evaluator provides the function

`reduce(f , S , b)`

which returns the reduction of S by f with base b which is more efficiently evaluated than a corresponding recursive function defined by the user (compare `sum` and `sum'` in the example below).

```
option silent = true; fun sum(S) = reduce(+, S, 0);
fun product(S) = reduce(*, S, 1);
fun S(n) = set(x in nat(1, n): true, x);
term S(10);
> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.
term sum(S(10));
> 55.
term product(S(10));
> 3628800.

fun +1(x, y) = +(1, y);
fun #(S) = reduce(+1, S, 0);
term #(S(10));
> 10.

fun sum'(S) recursive #(S) =
  if(=(S, {}), 0,
    let(e = such(x in S: true, x):
      +(e, sum'(set(x in S: not(=(x, e)), x))));
  term sum'(S(10));
> 55.
```

```
term sum'(S(100));
```

With the help of this operator also set union and powerset can be implemented as follows:

```
option silent = true;
fun ++(A: Set, B: Set) =
  reduce(join, A, B);
fun combine(e, S: Set) =
  ++(S, set(x in S: true, join(e, x)));
fun Powerset(S: Set) =
  reduce(combine, S, join({}, {}));
fun S(m, n) = set(x in nat(m, n): true, x);
term ++(S(1, 10), S(5, 15));
> {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 2, 3, 4}.
term Powerset(S(1, 4));
> {{1, 2, 3, 4}, {2, 3, 4}, {1, 3, 4}, {3, 4}, {1, 2, 4}, {2, 4},
{1, 4}, {4}, {1, 2, 3}, {2, 3}, {1, 3}, {3}, {1, 2}, {2}, {1}, {}}.
term combine(1, set(x in Powerset(S(2, 4)): true, x));
```

Chapter 2

Numbers and Such

In this chapter, we present *arithmetic* notions, i.e., notions that arise when dealing with numbers. As a starting point, we formally introduce the set of natural numbers \mathbb{N} on which other number domains like the integer numbers \mathbb{Z} , the rational numbers \mathbb{Q} , the real numbers \mathbb{R} , and the complex numbers \mathbb{C} can be based.

2.1 The Natural Numbers

Intuitively, the natural numbers are the numbers of *counting* distinct objects: there may be no object, one object, two objects, \dots . While the domain of natural numbers looks very simple and familiar, it is surprisingly powerful; e.g. every computer program can be encoded as a natural number. However, only *discrete* realities can be described by natural numbers; there are also *continuous* realities where the distinction between objects is so “fine” that they cannot be modeled by this domain.

Peano Arithmetic We may consider the natural numbers as an elementary domain by a theory that is due to the mathematician Peano.

Axiom 1 (Peano Arithmetic) The theory of natural numbers has an object constant 0 (*zero*) and a unary function constant $'$, the *successor* (*Nachfolger*), that satisfy the following axioms.

1. 0 is not the successor of any natural number:

$$\forall x : x' \neq 0.$$

2. Different natural numbers have different successors:

$$\forall x, y : x' = y' \Rightarrow x = y.$$

3. For every formula F , we have an axiom that states:

F holds for every natural number, if F holds for 0 and with every number also for its successor, i.e.,

$$(F[x \leftarrow 0] \wedge (\forall x : F \Rightarrow F[x \leftarrow x'])) \Rightarrow \forall x : F.$$

We call these axioms the *induction axioms* (*Induktionsaxiome*).

The Peano axioms essentially state that the natural numbers are a single infinite chain

$$0 \rightarrow 0' \rightarrow 0'' \rightarrow 0''' \rightarrow \dots$$

We will see later how all operations on the natural numbers can be defined just on the basis of 0 and $'$.

Construction from Sets It is not necessary to consider the natural numbers as elementary objects; they can be *defined* within the framework of set theory that provides a more fundamental kind of objects.

Definition 5 (Natural Numbers from Set) We define a constant 0 and a unary function $'$ as follows:

$$\begin{aligned} 0 &:= \emptyset; \\ x' &:= x \cup \{x\}. \end{aligned}$$

The first two axioms of Peano arithmetic are consequences of this definition.

Proof We sketch the proof of the first two laws of Peano.

- We prove $\forall x : x' \neq 0$. Take arbitrary x . By definition of 0 and $'$, we have to prove

$$x \cup \{x\} \neq \emptyset$$

which is true because $x \in (x \cup \{x\})$ but $x \neq \emptyset$.

- We prove $\forall x, y : x' = y' \Rightarrow x = y$. Take arbitrary x and y . We assume

$$(1) x' = y'$$

and have to show $x = y$. We assume (2) $x \neq y$ and show a contradiction. From (1), we know by the definition of $'$

$$(3) x \cup \{x\} = y \cup \{y\}.$$

By the definition of \cup and set enumeration, we know (4) $x \in x \cup \{x\}$ and (5) $y \in y \cup \{y\}$. From (3), (4), and the definition of $=$, we have (6) $x \in y \cup \{y\}$ which implies with (2)

$$(7) x \in y.$$

Likewise, we have from (3), (5), and the definition of $=$ that (8) $y \in x \cup \{x\}$ which implies with (2)

$$(9) y \in x.$$

The conjunction of (7) and (9) is in contradiction to the set-theoretic axiom of *regularity* (*Regularität*) that prohibits such “cycles” (in general any infinitely descending chains) with respect to \in .

The construction of natural numbers proceeds with the definition of a set \mathbb{N} (which we do not give) such that the following proposition holds.

Proposition 2 (Natural Numbers) \mathbb{N} , the set of *natural numbers* (*natürliche Zahlen*), is the smallest set that satisfies the following properties:

$$\begin{aligned} &0 \in \mathbb{N}; \\ &\forall x \in \mathbb{N} : x' \in \mathbb{N}; \\ &\forall F : \\ &\quad (F(0) \wedge \forall x \in \mathbb{N} : F(x) \Rightarrow F(x')) \Rightarrow \forall x \in \mathbb{N} : F(x) \end{aligned}$$

where the quantified variable F denotes a unary relation.

Thus also the third law of Peano holds in the set-based construction of natural numbers. Sometimes the following notations will be used.

Definition 6 (Natural Number Subsets)

$$\begin{aligned}\mathbb{N}_{>0} &:= \{x \in \mathbb{N} : x > 0\}; \\ \mathbb{N}_n &:= \{x \in \mathbb{N} : x < n\}.\end{aligned}$$

Independently of how we have introduced the natural numbers, we can define the basic arithmetic operations with the help of the functions 0 and ' and the following auxiliary operation.

Definition 7 (Predecessor) The *predecessor* (*Vorgänger*) of a natural number x is the natural number whose successor is x :

$$x^- := \mathbf{such} \ y : x = y'.$$

Because of the second Peano law, the predecessor is uniquely defined (if it exists). However, because of the first Peano law, there is no x such that $x' = 0$ and thus 0^- is undefined (and we must not assume $0^{-'} = 0$).

2.1.1 Arithmetic Operations

We now introduce the arithmetic operations on natural numbers by 1. In the rest of this section, all variables denote natural numbers.

Definition 8 (Natural Numbers Operations) We define the following functions and predicates (for all recursive definitions, a corresponding termination function is $r(x, y) := y$).

Constants

$$1 := 0', \quad 2 := 1';$$

Addition

$$x + y := \mathbf{if} \ y = 0 \ \mathbf{then} \ x \ \mathbf{else} \ (x + y^-)'$$

Multiplication

$$x * y := \text{if } y = 0 \text{ then } 0 \text{ else } x + (x * y^-)$$

Total Order

$$\begin{aligned} x \leq y &:\Leftrightarrow \\ &\text{if } x = 0 \text{ then T} \\ &\text{else if } y = 0 \text{ then F} \\ &\text{else } x^- \leq y^- \end{aligned}$$

The recursive definitions are constructive in the sense that they can be executed by repeated “unfolding” of the definition.

Example We compute $(1+2)$:

$$\begin{aligned} 1 + 2 &= (\text{definitions of 1 and 2}) \\ 0' + 0'' &= (\text{definition of } +) \\ (\text{if } 0'' = 0 \text{ then } 0' \text{ else } (0' + 0''^-)') &= (\text{first Peano law}) \\ (0' + 0''^-)' &= (\text{definition of } ^-) \\ (0' + 0')' &= (\text{definition of } +) \\ (\text{if } 0' = 0 \text{ then } 0' \text{ else } (0' + 0'^-)')' &= (\text{first Peano law}) \\ (0' + 0'^-)'' &= (\text{definition of } ^-) \\ (0' + 0)'' &= (\text{definition of } +) \\ (\text{if } 0 = 0 \text{ then } 0' \text{ else } (0' + 0^-)')'' &= (\text{reflexivity of } =) \\ &0'''. \end{aligned}$$

2.1.2 Arithmetic Laws

Immediate consequences of these definitions are the following laws.

Proposition 3 (Natural Numbers Operations) For all natural numbers x and y , we have:

Addition

$$\begin{aligned} x + 0 &= x, \\ x + y' &= (x + y)'; \end{aligned}$$

Multiplication

$$\begin{aligned}x * 0 &= 0, \\x * y' &= x + (x * y);\end{aligned}$$

Total Order

$$\begin{aligned}0 \leq x &\Leftrightarrow \text{T}, \\x \leq 0 &\Leftrightarrow x = 0, \\x' \leq y' &\Leftrightarrow x \leq y.\end{aligned}$$

Furthermore, the following well known laws hold.

Proposition 4 (Natural Number Laws) For all natural numbers x, y, z , we have:

$$\begin{aligned}x + 0 &= x, \\x * 1 &= x, \\x + y &= y + x, \\x * y &= y * x, \\x + (y + z) &= (x + y) + z, \\x * (y * z) &= (x * y) * z, \\x * (y + z) &= (x * y) + (x * z), \\x &\leq x, \\(x \leq y \wedge y \leq x) &\Rightarrow x = y, \\(x \leq y \wedge y \leq z) &\Rightarrow x \leq z.\end{aligned}$$

2.1.3 Ordering

We will use the following abbreviations for every domain that provides a binary predicate \leq .

Definition 9 (Order Predicates)

$$\begin{aligned}x < y &:\Leftrightarrow x \leq y \wedge x \neq y; \\x > y &:\Leftrightarrow x \not\leq y; \\x \geq y &:\Leftrightarrow x \not< y.\end{aligned}$$

We often write $a \leq x < b$ to denote $a \leq x \wedge x < b$ and similar for all other combinations of the order predicates.

Logic Evaluator Although there is a built-in implementation of the natural numbers, we are going to simulate the construction on the base of 0 and and $'$; we use the letter N to distinguish the natural number function and predicate constants from those that are going to be introduced in the following sections. These and the subsequent definitions are collected in Appendix A.1 (file `natural.txt`).

```

pred N(x) <=> Nat(x);
> predicate N/1.

fun N0 = 0;
> function N0/0.
fun '(x: N) = +(x, 1);
> function '/1.
fun ^-(x: N) = such(n in nat(0, x) : =(x, '(n)), n);
> function ^-/1.

fun N1 = '(N0);
> function N1/0.
fun N2 = '(N1);
> function N2/0.

fun +N(x: N, y: N) recursive y =
  if(=(y, N0), x, '(+N(x, ^-(y))));
> function +N/2.
fun *N(x: N, y: N) recursive y =
  if(=(y, N0), N0, +N(x, *N(x, ^-(y))));
> function *N/2.
pred <=N(x: N, y: N) recursive y <=>
  if(=(x, N0), true, if(=(y, N0), false, <=N(^-(x), ^-(y))));
> predicate <=N/2.

term ^-(0);
> ERROR: no such value.
term +N(5, 2);
> 7.
term *N(5, 3);
> 15.

formula <=(5, 3);

```

2.1.4 More Operations

Since we have addition, we would also like to have the inverse operation.

Definition 10 (Difference) z is a difference of x and y if $x = y + z$.

$$x - y := \mathbf{such} \ z : x = z + y.$$

Please note that a difference between two natural numbers does not always exist, e.g. $1 - 2$ is undefined because there is no z with $1 = z + 2$ (and we must not assume $1 = (1 - 2) + 2$). However, we have the following result.

Proposition 5 (Difference) If a difference exists, it is unique:

$$\forall x, y, z_0, z_1 : (x = z_0 + y \wedge x = z_1 + y) \Rightarrow z_0 = z_1.$$

If $x \geq y$, the difference of x and y is defined:

$$\forall x, y : x \geq y \Rightarrow x = (x - y) + y.$$

At the moment, we state these propositions without proofs. In Chapter 3, we will discuss in detail how to prove properties about natural numbers.

While the natural numbers do not have an operation for exact division, they provide the following pair of functions.

Definition 11 (Quotient and Remainder) The *quotient* (*Quotient*) and *remainder* (*Rest*) of two natural numbers are defined as follows:

$$\begin{aligned} x \operatorname{div} y &:= \mathbf{such} \ q : \exists r : r < y \wedge x = (q * y) + r; \\ x \operatorname{mod} y &:= \mathbf{such} \ r : \exists q : r < y \wedge x = (q * y) + r. \end{aligned}$$

By above definition $(x \operatorname{div} 0)$ and $(x \operatorname{mod} 0)$ are undefined for every x . On the other side, we have the following positive results.

Proposition 6 (Quotient and Remainder) If quotient respectively remainder of two numbers exist, they are unique.

$$\begin{aligned} \forall x, y, q_0, q_1, r_0 < y, r_1 < y : \\ (x = (q_0 * y) + r_0 \wedge x = (q_1 * y) + r_1) \Rightarrow (q_0 = q_1 \wedge r_0 = r_1). \end{aligned}$$

If the divisor is not null, quotient and remainder exist:

$$\forall x, y \neq 0 : (\exists q, r : r < y \wedge x = (q * y) + r).$$

We thus have the following relationship between quotient and remainder:

$$\forall x, y \neq 0 : x = (x \text{ div } y) * y + (x \text{ mod } y).$$

We define exponentiation as shown below.

Definition 12 (Exponentiation) We define recursively with termination function $r(x, n) := n$:

$$\begin{aligned} \cdot : \mathbb{N} \times \mathbb{N} &\rightarrow \mathbb{N}, \\ x^n &:= \text{if } n = 0 \text{ then } 1 \text{ else } x * x^{n-1}. \end{aligned}$$

A corresponding construction can be applied in every domain D with a constant 1 and a binary function $*$ such as the number domains that will be introduced in the following sections; we will then assume the existence of a corresponding function $\cdot : D \times \mathbb{N} \rightarrow D$.

Logic Evaluator Difference, quotient, remainder and exponentiation are defined as shown below.

```
option silent = true; read natural;

fun -N(x: N, y: N) = such(z in nat(0, x) :=(x, +N(z, y)), z);
fun divN(x: N, y: N) =
  such(q in nat(0, x), r in nat(0, ^-(y)) :
    =(x, +N(*N(q, y), r)), q);
fun modN(x: N, y: N) =
  such(q in nat(0, x), r in nat(0, ^-(y)) :
    =(x, +N(*N(q, y), r)), r);
fun ^N(x: N, n: N) recursive n =
  if(=(n, N0), N1, *N(x, ^N(x, ^-(n))));

term -N(5, 3);
> 2.
term -N(3, 5);
> ERROR: no such value.
term divN(5, 3);
> 1.

term modN(5, 3);
```

2.1.5 Even More Operations

Other important notions on natural numbers can be introduced as follows.

Definition 13 (Natural Number Operations)

- x divides y if $x * z = y$ for some z :

$$x|y :\Leftrightarrow \exists z : x * z = y.$$

- The *greatest common divisor* (*größter gemeinsamer Teiler*) of x and y is the largest number that divides both x and y :

$$\text{gcd}(x, y) := \mathbf{such} \ z : z|x \wedge z|y \wedge (\forall w : (w|x \wedge w|y) \Rightarrow w \leq z).$$

- The *least common multiple* (*kleinstes gemeinsames Vielfaches*) of x and y is the smallest number that both x and y divide:

$$\text{lcm}(x, y) := \mathbf{such} \ z : x|z \wedge y|z \wedge (\forall w : (x|w \wedge y|w) \Rightarrow z \leq w).$$

- Two numbers are *relatively prime* (*relativ prim*) if their greatest common divisor is 1:

$$x \text{ and } y \text{ are relatively prime} :\Leftrightarrow \text{gcd}(x, y) = 1.$$

- A number greater than 1 is *prime* (*prim*) if its only divisors are 1 and itself:

$$x \text{ is prime} :\Leftrightarrow x > 1 \wedge (\forall y : y|x \Rightarrow (y = 1 \vee y = x)).$$

The definition of prime numbers is motivated by the fact that every positive natural number n has a unique prime number factorization (which will be stated formally in Proposition 9 on page 33).

Logic Evaluator Above notions are defined as shown below (to speed up computation, we we revert to built-in natural number arithmetic using the file `natural0.txt`).

```

option silent = true;
read natural0;

pred divides(x, y) <=> exists(z in nat(N0, y) : (=(*N(x, z), y)));
fun gcd(x, y) =
  let(m = if(=(x, N0), y, x) :
    such(z in nat(N0, m) :
      and(divides(z, x), divides(z, y),
        forall(w in nat(+N(z, N1), m) :
          or(not(divides(w, x)), not(divides(w, y))))),
      z));
fun lcm(x, y) = such(z in nat(N1, *N(x, y)) :
  and(divides(x, z), divides(y, z),
  forall(w in nat(x, -(z, N1)) :
    or(not(divides(x, w)), not(divides(y, w))))),
  z);
pred relprime(x, y) <=> =(gcd(x, y), N1);
pred isprime(x) <=>
  and(not(<=N(x, N1)),
  forall(y in nat(N0, x) :
    implies(divides(y, x), or(=(y, N1), =(y, x))));
formula divides(4, 12);
> true.

term gcd(6, 9);
> 3.
term lcm(10, 6);
> 30.
formula relprime(6, 4);
> false.

```

```
formula isprime(17);
```

2.2 Minimum and Maximum

The notions defined in this and in the following sections can be used in all domains that have constants $0, 1, +, *, \leq$ with “number-like” properties.

Definition 14 (Minimum and Maximum Quantifier) If x is a variable and F is a formula, then the following are terms with bound variable x :

$\min_x F$;
 $\max_x F$.

The value of the first term is the smallest value of x such that F holds; the value of the second term is the largest such value, i.e.,

$$\begin{aligned}\min_x F &:= \mathbf{such} \ x : F \wedge (\forall y : F[x \leftarrow y] \Rightarrow x \leq y); \\ \max_x F &:= \mathbf{such} \ x : F \wedge (\forall y : F[x \leftarrow y] \Rightarrow x \geq y).\end{aligned}$$

Usually the variable x is dropped and has to be deduced from the context. We can use this notion to define corresponding functions on sets as follows.

Definition 15 (Minimum and Maximum Functions)

$$\begin{aligned}\min(S) &:= \min_x x \in S; \\ \max(S) &:= \max_x x \in S;\end{aligned}$$

Please note that the definition of “min” and “max” depends on the choice of the predicate \leq . Furthermore, minimum and maximum are not necessarily defined.

Example

- We have

$$\max_x (\text{isprime}(x) \wedge x|100) = 5.$$

- The value of

$$\min(\{1/x : x \in \mathbb{N}_{>0}\})$$

is undefined because for every $z = 1/x$ in $\{1/1, 1/2, 1/3, 1/4, \dots\}$ there is always an y in this set with $y < z$, namely $1/(x+1)$.

2.3 Sum Quantifier

Frequently we want to construct the sum of a large collection of values. Instead of writing $f(1) + f(2) + \dots + f(n)$, we may write $\sum_{1 \leq i \leq n} f(i)$.

Definition 16 (Sum Quantifier) If x is a variable, F is a formula and T is a term, then the following is a term with bound variable x :

$$\sum_{x,F} T.$$

The value of this term is 0, if F does not hold for any x ; otherwise it is, for every x that satisfies F , the sum of the value of T and of the value of the term for all other x :

$$\begin{aligned} (\forall x : \neg F) &\Rightarrow \sum_{x,F} T = 0; \\ (\forall y : F[x \leftarrow y]) &\Rightarrow \sum_{x,F} T = T[x \leftarrow y] + \sum_{x,F \wedge x \neq y} T. \end{aligned}$$

Please note that the value of the term is only defined if there do not exist infinitely many x such that F holds.

Usually F contains the condition $x \in \mathbb{N}$, which is therefore not written explicitly. Also the variable x is usually not written but has to be deduced from the context. Frequently also the form

$$\sum_{i=a}^b T$$

is used to denote $\sum_{a \leq i \leq b} T$.

Example

$$\sum_{1 \leq i \leq n} i^2 = \sum_{i, (i \in \mathbb{N} \wedge 1 \leq i \wedge i \leq n)} i^2;$$

$$\sum_{1 \leq i \leq 5} i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2;$$

\sum ... “Sigma” (the capital Greek letter) used here for “Sum”.

$$\sum_{1 \leq i \leq 0} i^5 = 0;$$

$$\sum_{1 \leq i \leq 9} (x - i)^2 = (x - 1)^2 + \sum_{2 \leq i \leq 9} (x - i)^2;$$

$$\sum_{1 \leq i \leq n} x^i = \sum_{1 \leq i \leq n \wedge \text{even}(i)} x^i + \sum_{1 \leq i \leq n \wedge \text{odd}(i)} x^i.$$

Example Let $a := [3, 1, 2, 9, 0, 7]$. We have

$$\sum_{0 \leq i \leq 5} a_i * 10^i = 709213.$$

In general, for any finite sequence d of “decimal digits” the term

$$\sum_{0 \leq i < \text{length}(d)} d_i * 10^i$$

denotes the value of this sequence in the decimal number system. Likewise, for any finite sequence b of binary digits 0 and 1, the value

$$\sum_{0 \leq i < \text{length}(b)} b_i * 2^i$$

denotes the value of this sequence in the binary number system, e.g., the value of $[0, 1, 1, 0, 1]$ is

$$1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 22.$$

The quantifier may also bind multiple variables.

Example

$$\sum_{1 \leq i \leq 5, 1 \leq j \leq 3} i * j = 1 * 1 + 1 * 2 + 1 * 3 + \sum_{2 \leq i \leq 5, 1 \leq j \leq 3} i * j;$$

$$\sum_{1 \leq i \leq 3, 1 \leq j \leq i} i * j = 1 * 1 + 2 * 1 + 2 * 2 + 3 * 1 + 3 * 2 + 3 * 3.$$

We have a number of important identities.

Proposition 7 (Sum Quantifier) For all variables i and j and formulas F (in which j does not occur freely), G (in which i does not occur freely), and H and terms T and U we have:

$$\sum_{i,F} T * \sum_{j,G} U = \sum_{i,F} \sum_{j,G} T * U.$$

$$\sum_{i,F} \sum_{j,G} T = \sum_{j,G} \sum_{i,F} T = \sum_{i,j,F \wedge G} T.$$

$$\sum_{i,F} T + \sum_{i,H} T = \sum_{i,F \vee H} T + \sum_{i,F \wedge H} T.$$

Furthermore, if term C is a term in which i does not occur freely, we have:

$$\sum_{i,F} C * T = C * \sum_{i,F} T.$$

$$\sum_{i,F} C = n * C \text{ (where } n \text{ is the number of } i\text{'s for which } F \text{ holds).}$$

Example

$$\sum_{1 \leq i \leq n} x^i * \sum_{1 \leq j \leq m} x^j = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} x^{i+j} = \sum_{1 \leq i \leq n \wedge 1 \leq j \leq m} x^{i+j}.$$

$$\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} i * x^j = \sum_{1 \leq i \leq n} (i * \sum_{1 \leq j \leq m} x^j).$$

2.4 Product Quantifier

Likewise we often wish to construct the product of a collection of values. Instead of writing $f(1) * f(2) * \dots * f(n)$, we may write $\prod_{1 \leq i \leq n} f(i)$.

Definition 17 (Product Quantifier) If x is a variable, F is a formula and T is a term, then the following is a term with bound variable x :

$$\prod_{x, F} T.$$

The value of this term is 1, if F does not hold for any x ; otherwise it is, for every x that satisfies F , the product of the value of T and of the value of the term for all other x , i.e.,

$$\begin{aligned} (\forall x : \neg F) &\Rightarrow \prod_{x, F} T = 1; \\ (\forall y : F[x \leftarrow y]) &\Rightarrow \prod_{x, F} T = T[x \leftarrow y] * \prod_{x, F \wedge x \neq y} T. \end{aligned}$$

\prod ... “Pi” (the capital Greek letter) used here for “Product”.

The same remarks apply as for the definition of the sum quantifier.

Example

$$\prod_{1 \leq i \leq 0} i^5 = 1;$$

$$\prod_{0 \leq i \leq 5} a_i = a_0 * a_1 * a_2 * a_3 * a_4 * a_5;$$

$$\prod_{0 \leq i \leq 100} a_i = a_0 * a_1 * a_2 * \prod_{3 \leq i \leq 100} a_i;$$

$$\prod_{0 \leq i \leq 100} a_i = \prod_{0 \leq i \leq 50} a_i * \prod_{51 \leq i \leq 100} a_i;$$

$$\prod_{\substack{1 \leq i \leq 3 \\ 1 \leq j \leq i}} i + j = (1 + 1) * (2 + 1) * (2 + 2) * (3 + 1) * (3 + 2) * (3 + 3).$$

We have the following identities.

Proposition 8 (Product Quantifier) For all variables i and j and formulas F (in which j does not occur freely), G (in which i does not occur freely), and H and terms T and U we have:

$$\prod_{i,F} \prod_{j,G} T = \prod_{j,G} \prod_{i,F} T = \prod_{i,j,F \wedge G} T.$$

$$\prod_{i,F} T * \prod_{i,H} T = \prod_{i,F \vee H} T * \prod_{i,F \wedge H} T.$$

Furthermore, if term C is a term in which i does not occur freely, we have:

$$\prod_{i,F} C * T = C^n \prod_{i,F} T \text{ (where } n \text{ is the number of } i\text{'s for which } F \text{ holds).}$$

$$\prod_{i,F} C = C^n \text{ (where } n \text{ is the number of } i\text{'s for which } F \text{ holds).}$$

With the help of the product quantifier we can now formulate the uniqueness of prime number composition stated in Section 2.1.

Proposition 9 (Prime Number Factorization) For every natural number $n \neq 0$, there exists a unique prime number factorization:

$$\forall n \in \mathbb{N}_{>0} : (\exists p : \text{pf}(n, p) \wedge (\forall q : \text{pf}(n, q) \Rightarrow p = q)).$$

A prime number factorization of n is a sequence of pairs (p, e) ordered by the primes p such that n is the product of all p^e .

$$\begin{aligned} \text{pf}(n, p) : &\Leftrightarrow \\ &p : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N} \wedge \\ &(\forall i \in \mathbb{N} : p(i)_0 \text{ is prime} \wedge p(i)_0 < p(i+1)_0) \wedge \\ &(\exists k \in \mathbb{N} : n = \prod_{0 \leq i \leq k} p(i)_0^{p(i)_1} \wedge \forall i > k : p(i)_1 = 0) \end{aligned}$$

Example Because $300 = 2^2 * 3^1 * 5^2$, we have for 300 the prime number factorization

$$[\langle 2, 2 \rangle, \langle 3, 1 \rangle, \langle 5, 2 \rangle, \langle 7, 0 \rangle, \langle 11, 0 \rangle, \langle 13, 0 \rangle, \dots].$$

Likewise, 1 has the prime number factorization

$$[\langle 2, 0 \rangle, \langle 3, 0 \rangle, \langle 5, 0 \rangle, \langle 7, 0 \rangle, \langle 11, 0 \rangle, \langle 13, 0 \rangle, \dots].$$

2.5 Binomials

In combinatorics (the branch of mathematics that solves questions of the kind “how many objects do exist such that ...?”), the following notions are of importance.

How many roads must a man go down...?

Definition 18 (Factorial) The *factorial* (*Fakultät*) of a natural number n is the product of all non-zero numbers less than or equal n :

$$n! := \prod_{1 \leq i \leq n} i.$$

Why?

Please note that the definition of \prod implies $0! = 1$.

Definition 19 (Binomial Coefficient) The *binomial coefficient* (*Binomialkoeffizient*) of two natural numbers is defined as follows:

$$\binom{n}{k} := \text{if } 0 \leq k \leq n \text{ then } \frac{n!}{k! * (n - k)!} \text{ else } 0.$$

We read this as “ n choose k ” (“ n über k ”).

The name “ n choose k ” stems from the fact that $\binom{n}{k}$ is the number of ways to choose a k -element set from an n -element set.

This triangle is bounded by sides of length 1 and where every interior element is the sum of both parents:

$$\begin{array}{ccc} & \binom{n}{k} & \binom{n}{k+1} \\ & \dots\dots & \dots\dots \\ & \binom{n+1}{k+1} & \dots\dots \end{array}$$

2.6 Matrix Operations

We define on matrices over \mathbb{R} the following arithmetic operations.

Definition 20 (Matrices over the Reals) The domain of *real matrices* of dimension $m \times n$ is defined as follows:

$$\mathbb{M}_{m,n} := \mathbb{N}_m \times \mathbb{N}_n \rightarrow \mathbb{R}.$$

Definition 21 (Matrix Operations) For every $m \in \mathbb{N}$, $n \in \mathbb{N}$, and $p \in \mathbb{N}$ we define the following operations on real matrices.

Constants

null matrix (Nullmatrix):

$$\begin{aligned} 0 &: \mathbb{N}_m \times \mathbb{N}_n \rightarrow \mathbb{R}, \\ 0_{i,j} &:= 0. \end{aligned}$$

unity matrix (Einheitsmatrix):

$$\begin{aligned} 1 &: \mathbb{N}_n \times \mathbb{N}_n \rightarrow \mathbb{R}, \\ 1_{i,j} &:= \mathbf{if } i = j \mathbf{ then } 1 \mathbf{ else } 0. \end{aligned}$$

Addition

$$\begin{aligned} + &: \mathbb{M}_{m,n} \times \mathbb{M}_{m,n} \rightarrow \mathbb{M}_{m,n} \\ A + B &:= \mathbf{such } C \in \mathbb{M}_{m,n} : \\ &\quad \forall i \in \mathbb{N}_m, j \in \mathbb{N}_n : C_{i,j} = A_{i,j} + B_{i,j}. \end{aligned}$$

$$\text{short: } (A + B)_{i,j} := A_{i,j} + B_{i,j}.$$

Scalar Product

$$\begin{aligned} * : \mathbb{R} \times \mathbb{M}_{m,n} &\rightarrow \mathbb{M}_{m,n} \\ c * A &:= \mathbf{such} \ C \in \mathbb{M}_{m,n} : \\ &\forall i \in \mathbb{N}_m, j \in \mathbb{N}_n : C_{i,j} = c * A_{i,j}. \end{aligned}$$

$$\text{short: } (c * A)_{i,j} := c * A_{i,j}.$$

Matrix Product

$$\begin{aligned} * : \mathbb{M}_{m,n} \times \mathbb{M}_{n,p} &\rightarrow \mathbb{M}_{m,p} \\ A * B &:= \mathbf{such} \ C \in \mathbb{M}_{m,p} : \\ &\forall i \in \mathbb{N}_m, j \in \mathbb{N}_p : C_{i,j} = \sum_{0 \leq k < n} A_{i,k} * B_{k,j}. \end{aligned}$$

$$\text{short: } (A * B)_{i,j} := \sum_{0 \leq k < n} A_{i,k} * B_{k,j}.$$

Determinant

$$\begin{aligned} | \cdot | : \mathbb{M}_{n,n} &\rightarrow \mathbb{R}, \\ \text{if } n = 1 : & \\ |A| &:= A_{0,0}, \\ \text{if } n > 1 : & \\ |A| &:= \sum_{0 \leq j < n} A_{0,j} * (-1)^j * |B| \\ &\mathbf{where} \ B = \mathbf{such} \ B \in \mathbb{M}_{n-1,n-1} : \\ &\forall k \in \mathbb{N}_{n-1}, l \in \mathbb{N}_{n-1} : \\ &\quad B_{k,l} = (\mathbf{if} \ l < j \ \mathbf{then} \ A_{k+1,l} \ \mathbf{else} \ A_{k+1,l+1}). \end{aligned}$$

The determinant of an $n \times n$ matrix A is defined by the determinants of a number of $(n - 1) \times (n - 1)$ matrices B that are constructed from A by deleting the first row and some column j .

Example

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 * \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2 * \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3 * \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

$$\begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} = 5 * 9 - 6 * 8 = -3.$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} * \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} = \begin{bmatrix} 1a + 2c + 3e & 1b + 2d + 3f \\ 4a + 5c + 6e & 4b + 5d + 6f \end{bmatrix}$$

The matrix operations satisfy many equations that also hold for numbers, e.g., $A * (B + C) = A * B + A * C$. However, unlike in number domains, matrix multiplication is *not* commutative.

Example

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 9 \\ 17 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 6 & 10 \\ 14 & 20 \end{bmatrix}$$

Chapter 3

Induction

The domain of natural numbers has a property (implied by the first two Peano axioms) that allows us to *define functions and predicates inductively*, i.e., by sets of equations respectively equivalences with a particular structure. Likewise, the third Peano axiom give us *induction* as a technique for proving properties about natural numbers. In this chapter, we investigate these two aspects of induction, defining and proving, and demonstrate their relationship by proving properties of inductively/recursively defined functions. We also generalize the concept of inductive definitions and proofs from natural numbers to *inductively defined sets* which are prominent in computer science.

Induction means something like “generalization”; from a special pattern we conclude the general case.

3.1 Inductive Definitions

We have introduced in Section 2.1 multiplication over \mathbb{N} by the following recursive definition (with termination function $r(x, y) := y$)

$$\begin{aligned} * : \mathbb{N} \times \mathbb{N} &\rightarrow \mathbb{N} \\ x * y &:= \mathbf{if} \ y = 0 \ \mathbf{then} \ 0 \ \mathbf{else} \ x + (x * y^-) \end{aligned}$$

from which we can infer the following knowledge (Proposition 3 on page 21)

$$\begin{aligned} x * 0 &= 0, \\ x * y' &= x + (x * y). \end{aligned}$$

or in another form

$$\begin{aligned} x * 0 &= 0, \\ x * (y + 1) &= x + (x * y). \end{aligned}$$

Actually also the inverse is true: this pair of equations uniquely determines the function $*$: the first Peano law implies that the left hand side of only one equation “matches” a particular application $a * b$ (because b is either 0 or $y + 1$ for some y but not both) and the second Peano law implies that argument b determines the value of parameter y uniquely.

We can therefore write the definition also in the form

$$\begin{aligned} * &: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\ x * 0 &:= 0, \\ x * (y + 1) &:= x + (x * y). \end{aligned}$$

The syntactic properties of these equations guarantee that the corresponding function is well defined.

Definition 22 (Inductive Function Definition) An *inductive definition* (*induktive Definition*) over \mathbb{N} of an n -ary function f is a pair of equations

$$\begin{aligned} f(x_0, \dots, 0, \dots, x_{n-1}) &:= T_b, \\ f(x_0, \dots, x_i + 1, \dots, x_{n-1}) &:= T_r \end{aligned}$$

where T_b (whose free variables are among $x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_{n-1}$) is the *base term* in which f does not occur and where every application of f in the *recursion term* T_r (whose free variables are among x_0, \dots, x_{n-1}) has the form

$$f(T_0, \dots, x_i, \dots, T_{n-1})$$

for some terms T_0, \dots, T_{n-1} . We say that the induction *runs over* x_i .

If there exist $A_0, \dots, A_{i-1}, A_{i+1}, \dots, A_{n-1}, B$ with

$$\begin{aligned} \forall x_0 \in A_0, \dots, x_{i-1} \in A_{i-1}, x_i \in \mathbb{N}, x_{i+1} \in A_{i+1}, \dots, x_{n-1} \in A_{n-1} : \\ T_b \in B \wedge T_r \in B \end{aligned}$$

then above definition introduces a function

$$f : A_0 \times \dots \times A_{i-1} \times \mathbb{N} \times A_{i+1} \times \dots \times A_{n-1} \rightarrow B$$

such that the following holds:

$$\begin{aligned} \forall x_0 \in A_0, \dots, x_{i-1} \in A_{i-1}, x_i \in \mathbb{N}, x_{i+1} \in A_{i+1}, \dots, x_{n-1} \in A_{n-1} : \\ f(x_0, \dots, 0, \dots, x_{n-1}) = T_b \wedge \\ f(x_0, \dots, x_i + 1, \dots, x_{n-1}) = T_r. \end{aligned}$$

A function introduced by an inductive definition is uniquely defined.

One can generalize inductive definitions in various ways. For instance, it is not necessary that the induction term is decreased by exactly 1. We may use any non-zero decrement value as long as every possible case is covered by a corresponding equation. For instance, in the definition of the *Fibonacci Numbers* (*Fibonacci-Zahlen*)

$$\begin{aligned}\text{fib}(0) &:= 1, \\ \text{fib}(1) &:= 1, \\ \text{fib}(x + 2) &:= \text{fib}(x) + \text{fib}(x + 1)\end{aligned}$$

we have two base equations because in the last equation the induction term x is decreased by 2.

Furthermore, we may let the induction not just run over a single parameter but over a *combination* of parameters as in

$$\begin{aligned}f(0, 0) &:= 0, \\ f(x + 1, 0) &:= 1 + f(x, 0), \\ f(x, y + 1) &:= 1 + f(x, y).\end{aligned}$$

which defines a function that computes (in an artificially complex way) the sum of both arguments x and y . In this case, the induction is guarded by the *termination term* $x + y$ whose value is decreased by one in every recursive application of f .

If the induction runs over multiple arguments, we have to take special care that every possible argument pattern is covered by an equation. For instance, the definition

$$\begin{aligned}f(0, 0) &:= 0, \\ f(x + 1, 0) &:= 1 + f(x, 0), \\ f(0, y + 1) &:= 1 + f(0, y), \\ f(x + 1, y + 1) &:= 2 + f(x, y),\end{aligned}$$

uses four equations to cover all possible cases.

The more complicated the recursive structure of a function definition is, the more important it becomes to show that a set of equations indeed defines a total function. This can be done by giving a termination term or, more general, by constructing a well-founded ordering (see page 7).

Example *Ackerman's function* is defined as follows:

This is a function which grows *very* fast.

$$\begin{aligned} A(0, y) &:= y + 1, \\ A(x + 1, 0) &:= A(x, 1), \\ A(x + 1, y + 1) &:= A(x, A(x + 1, y)). \end{aligned}$$

We have $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ because we can construct the well-founded *lexicographic ordering* (*lexikographische Ordnung*) on the function arguments

$$x \prec y :\Leftrightarrow x_0 < y_0 \vee (x_0 = y_0 \wedge x_1 < y_1)$$

(e.g. $\langle 0, 0 \rangle \prec \langle 0, 1 \rangle$ and $\langle 0, 1 \rangle \prec \langle 1, 0 \rangle$) and show that the arguments decrease in every recursive application with respect to this ordering, i.e., for the three recursive applications in the equations above:

1. $\langle x, 1 \rangle \prec \langle x + 1, 0 \rangle$,
2. $\langle x + 1, y \rangle \prec \langle x + 1, y + 1 \rangle$,
3. $\langle x, A(x + 1, y) \rangle \prec \langle x + 1, y + 1 \rangle$.

All three statements are true by definition of \prec .

Also predicates can be defined inductively.

Definition 23 (Inductive Predicate Definition) An *inductive definition* (*Induktive Definition*) over \mathbb{N} of an n -ary predicate p is a pair of equivalences

$$\begin{aligned} p(x_0, \dots, 0, \dots, x_{n-1}) &:\Leftrightarrow F_b, \\ p(x_0, \dots, x_i + 1, \dots, x_{n-1}) &:\Leftrightarrow F_r \end{aligned}$$

where F_b (whose free variables are among $x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_{n-1}$) is the *base formula* in which p does not occur and where every application of p in the *recursion formula* F_r (whose free variables are among x_0, \dots, x_{n-1}) has the form

$$p(T_0, \dots, x_i, \dots, T_{n-1})$$

for some terms T_0, \dots, T_{n-1} . We say that the induction *runs over* x_i .

Above definition defines a predicate

$$p \subseteq A_0 \times \dots \times A_{i-1} \times \mathbb{N} \times A_{i+1} \times \dots \times A_{n-1}.$$

such that the following holds:

$$\begin{aligned} \forall x_0 \in A_0, \dots, x_{i-1} \in A_{i-1}, x_i \in \mathbb{N}, x_{i+1} \in A_{i+1}, \dots, x_{n-1} \in A_{n-1} : \\ (p(x_0, \dots, 0, \dots, x_{n-1}) \Leftrightarrow F_b) \wedge \\ (p(x_0, \dots, x_i + 1, \dots, x_{n-1}) \Leftrightarrow F_r). \end{aligned}$$

A predicate introduced by an inductive definition is uniquely defined.

Example We can introduce the predicate $\text{iseven}(x) :\Leftrightarrow 2|x$ also by the following recursive definition:

$$\begin{aligned} \text{iseven}(0) &:\Leftrightarrow \text{T}, \\ \text{iseven}(x + 1) &:\Leftrightarrow \neg \text{iseven}(x). \end{aligned}$$

We can define the same predicate also in the following way with two base cases and an induction term x that is decreased by 2:

$$\begin{aligned} \text{iseven}(0) &:\Leftrightarrow \text{T}, \\ \text{iseven}(1) &:\Leftrightarrow \text{F}, \\ \text{iseven}(x + 2) &:\Leftrightarrow \text{iseven}(x). \end{aligned}$$

3.2 Induction as a Proof Technique

The Peano Axioms (page 17) include for every formula F an axiom

$$(F[x \leftarrow 0] \wedge (\forall x \in \mathbb{N} : F \Rightarrow F[x \leftarrow x + 1])) \Rightarrow \forall x \in \mathbb{N} : F.$$

We may apply this axiom for *proving* properties of formulas involving natural numbers.

Proposition 11 (Mathematical Induction) In order to prove

$$\forall x \in \mathbb{N} : F,$$

it suffices to prove

1. $F[x \leftarrow 0]$,
2. $(\forall x \in \mathbb{N} : F \Rightarrow F[x \leftarrow x + 1])$.

In natural language, this proof strategy is usually indicated by the following triple of statements:

1. **Induction Base:** We show $F[x \leftarrow 0]$.
2. **Induction Hypothesis:** We take arbitrary $x \in \mathbb{N}$ and assume F .
3. **Induction Step:** We show $F[x \leftarrow x + 1]$.

Several examples are given below.

Example We prove

$$\forall n \in \mathbb{N} : n < 2^n$$

by induction on n .

The induction base holds because $0 < 1 = 2^0$.

Now we take arbitrary $n \in \mathbb{N}$ and assume

$$(1) \quad n < 2^n.$$

We have to show

$$(2) \quad n + 1 < 2^{n+1}.$$

By (1) we have

$$(3) \quad n + 1 < 2^n + 1$$

and therefore

$$(4) \quad n + 1 < 2^n + 1 \leq 2^n + 2^n = 2 * 2^n = 2^{n+1}$$

which implies (2).

(The derivation of (2) from (1) and $1 \leq 2^n$ are knowledge that we assume granted in this proof).

Example We prove

$$\forall n \in \mathbb{N} : 3|n^3 + 2n$$

by induction on n .

The induction base holds because $3|0$ and $0 = 0^3 + 2 * 0$.

We take arbitrary $n \in \mathbb{N}$ and assume

$$(1) 3|n^3 + 2n.$$

We have to show

$$(2) 3|(n + 1)^3 + 2(n + 1).$$

By (1) and definition of $|$ we have some $a \in \mathbb{N}$ such that

$$(3) 3a = n^3 + 2n.$$

We therefore have

$$\begin{aligned} (n + 1)^3 + 2(n + 1) &= \\ (n^3 + 3n^2 + 3n + 1) + (2n + 2) &= \\ (n^3 + 2n) + (3n^2 + 3n + 3) &= (3) \\ 3a + 3(n^2 + n + 1) &= \\ 3(a + n^2 + n + 1) & \end{aligned}$$

which implies (2) by definition of $|$.

(In this proof we assume the computing rules in \mathbb{N} as granted).

Example We prove

$$\forall n \in \mathbb{N} : \sum_{1 \leq i \leq n} i = \frac{(n + 1)n}{2}$$

by induction on n .

The induction base holds because

$$\sum_{1 \leq i \leq 0} i = 0 = \frac{(0+1) * 0}{2}.$$

We take arbitrary $n \in \mathbb{N}$ and assume

$$(1) \quad \sum_{1 \leq i \leq n} i = \frac{(n+1)n}{2}.$$

We have to show

$$(2) \quad \sum_{1 \leq i \leq n+1} i = \frac{((n+1)+1)(n+1)}{2}.$$

We have

$$\begin{aligned} \sum_{1 \leq i \leq n+1} i &= (\text{definition } \sum) \\ \sum_{1 \leq i \leq n} i + (n+1) &= (1) \\ \frac{(n+1)n}{2} + (n+1) &= \\ \frac{(n+1)n + 2(n+1)}{2} &= \\ \frac{(n+1)(n+2)}{2} &= \\ \frac{(n+1)((n+1)+1)}{2} &= \end{aligned}$$

which implies (2).

(In this proof we assume the computing rules in \mathbb{Q} as granted).

In particular, we can prove by induction the “computing laws” stated in Section 2.1 on page 22.

Example We prove

$$\forall x \in \mathbb{N}, y \in \mathbb{N}, z \in \mathbb{N} : x + (y + z) = (x + y) + z.$$

We take arbitrary $x \in \mathbb{N}$ and $y \in \mathbb{N}$ and prove

$$\forall z \in \mathbb{N} : x + (y + z) = (x + y) + z$$

by induction on z .

We have by definition of $+$

$$x + (y + 0) = x + y = (x + y) + 0$$

and thus the induction base.

We assume

$$(1) \quad x + (y + z) = (x + y) + z$$

and show

$$(2) \quad x + (y + (z + 1)) = (x + y) + (z + 1).$$

We have

$$\begin{aligned} (x + y) + (z + 1) &= \text{(definition } +) \\ ((x + y) + z) + 1 &= (1) \\ (x + (y + z)) + 1 &= \text{(definition } +) \\ x + ((y + z) + 1) &= \text{(definition } +) \\ x + (y + (z + 1)) & \end{aligned}$$

which implies (2).

By induction, we are also able to prove a proposition presented in the previous course notes.

Proposition 12 (Number of Permutations) The number of permutations of length n is $n!$.

Proof We proceed by induction on n .

If $n=0$, then the only permutation is $p = []$.

Assume $|\{f : f \text{ is permutation of length } n\}| = n!$.

We define

$$\begin{aligned} \text{insert}(x, i, f) &:= \\ \text{such } s &: \\ \text{length}(s) &= 1 + \text{length}(f) \wedge \\ \forall j \in \mathbb{N}_{n+1} &: \\ j < i &\Rightarrow s(j) = f(j) \wedge \\ j = i &\Rightarrow s(j) = x \wedge \\ j > i &\Rightarrow s(j) = f(j - 1) \end{aligned}$$

which returns the sequence constructed from f by inserting element x at position i .

Then we have

$$\begin{aligned} |\{f : f \text{ is permutation of length } n + 1\}| &= \\ |\{\text{insert}(n + 1, i, f) : i \in \mathbb{N}_{n+1} \wedge f \text{ is permutation of length } n\}| &= \\ (n + 1) * |\{f : f \text{ is permutation of length } n\}| &= \\ (n + 1) * n! &= \\ (n + 1)! & \end{aligned}$$

Similar to inductive definitions with termination terms that are decreased by values greater than one, proofs by induction may use a more general induction rule which allows to rely on the truth of the formula to be proved for all predecessors of the current number.

Proposition 14 (Complete Induction) In order to prove

$$\forall x \in \mathbb{N} : F$$

it suffices to prove

$$(\forall x \in \mathbb{N} : (\forall n < x : F[x \leftarrow n]) \Rightarrow F).$$

In complete induction, we use a generalized induction hypothesis in which we assume that F holds for all previous values of x (not just for $x - 1$). There is no need for a separate induction base, because for $x = 0$ the hypothesis $(\forall n < x : F[x \leftarrow n])$ collapses to \mathbb{T} .

The application of this proof strategy is usually indicated as follows:

1. **Induction Hypothesis.** We take arbitrary $x \in \mathbb{N}$ and assume

$$\forall n < x : F[x \leftarrow n].$$

2. **Induction Step:** We show F .

Complete induction can be applied not only in \mathbb{N} but in every domain A with a well-founded ordering $\prec \subseteq A \times A$ (see page 7).

Example We prove that every natural number greater than 1 can be factorized into a sequence of prime numbers, i.e.,

$$\begin{aligned} \forall n \in \mathbb{N} : n > 1 \Rightarrow \\ (\exists k \in \mathbb{N}, f : \mathbb{N}_k \rightarrow \mathbb{N} : n = \prod_{0 \leq i < k} f(i) \wedge \forall i \in \mathbb{N}_k : f(i) \text{ is prime}). \end{aligned}$$

We proceed by complete induction over n .

We take arbitrary $n \in \mathbb{N}$ and assume

$$\begin{aligned} (1) \forall m < n : m > 1 \Rightarrow \\ (\exists k \in \mathbb{N}, f : \mathbb{N}_k \rightarrow \mathbb{N} : m = \prod_{0 \leq i < k} f(i) \wedge \forall i \in \mathbb{N}_k : f(i) \text{ is prime}). \end{aligned}$$

We have to show

$$\begin{aligned} n > 1 \Rightarrow \\ (\exists k \in \mathbb{N}, f : \mathbb{N}_k \rightarrow \mathbb{N} : n = \prod_{0 \leq i < k} f(i) \wedge \forall i \in \mathbb{N}_k : f(i) \text{ is prime}). \end{aligned}$$

We assume (3) $n > 1$ and show

$$(4) (\exists k \in \mathbb{N}, f : \mathbb{N}_k \rightarrow \mathbb{N} : n = \prod_{0 \leq i < k} f(i) \wedge \forall i \in \mathbb{N}_k : f(i) \text{ is prime}).$$

If n is prime, then take $k := 1$ and $f_0 := n$ and we are done.

If n is not prime, then, by definition of primality, there exists some a with (5) $1 < a < n$ and (6) $a|n$. By (6) and definition of $|$, there exists some b such that

$$(7) a * b = n.$$

From (5) and (7), we have (8) $1 < b < n$.

By (1) and (5), we have some $k_a \in \mathbb{N}$ and $f_a : \mathbb{N}_{k_a} \rightarrow \mathbb{N}$ with

$$(9) \ a = \prod_{0 \leq i < k_a} f_a(i) \wedge \forall i \in \mathbb{N}_k : f_a(i) \text{ is prime.}$$

Likewise we have by (1) and (8) some $k_b \in \mathbb{N}$ and $f_b : \mathbb{N}_{k_b} \rightarrow \mathbb{N}$ with

$$(10) \ b = \prod_{0 \leq i < k_b} f_b(i) \wedge \forall i \in \mathbb{N}_k : f_b(i) \text{ is prime.}$$

Take $k := k_a + k_b$ and $f(i) := \mathbf{if} \ i < k_a \ \mathbf{then} \ f_a(i) \ \mathbf{else} \ f_b(i - k_a)$. By (7), (9), and (10), we have

$$(11) \ n = \prod_{0 \leq i < k} f(i) \wedge \forall i \in \mathbb{N}_k : f(i) \text{ is prime}$$

which implies (4).

3.3 Induction on Sets

The concept of induction is not limited to \mathbb{N} ; it can be applied to any set whose construction is based on a corresponding principle.

This is the kind of induction that you need when dealing with data structures in a computer.

Example Let “List(T)” be the set of all finite lists whose elements are from set T . Let “nil” denote the empty list and let “cons(e, l)” be the list with first element $e \in T$ and rest list $l \in \text{List}(T)$.

Then every element $l \in \text{List}(T)$ is either “nil” or “cons(e, l')” for some unique $e \in T$ and $l' \in \text{List}(T)$. Therefore we may inductively define functions and predicates on $\text{List}(T)$ and we may prove properties of the form $\forall x \in \text{List}(T) : F$ by induction.

In general, we may use inductive definitions as follows.

Definition 24 (Inductive Set Definition) An inductive definition of a set S is a collection of formulas

$$\begin{aligned} & (\forall x_1, \dots, x_{m_1}, y_1 \in S, \dots, y_{n_1} \in S : \\ & \quad f_1(x_1, \dots, x_{m_1}, y_1, \dots, y_{n_1}) \in S) \\ & , \dots , \\ & (\forall x_1, \dots, x_{m_c}, y_1 \in S, \dots, y_{n_c} \in S : \\ & \quad f_c(x_1, \dots, x_{m_c}, y_1, \dots, y_{n_c}) \in S) \end{aligned}$$

with object constant S and function constants f_1, \dots, f_c which denote the *constructors (Konstruktoren)* of S .

By this definition, S denotes the smallest set on which the conjunction of these formulas holds.

The notion “smallest” used in above definition implies that every element of the defined set is denoted by a constructor term

$$f_i(T_1, \dots, T_{m_i}, S_1, \dots, S_{n_i})$$

for some terms $T_1, \dots, T_{m_i}, S_1, \dots, S_{n_i}$ where the S_1, \dots, S_{n_i} are also such constructor terms.

Example

- The set \mathbb{N} is inductively defined by

$$\begin{aligned} 0 &\in \mathbb{N}, \\ \forall x \in \mathbb{N} : x' &\in \mathbb{N} \end{aligned}$$

for some constructors 0 and $'$. Every element of \mathbb{N} is of the form

$$0^{i' \dots'}$$

e.g. $0^{''''}$ is interpreted as the number 4 in \mathbb{N} .

- For every set T , the set “List(T)” is defined by

$$\begin{aligned} \text{nil} &\in \text{List}(T), \\ \forall e \in T, l \in \text{List}(T) : \text{cons}(e, l) &\in \text{List}(T). \end{aligned}$$

for some constructors “nil”, “cons”. Every element of List(T) has form

$$\text{cons}(e_0, \dots, \text{cons}(e_{n-1}, \text{nil})),$$

e.g. “cons(2, cons(3, nil))” is interpreted as the list [2, 3] in List(\mathbb{N}).

- For every set T , the set “Tree(T)” is defined by

$$\begin{aligned} \text{empty} &\in \text{Tree}(T), \\ \forall e \in T, l \in \text{Tree}(T), r \in \text{List}(T) : \text{node}(e, l, r) &\in \text{Tree}(T). \end{aligned}$$

with constructors “empty”, “node”. Every element of $\text{Tree}(T)$ has form

$$\text{node}(n_0, \text{node}(n_{11}, \dots), \text{node}(n_{21}, \dots)),$$

e.g. “node(1, node(2, node(3, empty, empty), node(4, empty, empty)), node(5, empty, empty))” is interpreted as the following tree in $\text{Tree}(\mathbb{N})$:

$$\begin{array}{ccc} & & 1 \\ & & / \quad \backslash \\ & 2 & & 5 \\ & / \quad \backslash & & \\ 3 & & 4 & \end{array}$$

- The set “Term” is defined by

$$\begin{aligned} 0 &\in \text{Term}, \\ 1 &\in \text{Term}, \\ \forall x \in \text{Term} : -x &\in \text{Term}, \\ \forall x \in \text{Term}, y \in \text{Term} : x + y &\in \text{Term}, \\ \forall x \in \text{Term}, y \in \text{Term} : x * y &\in \text{Term} \end{aligned}$$

with constructors $0, 1, -, +, *$. An element of “Term” is “ $1 + (1 + 0) * 1$ ”.

- The set “Formula” is defined by

$$\begin{aligned} T &\in \text{Formula} \\ \forall x \in \text{Formula} : \text{not}(x) &\in \text{Formula}, \\ \forall x \in \text{Formula}, y \in \text{Formula} : \text{and}(x, y) &\in \text{Formula}, \\ \forall x \in \text{Variable}, y \in \text{Formula} : \text{forall}(x, y) &\in \text{Formula} \end{aligned}$$

for some constructors “T”, “not”, “and”, “forall”. An element of “Formula” is “forall(X, and(T, and(T, T)))” (assuming $X \in \text{“Variable”}$).

Inductive set definitions may appear in various notations; e.g., the set “Term” in above example is more conveniently defined as

$$\text{Term} ::= 0 \mid 1 \mid -\text{Term} \mid \text{Term} + \text{Term} \mid \text{Term} * \text{Term}.$$

in the syntax of *BNF* (*Backus Naur Form*) that is used to describe *grammars* (*Grammatiken*) of formal languages.

We now constrain our notion of inductive sets such that the elements preserve “knowledge” about their construction.

Definition 25 (Term Algebra) An inductively defined set is a *term algebra* (*Term Algebra*) if we have for every constructor f of this set

$$\forall x, y : f(x) = f(y) \Rightarrow x = y$$

i.e., f is *injective* (*injektiv*). Furthermore, for all constructors f and g

$$\forall x, y : f(x) \neq g(y)$$

i.e., different constructors yield different results.

Every element of a term algebra is denoted by *exactly one* constructor term

$$f_i(T_1, \dots, T_{m_i}, S_1, \dots, S_{n_i})$$

for some terms $T_1, \dots, T_{m_i}, S_1, \dots, S_{n_i}$ where the S_1, \dots, S_{n_i} are also constructor terms. The name *term algebra* stems from this one to one correspondence between set elements and the terms that denote these elements.

Example Because of the first two Peano axioms (page 17), \mathbb{N} is a term algebra with constructors 0 and '.

A nice property of a term algebra is that they allow inductive function and predicate definitions in the style of inductive definitions over \mathbb{N} ; we do not introduce these definitions formally but give some examples.

Example

- Take the set $\text{List}(T)$ defined in the previous example and assume that it is a term algebra. We define the length of a list as

$$\begin{aligned} \text{length} &: \text{List}(T) \rightarrow \mathbb{N} \\ \text{length}(\text{nil}) &:= 0 \\ \text{length}(\text{cons}(e, l)) &:= 1 + \text{length}(l). \end{aligned}$$

Then we have $\text{length}(\text{cons}(1, \text{cons}(2, \text{nil}))) = 2$.

- Take the set $\text{List}(T)$ defined in the previous example and assume that it is a term algebra. We define the depth of a tree as

$$\begin{aligned} \text{depth} &: \text{Tree}(T) \rightarrow \mathbb{N} \\ \text{depth}(\text{empty}) &:= 0 \\ \text{depth}(\text{node}(n, l, r)) &:= 1 + \max(\text{depth}(l), \text{depth}(r)). \end{aligned}$$

Then we have $\text{depth}(\text{node}(1, \text{empty}, \text{node}(2, \text{node}(3, \text{empty}, \text{empty}), \text{empty}))) = 3$.

- Take the set Term defined in the previous example and assume that it is a term algebra. We define the value of a term as

$$\begin{aligned} \text{value} &: \text{Term} \rightarrow \mathbb{Z} \\ \text{value}(0) &:= 0_{\mathbb{Z}} \\ \text{value}(1) &:= 1_{\mathbb{Z}} \\ \text{value}(-x) &:= -_{\mathbb{Z}}\text{value}(x) \\ \text{value}(x + y) &:= \text{value}(x) +_{\mathbb{Z}} \text{value}(y) \\ \text{value}(x * y) &:= \text{value}(x) *_{\mathbb{Z}} \text{value}(y) \end{aligned}$$

Then we have $\text{value}(1 + (1 + 0) * 1) = 2$.

On inductively defined sets (not necessarily term algebras) we have the following induction principle.

Proposition 15 (Structural Induction) In order to prove a property

$$\forall x \in S : F$$

for an inductively defined set S , it suffices to prove

$$\begin{aligned} \forall x_1, \dots, x_{m_i}, y_1 \in S, \dots, y_{n_i} \in S : \\ (F[x := y_1] \wedge \dots \wedge F[x := y_{n_i}]) \Rightarrow \\ F[x := f_i(x_1, \dots, x_{m_i}, y_1, \dots, y_{n_i})] \end{aligned}$$

for every constructor f_i of S .

Intuitively, we let the induction run over the “structure” of every term

$$f_i(x_1, \dots, x_{m_i}, y_1, \dots, y_{n_i}).$$

denoting some element x in S . We assume that F holds for every “ S -component” y_j of x and show that F is propagated to x itself.

Example Take the set “List(T)” defined inductively as

$$\begin{aligned} \text{nil} &\in \text{List}(T), \\ \forall e \in T, l \in \text{List}(T) : \text{cons}(e, l) &\in \text{List}(T). \end{aligned}$$

We define

$$\begin{aligned} \text{append} &: \text{List}(T) \times \text{List}(T) \rightarrow \text{List}(T) \\ \text{append}(\text{nil}, y) &:= y \\ \text{append}(\text{cons}(e, x), y) &:= \text{cons}(e, \text{append}(x, y)) \end{aligned}$$

and claim that (for the function “length” defined in the previous example) the following holds:

$$\begin{aligned} \forall x \in \text{List}(T), y \in \text{List}(T) : \\ \text{length}(\text{append}(x, y)) &= \text{length}(x) + \text{length}(y). \end{aligned}$$

We proceed by structural induction on x :

Case $x = \text{nil}$: We have to show

$$\begin{aligned} \forall y \in \text{List}(T) : \\ \text{length}(\text{append}(\text{nil}, y)) &= \text{length}(\text{nil}) + \text{length}(y). \end{aligned}$$

Take arbitrary $y \in \text{List}(T)$. We have

$$\begin{aligned} \text{length}(\text{append}(\text{nil}, y)) &= (\text{definition append}) \\ \text{length}(y) &= \\ 0 + \text{length}(y) &= (\text{definition length}) \\ \text{length}(\text{nil}) + \text{length}(y). & \end{aligned}$$

Case $x = \text{cons}(e, l)$: Take arbitrary $e \in T$ and $l \in \text{List}(T)$.

We assume (induction hypothesis)

$$\begin{aligned} \forall y \in \text{List}(T) : \\ \text{length}(\text{append}(l, y)) &= \text{length}(l) + \text{length}(y) \end{aligned}$$

and have to show

$$\begin{aligned} \forall y \in \text{List}(T) : \\ \text{length}(\text{append}(\text{cons}(e, l), y)) &= \text{length}(\text{cons}(e, l)) + \text{length}(y). \end{aligned}$$

Take arbitrary $y \in \text{List}(T)$. We have

$$\begin{aligned} \text{length}(\text{append}(\text{cons}(e, l), y)) &= \text{(definition append)} \\ \text{length}(\text{cons}(e, \text{append}(l, y))) &= \text{(definition length)} \\ 1 + \text{length}(\text{append}(l, y)) &= \text{(induction hypothesis)} \\ 1 + (\text{length}(l) + \text{length}(y)) &= \\ (1 + \text{length}(l)) + \text{length}(y) &= \text{(definition length)} \\ \text{length}(\text{cons}(e, l)) + \text{length}(y). & \end{aligned}$$

Several formal concepts in computer science, e.g. datatypes and formal languages, can be reduced to inductively defined sets; the principle of structural induction is therefore of great importance.

Part II

Predicate Logic Proofs

Chapter 4

Introduction

“Contrariwise,” continued Tweedledee, “if it was so, it might be, and if it were so, it would be; but as it isn’t, it ain’t. That’s logic!”

– Lewis Carroll, “Through the Looking Glass”

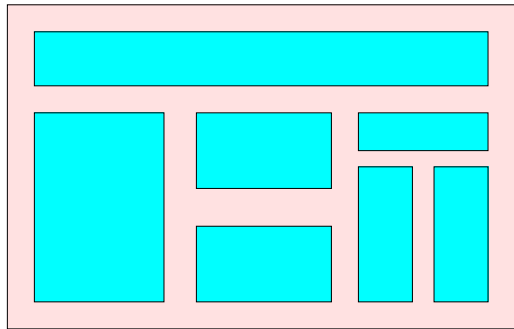
A *proposition* or *theorem* (*Satz*) is a formula that is claimed to be true in a given theory. However, a claim is only as good as the substantiating argument is. Since different people may disagree on the quality of an argument, we need some objective criteria whether an argument is correct or not. Formal logic solves this problem by providing rules for the *structure* (i.e., form, not content) of an argument: any argument whose form corresponds to these rules is correct. We call such a well-structured argument a *proof* (*Beweis*).

It is therefore in principle always possible to decide whether a given argument represents a proof or not; once a proof is given, there is no more dispute about the validity of a proposition. Proving is thus at the heart of any critical discourse and the knowledge about the correct application of proof rules is a must for a scientist or engineer: if we derive new knowledge, we must be able to prove it; if we are given some knowledge, we must be able to check it (i.e., its proof).

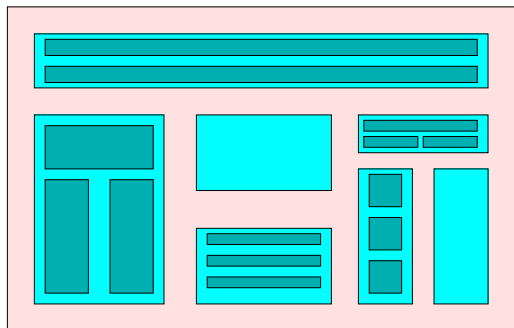
To *invent* a proof is a creative activity. The proof rules give some guiding principles that help in the first steps; however, from a certain point on, some insight is required to discover the “killer argument” that “slays” the proposition. To *check* a proof is simply craft; everybody who understands “proof terminology” is able to read a proof and judge its correctness.

A proof can be given on different levels of detail; on the lowest level, the individual reasoning steps are so small that their correct application can be even automatically verified by a computer. However, such proofs become very large and are rarely written by humans.

On a higher level, the reasoning steps are larger and less self-evident; still the *structure* of a (less detailed) high-level proof matches the structure of



A proof in less detail



The same proof with some more details

Figure 4.1: The Structure of a Proof

the (more detailed) low-level proof as it is determined by the proof rules (see Figure 4.1). If some opponent questions our high-level proof, we can correspondingly *refine* it by decomposing a large reasoning step into some smaller steps. This process can be repeated until we reach the lowest level (which is automatically checkable) or our opponent is satisfied. Even if we are not writing all details in a high-level proof, the skeleton provided by the proof rules helps us to maintain its power of persuasion and justify our confidence in its correctness. This is the kind of proofs that humans typically write in scientific works.

To choose the right size of reasoning steps is a tradeoff between demonstrating the “key ideas” of a proof (by skipping details that the reader is supposed to be able to fill in herself) and giving convincing arguments (by providing sufficient details that enable the reader to understand the line of reasoning). However, while we may easily bore our audience with too many details, we may even more easily lose its understanding by giving too few. Also for the

Confidence is the feeling you have before you understand the situation.

If you cannot convince them, confuse them.
– Harry S Truman

sake of correctness, it is better to elaborate more details than we may be originally inclined: many textbook proofs are wrong¹, because the author tried to save some work: she used reasoning steps that were so large that she could ultimately not grasp them herself, or she took wrong shortcuts in the application of proof rules.

If a proof is not understood, the proof uses too large reasoning steps, has bad style, or is simply wrong (the boundaries are fluid). This is in general not the fault of the audience; the presenter has the duty to adapt her presentation to the available knowledge, to make a clear argument and to refine it down to a satisfying level of detail. Science is not theology: no one is obliged to believe our claim unless we give a compelling argument.

Lemma: All horses have the same color.

Proof (by induction):

Base case *1 horse. Clearly with just 1 horse, all horses have the same color.*

Inductive Step *We'll show that if it is true for any group of N horses, that all have the same color, then it is true for any group of $N + 1$ horses.*

Well, given any set of $N + 1$ horses, if you exclude the last horse, you get a set of N horses. By the inductive step these N horses all have the same color. But by excluding the first horse in the pack of $N + 1$ horses, you can conclude that the last N horses also have the same color. Therefore all $N + 1$ horses have the same color.

Theorem: *All horses have an infinite number of legs.*

Proof (by intimidation):

Everyone would agree that all horses have an even number of legs. It is also well-known that horses have forelegs in front and two legs in back. $4 + 2 = 6$ legs, which is certainly an odd number of legs for a horse to have! Now the only number that is both even and odd is infinity; therefore all horses have an infinite number of legs. However, suppose that there is a horse somewhere that does not have an infinite number of legs. Well, that would be a horse of a different color; and by the Lemma, it doesn't exist.

¹Usually the corresponding propositions are nevertheless true.

Chapter 5

Preliminaries

The proof of a proposition is always relative to given knowledge. Initially, this is the knowledge

The best defense against logic is ignorance.

- that characterizes the considered theory (*axioms*), or
- that is derived from a “minor” extension of the theory (*definitions*), or
- that is true in any domain (*tautologies*), or
- that has been derived by another proof (*propositions*).

In the course of a proof, new knowledge is gradually added (*assumptions*).

5.1 Proof Situations

A *proof situation* consists of the available knowledge K (a set of formulas) and the goal G to be proved (a formula). We represent such a situation graphically as

$$\frac{K}{G}$$

and read this as “we (have to) prove G with knowledge K ”. In a natural language proof, the knowledge that is available in a particular proof situation has to be deduced from the context: it consists of the knowledge at the beginning of the proof extended by all the temporary definitions and assumptions that were made in the branch of the proof that led to the current situation.

Example We want to prove $\forall A : A \subseteq A$.

In this case,

- our goal G is the formula

$$\forall A : A \subseteq A;$$

- our knowledge K are all the axioms of set theory as well as the theorem

$$\forall A, B : A \subseteq B \Leftrightarrow \forall x \in A : x \in B$$

which is a direct consequence of the definition of the predicate \subseteq .

5.2 Proof Rules

A *proof rule* reduces a proof situation to one or more other situations. We denote this reduction by \rightsquigarrow and read e.g.

$$\begin{array}{|c|} \hline K_0 \\ \hline G_0 \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|} \hline K_1 & K_2 \\ \hline G_1 & G_2 \\ \hline \end{array}$$

as “in order to prove G_0 with knowledge K_0 it suffices to prove G_1 with knowledge K_1 and G_2 with knowledge K_2 ”.

A proof is the reduction of the start situation to other situations that are again reduced to other situations until we have only situations in which nothing is left to be proved. Graphically, a proof can therefore be depicted as a *tree* with the initial proof situation as the root and only proof situations with empty goals as the leaves (see Figure 5.1).

Example We want to prove

$$\exists x \in \mathbb{N} : \forall y \in \mathbb{N} : x \leq y \wedge \neg \exists x \in \mathbb{N} : \forall y \in \mathbb{N} : y \leq x.$$

Our goal G_0 is above formula; our knowledge K_0 consists of the axioms of theorems of sets and natural numbers including the theorem immediately derived from the definition of \leq .

We prove this by proving two theorems:

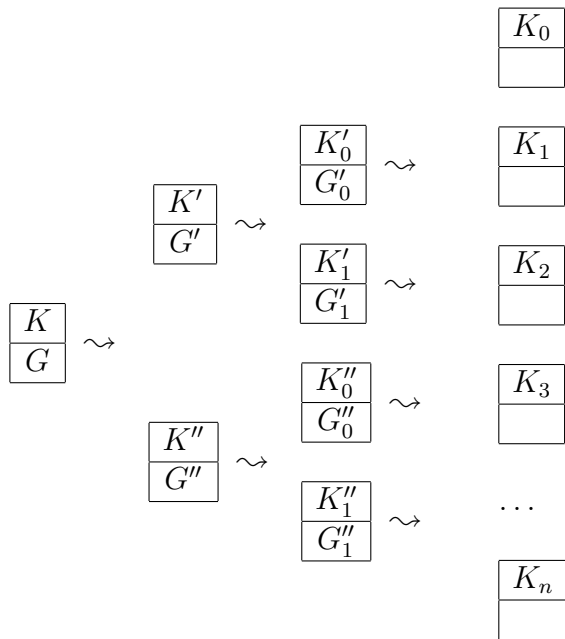


Figure 5.1: A Proof Tree

1. $\exists x \in \mathbb{N} : \forall y \in \mathbb{N} : x \leq y.$

In this subproof, our goal G_1 is above formula and our knowledge K_1 is K_0 .

2. $\neg \exists x \in \mathbb{N} : \forall y \in \mathbb{N} : y \leq x.$

In this subproof, our goal G_2 is above formula and our knowledge K_1 is also K_0 .

We have therefore reduced a proof situation to two different proof situations.

5.3 Proof Termination

The only way to reach a leaf in the proof tree (i.e., to complete a particular subproof) is by the following rule.

Proposition 16 (Proof Completion) For proving with knowledge $K \cup \{G\}$ the goal G , nothing has to be done any more.

$$\frac{K \cup \{G\}}{G} \rightsquigarrow \frac{K \cup \{G\}}{}$$

In other words, if the goal is in our “knowledge base”, we are done.

Example Assume our knowledge base contains the formula $c = 1$ and we want to prove $c = 1$. Then our proof is done.

Theorem: a cat has nine tails.

Proof: No cat has eight tails. A cat has one tail more than no cat. Thus a cat has nine tails. q.e.d.

The ultimate end of a proof is often denoted by the marker \square or by the acronym “q.e.d.” (*quod erat demonstrandum*, i.e., Latin for “what had to be shown”).

Chapter 6

General Strategies

While the construction of a proof is a creative process, there are some general guidelines. In this chapter, we introduce two classes of proofs:

- “direct proofs” which prove the goal formula, and
- “indirect proofs” which assume that the goal does not hold and from this derive a contradiction.

How many mathematical logicians does it take to replace a lightbulb? None: They can’t do it, but they can prove that it can be done.

Furthermore, we classify the proving rules into those that decompose the goal and thus make the proof simpler (“top-down” steps) and those that derive new knowledge and thus extend our knowledge base (“bottom-up” steps).

6.1 Direct Proofs

A *direct proof* proves a goal G given some knowledge K . However, in reality the usual situation is that we want to find out whether a formula G is true or not, i.e., we *don’t know* the truth of G in advance. Therefore we have actually two possibilities:

1. We try $\boxed{\frac{K}{G}}$. If we are successful, then G holds.

2. We try $\boxed{\frac{K}{\neg G}}$. If we are successful, then G does not hold.

If the first proof does not work out, it may give sufficient insight in order to establish the second proof. If we can show $\neg G$, we may have gained additional insight that enables us to transform G into some G' whose validity we can actually prove.

Example We are interested in the formula

$$(*) \forall A : \exists x : x \in A.$$

We have two possibilities.

1. We try to prove $\forall A : \exists x : x \in A$. If we succeed, $(*)$ holds.
2. We try to prove $\neg \forall A : \exists x : x \in A$, i.e.,

$$\exists A : \forall x : x \notin A.$$

If we succeed, $(*)$ does not hold.

If we do not succeed in both cases, we do not know whether $(*)$ holds.

6.2 Indirect Proofs

If we are not successful with direct attempts, we may try an *indirect proof* where we assume that the goal does not hold and derive a *contradiction*.

This sentence contradicts itself — no actually it doesn't. — Hofstadter

Proposition 17 (Proof by Contradiction) For proving with knowledge K the goal G , it suffices to prove $F(\text{alse})$ with additional knowledge $\neg G$.

$$\frac{K}{G} \rightsquigarrow \frac{K \cup \{\neg G\}}{F(\text{alse})}$$

If we can prove the formula “ $F(\text{alse})$ ” (which does certainly not hold), our knowledge base is inconsistent. Assuming that the original knowledge base K is consistent, the inconsistency can only come from adding $\neg G$, therefore G must hold.

In natural language, a proof of G by contradiction is usually indicated as

We assume $\neg G$ and show a contradiction.

A contradiction is usually derived by establishing a proof situation

$$\frac{K \cup \{F, \neg F\}}{F(\text{alse})}$$

where we have at the same time a formula and its negation in the knowledge base. We can then immediately terminate the proof, because we know that we can derive

$$\frac{K \cup \{F, \neg F\}}{F(\text{alse})} \rightsquigarrow \frac{K \cup \{F(\text{alse})\}}{F(\text{alse})} \rightsquigarrow \frac{K \cup \{F(\text{alse})\}}{}.$$

Theorem There is no square root of 2 in \mathbb{Q} , i.e.,

$$\neg \exists x \in \mathbb{Q} : x^2 = 2.$$

Proof: We assume

$$\exists x \in \mathbb{Q} : x^2 = 2.$$

and show a contradiction.

Because of the assumption, we have some $x \in \mathbb{Q}$ with

$$(1) x^2 = 2$$

We know

$$\mathbb{Q} := \{y : \exists a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\} : y = a/b \wedge a \text{ and } b \text{ are relatively prime}\}.$$

Since $x \in \mathbb{Q}$, we thus know

$$\exists a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\} : x = a/b \wedge a \text{ and } b \text{ are relatively prime.}$$

Thus we have some $a \in \mathbb{Z}$ and $b \in \mathbb{Z} \setminus \{0\}$ such that $x = \frac{a}{b}$ and a and b are relatively prime, i.e., by definition,

$$(2) \neg \exists x \in \mathbb{Z} \setminus \{-1, 1\} : x|a \wedge x|b.$$

We typeset those parts of a proof that refer to the previously introduced rule in **boldface**.

We have $x^2 = \left(\frac{a}{b}\right)^2 = \frac{a^2}{b^2} = 2$ and thus

$$(3) \quad a^2 = 2b^2$$

From (3) and the definition of $|$, we know $2|a^2$ and thus also

$$(4) \quad 2|a.$$

(because, if a is odd, also a^2 is odd).

Then, by the definition of $|$, there exists some $c \in \mathbb{Z}$ such that

$$(5) \quad a = 2c.$$

From (3) and (5) we have $(2c)^2 = 2b^2$, i.e., $4c^2 = 2b^2$, thus $2c^2 = b^2$.

Thus, by definition of $|$, we have $2|b^2$ and therefore

$$(6) \quad 2|b$$

(because, if b is odd, also b^2 is odd).

(4) and (6) contradict (2).

Theorem The second law of Peano states

$$\forall x, y : x' = y' \Rightarrow x = y.$$

Proof: Take arbitrary x and y . We have to prove

$$x' = y' \Rightarrow x = y.$$

We assume

$$(1) \quad x' = y'$$

and show $x = y$.

We assume

$$(2) \quad x \neq y$$

and show a contradiction.

From (1), we know by the definition of ' \in '

$$(3) x \cup \{x\} = y \cup \{y\}.$$

We know by the definition of \cup and set enumeration

$$(4) x \in x \cup \{x\}$$

and

$$(5) y \in y \cup \{y\}.$$

From (3), (4), and the definition of $=$, we have

$$(6) x \in y \cup \{y\}$$

which implies with (2)

$$(7) x \in y.$$

Likewise, we have from (3), (5), and the definition of $=$ that

$$(8) y \in x \cup \{x\}$$

which implies with (2)

$$(9) y \in x.$$

The conjunction of (7) and (9), i.e., $x \in y \wedge y \in x$, is a **contradiction** to the set-theoretic axiom of regularity that prohibits such "cycles".

Proposition 18 (Size of Powerset) Every set is smaller than its powerset:

$$\forall S : S \text{ is smaller than } \mathbb{P}(S).$$

Proof Take arbitrary S . We have to prove that S is smaller than $\mathbb{P}(S)$, i.e., by definition,

1. S is not larger than $\mathbb{P}(S)$.

By definition, we have to find some $f : S \xrightarrow{\text{injective}} \mathbb{P}(S)$.

Take $f(x) := \{x\}$.

2. S and $\mathbb{P}(S)$ are not of the same size.

Assume that S and $\mathbb{P}(S)$ are of the same size, i.e., there exists some $f : S \xrightarrow{\text{bijective}} \mathbb{P}(S)$. We show a contradiction.

Take $A := \{x \in S : x \notin f(x)\}$. Since f is surjective and $A \subseteq S$, i.e., $A \in \mathbb{P}(S)$, we have some $a \in S$ with $f(a) = A$. But then we know

$$a \in A \Leftrightarrow a \notin f(a) \Leftrightarrow a \notin A.$$

Also in indirect proofs, since we don't know the falseness of G for sure, we have two possibilities:

1. We try $\frac{K \cup \{\neg G\}}{F(\text{false})}$. If we are successful, then G holds.
2. We try $\frac{K \cup \{G\}}{F(\text{false})}$. If we are successful, then $\neg G$ holds.

If all attempts have failed, we have hopefully gained more insight to start a new round of attempts with refined ideas.

6.3 Proof Directions

When pursuing a proof, we may proceed in two ways (see Figure 6.1):

1. **Top-Down** by decomposing the goal into simpler formulas.

$$\frac{K}{G} \rightsquigarrow \frac{K_0}{G_0} \cdots \frac{K_{n-1}}{G_{n-1}}$$

2. **Bottom-Up** by deriving new knowledge from the given knowledge.

$$\frac{K}{G} \rightsquigarrow \frac{K \cup \{F\}}{G}$$

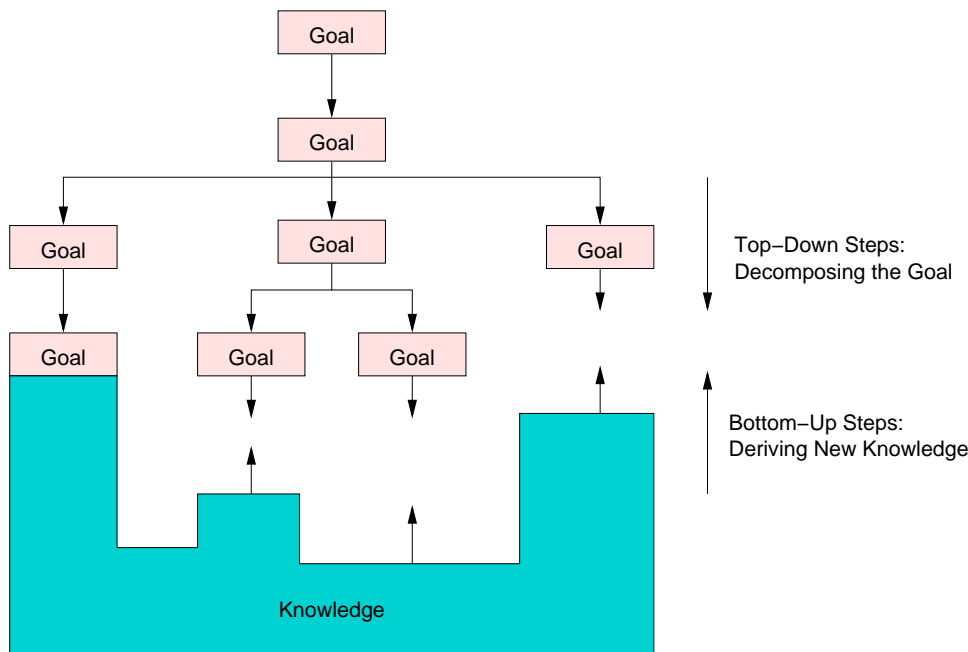


Figure 6.1: The Proof Directions

We proceed to decompose the goal into simpler goals and add knowledge to our knowledge base such that ultimately we have the situation explained in Section 5.3: the goal to be proved is part of the knowledge.

In the course of a proof we mix top-down steps with bottom-up steps; usually we begin with the top-down strategy.

Theorem We repeat the proof of the second law of Peano from page 68

$$\forall x, y : x' = y' \Rightarrow x = y.$$

and state explicitly where we apply top-down respectively bottom up steps.

Proof: (*We start in a proof situation where our goal G is above formula and the knowledge base K are the laws of set theory and the definition of $'$.*)

Take arbitrary x and y . We have to prove

$$x' = y' \Rightarrow x = y.$$

(*Here we have applied a top-down step where we have removed the outermost quantifier from G yielding a new goal G_0 . The knowledge K_0 for proving this new goal equals the original knowledge K .*)

We put additional explanations that are not part of the proof in parentheses and typeset them in *italics*.

We assume

$$(1) x' = y'$$

and show $x = y$.

(Here we have applied another top-down step. The new proof situation has knowledge $K_1 := K_0 \cup \{x' = y'\}$ and $x = y$ as the new goal G_1 .)

We assume

$$(2) x \neq y$$

and show a contradiction.

(We start an indirect proof.)

From (1), we know by the definition of $'$

$$(3) x \cup \{x\} = y \cup \{y\}.$$

(This was a bottom-up step where we have added to our knowledge base K_1 a new formula (3) which is a consequence of two other formulas in the knowledge base, namely (1) and the definition of the successor function $'$.)

Then we know by the definition of \cup and set enumeration

$$(4) x \in x \cup \{x\}$$

and

$$(5) y \in y \cup \{y\}.$$

(Again a bottom-up step which adds two formulas to the knowledge base.)

From (3), (4), and the definition of $=$, we have

$$(6) x \in y \cup \{y\}$$

which implies with (2)

$$(7) x \in y.$$

(Two bottom-up steps which add two more formulas.)

Likewise, we have from (3), (5), and the definition of $=$ that

$$(8) y \in x \cup \{x\}$$

which implies with (2)

$$(9) y \in x.$$

(Two more bottom-up steps adding two formulas.)

The conjunction of (7) and (9), i.e., $x \in y \wedge y \in x$,

(This was the last bottom up step adding the conjunction to the knowledge base.)

is in contradiction to the set-theoretic axiom of regularity (Regularität) that prohibits such “cycles”.

(The last formula added to the knowledge base is in contradiction to another formula in the knowledge base.)

In the following sections, we will explain which top-down and bottom-up steps are legal in a proof.

Chapter 7

Decomposing the Goal

In a proof situation $\frac{K}{G}$ where the goal G is not an atomic formula, the usual strategy is to decompose the proof into subproofs in a way that is determined by the outermost quantifier respectively connective of G .

7.1 Quantifiers

In this section, we show decomposition rules that can be applied, if the outermost symbol of a formula is a (universal or existential) quantifier.

As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality.

– Albert Einstein

7.1.1 Universal Quantifiers

Proposition 20 (Decomposition of Universal Formulas) For proving with knowledge K the goal $\forall x : G$, it suffices to prove $G[x \leftarrow a]$ where a is an object constant that does not appear in K and not in G .

$$\frac{K}{\forall x : G} \rightsquigarrow \frac{K}{G[x \leftarrow a]} \quad (a \text{ not in } K \cup \{G\})$$

In the proof of a universal formula we thus replace the variable by a constant about which no assumptions have been made yet. This may be indicated in a natural language proof in a very verbose form as

We have to prove $(\forall x : G)$. We take an arbitrary (but fixed) constant a and show $G[x \leftarrow a]$.

Theorem We prove the second law of Peano

$$\forall x, y : x' = y' \Rightarrow x = y.$$

We take arbitrary constants a and b and prove

$$a' = b' \Rightarrow a = b.$$

...

(The remainder of the proof was shown on page 68.)

Usually, we choose instead of a some name that indicates the name of the corresponding variable, e.g., \bar{x} . Even more frequently, we simply chose x as the name of the constant provided that x *does not appear freely* in K or G . We then simply write

We have to prove $(\forall x : G)$. We take arbitrary x and show G .

or even shorter

We have to prove $(\forall x : G)$. Take arbitrary x . Then ... (proof of G).

However, we have to understand that x now represents an arbitrary (but fixed) constant. Almost all proofs in this document use this style.

Theorem We prove the second law of Peano

$$\forall x, y : x' = y' \Rightarrow x = y.$$

Take arbitrary x and y .

(Now the proof of $x' = y' \Rightarrow x = y$ follows).

We assume

$$(1) x' = y'$$

and show $x = y$

(The remainder of the proof was shown on page 68.)

Theorem We prove the first law of Peano

$$\forall x : x' \neq 0.$$

Take arbitrary x .

(Now the proof of $x' \neq 0$ follows.)

By definition of 0 and $'$, we have to prove

$$x \cup \{x\} \neq \emptyset$$

which is true because $x \in (x \cup \{x\})$ but $x \notin \emptyset$.

Frequently also the *indirect method* is applied in such a situation. Instead of proving $(\forall x : G)$ we assume $(\neg \forall x : G)$, i.e., $(\exists x : \neg G)$ and proceed to derive a contradiction with the help of a constant a such that G holds (see Proposition 30 on page 100):

$$\frac{K}{\forall x : G} \rightsquigarrow \frac{K \cup \{\exists x : \neg G\}}{F(\text{alse})} \rightsquigarrow \frac{K \cup \{\neg G[x \leftarrow a]\}}{F(\text{alse})}$$

This is typically indicated as

We have to prove $(\forall x : G)$. Assume $\neg G$ for some x . Then ...
(derivation of a contradiction with additional knowledge $\neg G$).

In this formulation, x now represents a constant for the rest of the proof.

Theorem We prove that there exist infinitely many prime numbers, i.e., that there is no largest prime number:

$$\forall x \in \mathbb{N} : \exists p \in \mathbb{N} : p > x \wedge p \text{ is prime.}$$

Assume that there is some $x \in \mathbb{N}$ such that

$$(1) \forall p \in \mathbb{N} : p \text{ is prime} \Rightarrow p \leq x.$$

(The negation of the goal is $(\exists x \in \mathbb{N} : \forall p \in \mathbb{N} : p \leq x \vee p \text{ is not prime})$, i.e., $\exists x \in \mathbb{N} : \forall p \in \mathbb{N} : p \text{ is prime} \Rightarrow p \leq x$).

Define $a := 1 + \prod_{1 \leq i \leq x} i$. Now we have two possibilities:

1. a is prime. But from the definition of a we know $a > x$, thus we have a contradiction to (1).
2. a is not prime. Thus we can define

$$i := \min_{i \in \mathbb{N}} 1 < i < a \wedge i|a.$$

(since a is not prime, we know $\exists i \in \mathbb{N} : 1 < i < a \wedge i|a$).

From the definition of a , we know

$$\forall i \in \mathbb{N} : 1 \leq i \leq x \Rightarrow i \nmid a.$$

therefore

$$(2) i > x.$$

It remains to be shown that

$$(3) i \text{ is prime}$$

which together with (2) contradicts (1).

(We are now going to prove (3) in a small subproof.)

Assume i is not prime.

(We start an indirect proof.)

Then we have some $j \in \mathbb{N}$ with

$$(4) \quad 1 < j < i$$

and

$$(5) \quad j|i.$$

From (4) and the definition of i , we have

$$(6) \quad 1 < j < a.$$

From (5) and the definition of i , we have

$$(7) \quad j|a$$

(This deserves a little extra proof.)

(4), (6) and (7) contradict the definition of i .

7.1.2 Existential Quantifiers

Proposition 21 (Decomposition of Existential Formulas) For proving with knowledge K the goal $\exists x : G$, it suffices to prove $G[x \leftarrow T]$ for some term T .

$$\frac{K}{\exists x : G} \rightsquigarrow \frac{K}{G[x \leftarrow T]}$$

An existential proof requires creativity. We have to find a *witness* (*Zeugen*), i.e., a value for x that makes G true. This value can be denoted by a term T constructed from those function and object constants about which we have some knowledge in K such that we can proceed to prove G .

In natural language, such a proof is usually indicated as

We have to prove $(\exists x : G)$. We prove $G[x \leftarrow T]$.

or, if x occurs multiple times in G , shorter as

We have to prove $(\exists x : G)$. Take $a := T$. We prove $G[x \leftarrow a]$.

where a is an object constant that does not appear in K and for which we assume the additional knowledge $a = T$. Even shorter, we may write

We have to prove $(\exists x : G)$. Take $x := T$. We then have ... (proof of G with additional knowledge $x = T$).

This is possible if x *does not occur freely* in K .

Theorem Between any two rational numbers, there is another rational number:

$$\forall x \in \mathbb{Q}, y \in \mathbb{Q} : x < y \Rightarrow \exists z \in \mathbb{Q} : x < z < y.$$

Proof: Take arbitrary $x \in \mathbb{Q}$ and $y \in \mathbb{Q}$ with $x < y$. We have to show

$$\exists z \in \mathbb{Q} : x < z < y.$$

Take $z := (x + y)/2$.

Clearly $z \in \mathbb{Q}$. We have to show

$$x < z < y.$$

Because $x < y$, we know

$$z = (x + y)/2 < (y + y)/2 = y$$

and likewise

$$z = (x + y)/2 > (x + x)/2 = x$$

and are therefore done.

Sometimes, the witness cannot be constructed explicitly but its existence can be deduced in a bottom up step from knowledge about the existence of other objects (see Section 8.3).

Theorem The composition of two bijective functions is also a bijective function:

$$\forall A, B, C, f : A \xrightarrow{\text{bijective}} B, g : B \xrightarrow{\text{bijective}} C : \\ f \circ g : A \xrightarrow{\text{bijective}} C.$$

Proof: Take arbitrary $A, B, C, f : A \xrightarrow{\text{bijective}} B, g : B \xrightarrow{\text{bijective}} C$.

(We show $f \circ g : A \rightarrow C$ and f is bijective from A to B .)

1. We show $f \circ g : A \rightarrow C$.

This follows from $f : A \rightarrow B$ and $g : B \rightarrow C$.

(The proof of this will be shown on page 92.)

2. We show $f \circ g$ is injective, i.e.,

$$\forall x_0 \in A, x_1 \in A : (f \circ g)(x_0) = (f \circ g)(x_1) \Rightarrow x_0 = x_1.$$

Take arbitrary $x_0 \in A$ and $x_1 \in A$ with $(f \circ g)(x_0) = (f \circ g)(x_1)$. We have to show $x_0 = x_1$.

We know, by definition of \circ , that $g(f(x_0)) = g(f(x_1))$ and thus, because g is injective, $f(x_0) = f(x_1)$. Since f is injective, we then have $x_0 = x_1$.

3. We show $f \circ g$ is surjective, i.e.,

$$\forall z \in C : \exists x \in A : (f \circ g)(x) = z.$$

Take arbitrary $z \in C$.

We have to prove

$$\exists x \in A : (f \circ g)(x) = z.$$

i.e., we have to find some $x \in A$ such that $(f \circ g)(x) = z$.

(We are going to deduce the existence of such an x from knowledge that we have.)

Since g is surjective, there is some $y \in B$ such that $g(y) = z$. Since f is surjective, **there is some $x \in A$ such that $f(x) = y$.**

(We have deduced the existence of a x with particular properties. Now we are going to show that this x also satisfies the property in which we are interested.)

Thus $(f \circ g)(x) = g(f(x)) = g(y) = z$.

The *indirect method* may be applied by assuming $(\neg\exists x : G)$, i.e., $(\forall x : \neg G)$ and deriving a contradiction, see Proposition 29 on page 98:

$$\frac{K}{\exists x : G} \rightsquigarrow \frac{K \cup \{\forall x : \neg G\}}{F(\text{false})}.$$

This is usually indicated as follows:

We have to prove $(\exists x : G)$. Assume $(\forall x : \neg G)$. Then ... (derivation of a contradiction with additional knowledge $(\forall x : \neg G)$).

Theorem Every natural number greater than one is divided by some prime number:

$$\forall x \in \mathbb{N} : x > 1 \Rightarrow \exists y \in \mathbb{N} : y \text{ is prime} \wedge y|x.$$

Proof: Take arbitrary $x \in \mathbb{N}$ such that $x > 1$. We have to prove

$$\exists y \in \mathbb{N} : y \text{ is prime} \wedge y|x.$$

Assume that this does not hold, i.e.,

$$(*) \forall y \in \mathbb{N} : y \text{ is prime} \Rightarrow y \nmid x.$$

There are two cases:

1. x is prime. But $x|x$ **which contradicts (*)**.
2. x is not prime. By definition of primality and since $x > 1$, we can define

$$y := \min_{y \in \mathbb{N}} (1 < y < x \wedge y|x).$$

We again have two cases.

- (a) If y is prime, **we have a contradiction to (*)**.
- (b) If y is not prime, then by definition of primality, there is some $z \in \mathbb{N}$ such that

$$1 < z < y \wedge z|y.$$

But then, since $y < x$ and $y|x$ we also have

$$1 < z < x \wedge z|x$$

which contradicts the definition of y .

7.2 Connectives

In this section, we show decomposition rules that can be applied, if the outermost symbol of a formula is a logical connective.

7.2.1 Equivalences

Proposition 22 (Decomposition of Equivalences) For proving with knowledge K the goal $G_0 \Leftrightarrow G_1$, it suffices to prove both $G_0 \Rightarrow G_1$ and $G_1 \Rightarrow G_0$:

$$\boxed{\begin{array}{c} K \\ G_0 \Leftrightarrow G_1 \end{array}} \rightsquigarrow \boxed{\begin{array}{c} K \\ G_0 \Rightarrow G_1 \end{array}} \boxed{\begin{array}{c} K \\ G_1 \Rightarrow G_0 \end{array}}$$

An equivalence is shown by proving the implication “from left to right” and “from right to left”, sometimes denoted as

We have to prove $G_0 \Leftrightarrow G_1$:

- \Rightarrow : ... (proof of $G_0 \Rightarrow G_1$).
- \Leftarrow : ... (proof of $G_1 \Rightarrow G_0$).

Theorem For every x and y , we have

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

Proof: Take arbitrary x and y . **We have to prove**

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

- **We prove** $x = y \Rightarrow (x \subseteq y \wedge y \subseteq x)$.
- ...
- **We prove** $(x \subseteq y \wedge y \subseteq x) \Rightarrow x = y$.
- ...

(The missing parts of the proof are shown in the following section.)

Theorem For every A, B, C, D and every $R \subseteq A \times B, S \subseteq B \times C,$ and $T \subseteq C \times D,$ we have:

$$(R^{-1})^{-1} = R$$

Proof: Take arbitrary $A, B,$ and $R \subseteq A \times B.$ We prove

$$(1) (R^{-1})^{-1} = R.$$

Because of the definition of $=,$ we have to show

$$(2) \forall r : r \in (R^{-1})^{-1} \Leftrightarrow r \in R.$$

Take arbitrary $r.$

(Because it is clear that we are now going to prove the equivalence, this is not extra stated.)

- **We prove** $r \in (R^{-1})^{-1} \Rightarrow r \in R.$

...

- **We prove** $r \in R \Rightarrow r \in (R^{-1})^{-1}.$

...

(The missing parts of the proof are shown in the following section.)

If we have to show $G_0 \Leftrightarrow G_1 \Leftrightarrow G_2,$ i.e., $(G_0 \Leftrightarrow G_1) \wedge (G_1 \Leftrightarrow G_2),$ it suffices to “traverse the circle”, i.e., to show $(G_0 \Rightarrow G_1) \wedge (G_1 \Rightarrow G_2) \wedge (G_2 \Rightarrow G_0)$ (and analogously for an arbitrary number of equivalences).

7.2.2 Implications

Proposition 23 (Decomposition of Implications) For proving with knowledge K the goal $G_0 \Rightarrow G_1$, it suffices to prove G_1 with additional knowledge G_0 :

$$\frac{K}{G_0 \Rightarrow G_1} \rightsquigarrow \frac{K \cup \{G_0\}}{G_1}$$

Almost every proof contains applications of above rule indicated as

We have to show $G_0 \Rightarrow G_1$. Assume G_0 . Then ... (proof of G_1 with additional knowledge G_0).

Theorem The second law of Peano states

$$\forall x, y : x' = y' \Rightarrow x = y.$$

Proof: Take arbitrary x and y . **We have to prove**

$$x' = y' \Rightarrow x = y.$$

We assume

$$(1) \ x' = y'$$

and show $x = y$.

...

(The rest of the proof was shown on page 68.)

Theorem For every x and y , we have

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

Proof: Take arbitrary x and y . **We have to prove**

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

- We prove $x = y \Rightarrow (x \subseteq y \wedge y \subseteq x)$.

Assume $x = y$, i.e., by definition of '=',

$$(1) \forall z : z \in x \Leftrightarrow z \in y.$$

We have to prove $x \subseteq y \wedge y \subseteq x$.

- We prove $x \subseteq y$, i.e., by definition of ' \subseteq ', $\forall z \in x : z \in y$. Take arbitrary z . **We have to prove** $z \in x \Rightarrow z \in y$. **Assume (2) $z \in x$. We have to prove** $z \in y$ which is a consequence of (1) and (2).
- The proof of $y \subseteq x$ proceeds analogously.

- We prove $(x \subseteq y \wedge y \subseteq x) \Rightarrow x = y$.

Assume $x \subseteq y \wedge y \subseteq x$, i.e., by definition of ' \subseteq '

$$(1) \forall z \in x : z \in y;$$

$$(2) \forall z \in y : z \in x.$$

We prove $x = y$, i.e., by definition of '=', $\forall z : z \in x \Leftrightarrow z \in y$. Take arbitrary z . We have to prove $z \in x \Leftrightarrow z \in y$.

- **We prove** $z \in x \Rightarrow z \in y$. **Assume (3) $z \in x$. We have to prove** $z \in y$ which is a consequence of (1) and (3).
- **We prove** $z \in y \Rightarrow z \in x$. **Assume (4) $z \in y$. We have to prove** $z \in x$ which is a consequence of (2) and (4).

Theorem For every A, B, C, D and every $R \subseteq A \times B, S \subseteq B \times C$, and $T \subseteq C \times D$, we have:

$$(R^{-1})^{-1} = R;$$

Proof: Take arbitrary A, B and $R \subseteq A \times B$. We prove

$$(1) (R^{-1})^{-1} = R.$$

Because of the definition of =, we have to show

$$(2) \forall r : r \in (R^{-1})^{-1} \Leftrightarrow r \in R.$$

Take arbitrary r .

- We prove

$$r \in (R^{-1})^{-1} \Rightarrow r \in R.$$

We assume

$$(3) r \in (R^{-1})^{-1}$$

and show $r \in R$.

We have $R \subseteq A \times B$, therefore we know, by definition of $^{-1}$, that $R^{-1} \subseteq B \times A$ and $(R^{-1})^{-1} \subseteq A \times B$. Thus we can find by (3) some $a \in A$ and $b \in B$ such that $r = \langle a, b \rangle$. By definition of $^{-1}$, we have $\langle b, a \rangle \in R^{-1}$. Again, by definition of $^{-1}$, we know that $\langle a, b \rangle \in R$, therefore $r \in R$.

- The proof of $r \in R \Rightarrow r \in (R^{-1})^{-1}$ proceeds in a similar way.

Because of $(G_0 \Rightarrow G_1) \text{ iff } (\neg G_1 \Rightarrow \neg G_0)$, an alternative is to apply the rule

$$\frac{K}{G_0 \Rightarrow G_1} \rightsquigarrow \frac{K \cup \{\neg G_1\}}{\neg G_0}$$

indicated as

We have to show $G_0 \Rightarrow G_1$. Assume $\neg G_1$. Then ... (proof of $\neg G_0$ with additional knowledge $\neg G_1$).

Because of $\neg(G_0 \Rightarrow G_1) \text{ iff } (G_0 \wedge \neg G_1)$, the indirect method yields

$$\frac{K}{G_0 \Rightarrow G_1} \rightsquigarrow \frac{K \cup \{G_0 \wedge \neg G_1\}}{\text{F(false)}}$$

which is typically indicated as

We have to show $G_0 \Rightarrow G_1$. Assume $G_0 \wedge \neg G_1$. Then we have ... (derivation of a contradiction).

7.2.3 Conjunctions

Proposition 24 (Decomposition of Conjunctions) For proving with knowledge K the goal $G_0 \wedge G_1$, it suffices to prove both G_0 and G_1 :

$$\boxed{\begin{array}{c} K \\ G_0 \wedge G_1 \end{array}} \rightsquigarrow \boxed{\begin{array}{c} K \\ G_0 \end{array}} \boxed{\begin{array}{c} K \\ G_1 \end{array}}$$

A conjunction is shown by showing both conjuncts in turn:

We have to show $G_0 \wedge G_1$.

1. ... (proof of G_0).
2. ... (proof of G_1).

Theorem For every x and y , we have

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

Proof: Take arbitrary x and y . We have to prove

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

- We prove $x = y \Rightarrow (x \subseteq y \wedge y \subseteq x)$.
Assume $x = y$, i.e., by definition of '=',

$$(1) \forall z : z \in x \Leftrightarrow z \in y.$$

We have to prove $x \subseteq y \wedge y \subseteq x$.

- **We prove $x \subseteq y$.** ...
- **We prove $y \subseteq x$.** ...
- We prove $(x \subseteq y \wedge y \subseteq x) \Rightarrow x = y$.
...

(The rest of the proof was shown on page 82.)

Because of $\neg(G_0 \wedge G_1)$ iff $\neg G_0 \vee \neg G_1$, the *indirect method* leads to

$$\frac{K}{G_0 \wedge G_1} \rightsquigarrow \frac{K \cup \{\neg G_0 \vee \neg G_1\}}{F(\text{alse})} \rightsquigarrow \frac{K \cup \{\neg G_0\}}{F(\text{alse})} \quad \frac{K \cup \{\neg G_1\}}{F(\text{alse})}$$

applying the technique of “case distinction” explained on page 95:

We have to prove $G_0 \wedge G_1$.

1. Assume $\neg G_0$. Then ... (derivation of contradiction with additional assumption $\neg G_0$).
2. Assume $\neg G_1$. Then ... (derivation of contradiction with additional assumption $\neg G_1$).

7.2.4 Disjunctions

Proposition 25 (Decomposition of Disjunctions) For proving with knowledge K the goal $G_0 \vee G_1$, it suffices to prove G_1 with additional knowledge $\neg G_0$.

$$\frac{K}{G_0 \vee G_1} \rightsquigarrow \frac{K \cup \{\neg G_0\}}{G_1}$$

This rule is a consequence of “ $(G_0 \vee G_1)$ iff $(\neg G_0 \Rightarrow G_1)$ ” such that the same techniques can be applied as for the decomposition of implications (see Proposition 23):

We have to show $G_0 \vee G_1$. Assume $\neg G_0$. Then ... (proof of G_1).

Of course, the roles of G_0 and G_1 can be inverted.

Frequently, in a particular proof situation, the additional assumption is not required and one simply says

We have to show $G_0 \vee G_1$. We have ... (proof of G_0 or of G_1).

Theorem We prove

$$\forall A, B : A \cup B \neq \emptyset \Rightarrow A \neq \emptyset \vee B \neq \emptyset.$$

Proof: Take arbitrary A and B and assume

$$(0) A \cup B \neq \emptyset.$$

We show $A \neq \emptyset \vee B \neq \emptyset$.

Assume

$$(1) A = \emptyset.$$

We show $B \neq \emptyset$.

From (0) and the definition of \cup , we know

$$\{x : x \in A \vee x \in B\} \neq \emptyset$$

i.e., by the definition of \emptyset ,

$$\exists x : x \in A \vee x \in B.$$

Let x be such that

$$(2) x \in A \vee x \in B.$$

From (1) and the definition of \emptyset , we know

$$(3) x \notin A$$

From (2) and (3), we know

$$(4) x \in B$$

thus $B \neq \emptyset$.

7.3 Defined Constants

A proposition usually contains predicate and function symbols that are not part of the basic theory but that have been introduced by *definitions*. In the course of the proof, we usually have to resort to the knowledge that has been introduced by the definition, namely that a predicate/function application is equivalent/equal to an instance of the defining formula/term.

The content of this section can be summarized in a nutshell: when you don't know how to proceed with a predicate or function, **insert its definition!**

7.3.1 Predicates

Proposition 26 (Explicitly Defined Predicates) For proving an atomic formula $p(a_0, \dots, a_{n-1})$ where p is predicate explicitly defined as

$$p(x_0, \dots, x_{n-1}) :\Leftrightarrow G,$$

and a_0, \dots, a_{n-1} are terms, it suffices to prove $G[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]$:

$$\frac{K \cup \{\forall x_0, \dots, x_{n-1} : p(x_0, \dots, x_{n-1}) \Leftrightarrow G\}}{p(a_0, \dots, a_{n-1})} \rightsquigarrow$$

$$\frac{K \cup \{\forall x_0, \dots, x_{n-1} : p(x_0, \dots, x_{n-1}) \Leftrightarrow G\}}{G[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]}$$

We may thus just insert the definition of a predicate into a goal formula:

We have to prove $p(a_0, \dots, a_{n-1})$. By definition of p , it suffices to prove $G[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]$.

Theorem For every x and y , we have

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

Proof: Take arbitrary x and y . We have to prove

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

- We prove $x = y \Rightarrow (x \subseteq y \wedge y \subseteq x)$.

Assume $x = y$, i.e., by definition of ‘=’,

$$(1) \forall z : z \in x \Leftrightarrow z \in y.$$

We have to prove $x \subseteq y \wedge y \subseteq x$.

- **We prove $x \subseteq y$, i.e., by definition of ‘ \subseteq ’,**

$$\forall z \in x : z \in y.$$

Take arbitrary z . We have to prove $z \in x \Rightarrow z \in y$. Assume (2) $z \in x$. We have to prove $z \in y$ which is a consequence of (1) and (2).

- The proof of $y \subseteq x$ proceeds analogously.

- We prove $(x \subseteq y \wedge y \subseteq x) \Rightarrow x = y$.

...

(The remainder of the proof was shown on page 82.)

In the next section, we will show another proof, where this principle is extensively applied.

7.3.2 Functions

Proposition 27 (Explicitly Defined Functions) For proving the goal $G[x \leftarrow F(a_0, \dots, a_{n-1})]$ where F is a function explicitly defined as

$$F(x_0, \dots, x_{n-1}) := T,$$

it suffices to prove $G[x \leftarrow T[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]]$ (for some terms T, a_0, \dots, a_{n-1}):

$$\frac{K \cup \{\forall x_0, \dots, x_{n-1} : F(x_0, \dots, x_{n-1}) = T\}}{G[x \leftarrow F(a_0, \dots, a_{n-1})]} \rightsquigarrow$$

$$\frac{K \cup \{\forall x_0, \dots, x_{n-1} : F(x_0, \dots, x_{n-1}) = T\}}{G[x \leftarrow T[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]]}$$

Above rule simply tells us that, if a goal contains an elementary term with an explicitly defined function, we are allowed to “insert the definition” of the function:

We have to prove $G[x \leftarrow F(a_0, \dots, a_{n-1})]$. By definition of F , it suffices to prove $G[x \leftarrow T[x_0 \leftarrow a_0, \dots, x_{n-1} \leftarrow a_{n-1}]]$.

Theorem

$$\forall A, B : A = \emptyset \Rightarrow A \cap B = \emptyset.$$

Proof: Take arbitrary A and B and assume

$$A = \emptyset.$$

We show $A \cap B = \emptyset$, i.e., by **definition of \cap** ,

$$\{x : x \in A \wedge x \in B\} = \emptyset.$$

By the axiom of the empty set and the equality of sets, it suffices to prove

$$\neg \exists x : x \in A \wedge x \in B$$

Assume that this does not hold. Then we have some x such that

$$x \in A \wedge x \in B$$

and thus $x \in A$ which contradicts $A = \emptyset$.

Theorem The composition of two functions is also a function, i.e., for all A , B , and C , and all $f : A \rightarrow B$ and $g : B \rightarrow C$, we have:

$$f \circ g : A \rightarrow C.$$

Take arbitrary A , B , C , $f : A \rightarrow B$, and $g : B \rightarrow C$. We have to prove $f \circ g : A \rightarrow C$, i.e., by **definition of the predicate \rightarrow** , that

- (1) $(f \circ g) : A \xrightarrow{\text{partial}} C$;
- (2) $\forall x \in A : \exists y \in C : \langle x, y \rangle \in (f \circ g)$.

- We prove (1), i.e., **by definition of** $\xrightarrow{\text{partial}}$, that

$$(3) (f \circ g) \subseteq A \times C;$$

$$(4) \forall x, y_0, y_1 : (\langle x, y_0 \rangle \in (f \circ g) \wedge \langle x, y_1 \rangle \in (f \circ g)) \Rightarrow y_0 = y_1.$$

We know (3) **from the definition of** \circ ; we still have to show (4). Take arbitrary x, y_0, y_1 and assume

$$(5) \langle x, y_0 \rangle \in (f \circ g);$$

$$(6) \langle x, y_1 \rangle \in (f \circ g).$$

We have to show $y_0 = y_1$.

From (5), (6), and the definition of \circ , we know $y_0 \in C, y_1 \in C$, and

$$(7) \exists b \in B : \langle x, b \rangle \in f \wedge \langle b, y_0 \rangle \in g;$$

$$(8) \exists b \in B : \langle x, b \rangle \in f \wedge \langle b, y_1 \rangle \in g.$$

By (7), we have some $b_0 \in B$ such that $\langle x, b_0 \rangle \in f \wedge \langle b_0, y_0 \rangle \in g$; by (8), we have some $b_1 \in B$ such that $\langle x, b_1 \rangle \in f \wedge \langle b_1, y_1 \rangle \in g$. From $\langle x, b_0 \rangle \in f, \langle x, b_1 \rangle \in f$ and the fact that f is a function, we know that $b_0 = b_1$. From this and from $\langle b_0, y_0 \rangle \in g, \langle b_1, y_1 \rangle \in g$, and the fact that g is a function, we know that $y_0 = y_1$.

- We prove (2). Take arbitrary (3) $x \in A$. We have to show

$$(4) \exists y \in C : \langle x, y \rangle \in (f \circ g).$$

From (3) and $f : A \rightarrow B$, we know (5) $\langle x, f(x) \rangle \in f$ and (6) $f(x) \in B$. From (6) and $g : B \rightarrow C$, we know (7) $\langle f(x), g(f(x)) \rangle \in g$ and (8) $g(f(x)) \in C$.

To show (4), we take (9) $y := g(f(x))$ and show

$$(10) y \in C;$$

$$(11) \langle x, y \rangle \in (f \circ g).$$

We know (10) from (8) and (9). To show (11), **we have to show, by definition of** \circ , **that**

$$(12) x \in A \wedge y \in C;$$

$$(13) \exists b : \langle x, b \rangle \in f \wedge \langle b, y \rangle \in g.$$

We know (12) from (3), (8), and (9). To show (13), we take (14) $b := f(x)$ and have to show:

$$(15) \langle x, b \rangle \in f;$$

$$(16) \langle b, y \rangle \in g.$$

We know (15) from (5) and (14). We know (16) from (7), (9), and (14).

Chapter 8

Deriving New Knowledge

At some point (at least if the goal is an atomic formula with a predicate of the underlying theory), we have to start to operate with the available knowledge in order to derive a situation in which the goal is part of the knowledge.

8.1 Case Distinctions

Proposition 28 (Proof by Case Distinction) For proving with knowledge K the goal G , it suffices to prove G with additional knowledge F and to prove G with additional knowledge $\neg F$ (for some formula F).

$$\frac{\boxed{K}}{\boxed{G}} \rightsquigarrow \frac{\boxed{K \cup \{F\}}}{\boxed{G}} \quad \frac{\boxed{K \cup \{\neg F\}}}{\boxed{G}}$$

A proof by case distinction decomposes the “universe of situations” into those where a particular assumption F holds and those where it does not hold:

We have to prove G .

1. Assume F . Then ... (proof of G with additional knowledge F).
2. Assume $\neg F$. Then ... (proof of G with additional knowledge $\neg F$).

Lemma: $1 = 0$.

Proof 1:

Take any x and y such that $x = y$.

$$x^2 = xy.$$

$$x^2 - y^2 = xy - y^2.$$

$$(x + y)(x - y) = y(x - y).$$

$$x + y = y.$$

$$2y = y.$$

$$2 = 1.$$

$$1 = 0.$$

Proof 2:

$$-2 = -2.$$

$$4 - 6 = 1 - 3.$$

$$4 - 6 + 9/4 = 1 - 3 + 9/4.$$

$$(2 - 3/2)^2 = (1 - 3/2)^2.$$

$$2 - 3/2 = 1 - 3/2.$$

$$2 = 1.$$

$$1 = 0.$$

Theorem Every natural number greater than one is divided by some prime number:

$$\forall x \in \mathbb{N} : x > 1 \Rightarrow \exists y \in \mathbb{N} : y \text{ is prime} \wedge y|x.$$

Proof: Take arbitrary $x \in \mathbb{N}$ such that $x > 1$. We have to prove

$$\exists y \in \mathbb{N} : y \text{ is prime} \wedge y|x.$$

Assume that this does not hold, i.e.,

$$(*) \forall y \in \mathbb{N} : y \text{ is prime} \Rightarrow y \nmid x.$$

There are two cases:

1. **x is prime.** But $x|x$ which contradicts (*).
2. **x is not prime.** By definition of primality and since $x > 1$, we can define

$$y := \min_{y \in \mathbb{N}} (1 < y < x \wedge y|x).$$

We again have two cases.

- (a) **If y is prime,** we have a contradiction to (*).
- (b) **If y is not prime,** then by definition of primality, there is some $z \in \mathbb{N}$ such that

$$1 < z < y \wedge z|y.$$

But then, since $y < x$ and $y|x$ we also have

$$1 < z < x \wedge z|x$$

which contradicts the definition of y .

Frequently a case distinction is introduced in situations where we have a formula $(F_0 \vee \dots \vee F_{n-1})$ in our knowledge:

$$\frac{K \cup \{F_0 \vee \dots \vee F_{n-1}\}}{G} \rightsquigarrow \frac{K \cup \{F_0\}}{G} \cdots \frac{K \cup \{F_{n-1}\}}{G}$$

which is indicated as

We have to prove G . Since we know $(F_0 \vee \dots \vee F_{n-1})$, it suffices to consider the following cases:

- Case F_0 : ... (proof of G with additional knowledge F_0).
- ...
- Case F_{n-1} : ... (proof of G with additional knowledge F_{n-1}).

Theorem $\forall A, B, C : A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Proof: Take arbitrary A, B, C . By the axiom of the equality of sets, it suffices to prove

$$\forall x : x \in A \cup (B \cap C) \Leftrightarrow x \in (A \cup B) \cap (A \cup C).$$

Take arbitrary x .

- We prove $x \in A \cup (B \cap C) \Rightarrow x \in (A \cup B) \cap (A \cup C)$.

Assume $x \in A \cup (B \cap C)$, i.e., by the definition of \cup ,

$$(1) \quad x \in A \vee x \in B \cap C.$$

We prove $x \in (A \cup B) \cap (A \cup C)$. By definition of \cap , it suffices to prove $x \in A \cup B \wedge x \in A \cup C$.

– We prove $x \in A \cup B$. **From (1), we have two cases:**

- * **Case $x \in A$:** by definition of \cup , we are done.
- * **Case $x \in B \cap C$:** from definition of \cap , we know $x \in B$ and, by definition of \cup , are done.

– We prove $x \in A \cup C$. **From (1), we have two cases:**

- * **Case $x \in A$:** we are done.
- * **Case $x \in B \cap C$:** from definition of \cap , we know $x \in C$ and are, by definition of \cup , done.

- We prove $x \in (A \cup B) \cap (A \cup C) \Rightarrow x \in A \cup (B \cap C)$.

Assume $x \in (A \cup B) \cap (A \cup C)$, i.e., by the definition of \cup ,

$$(2) \quad x \in (A \cup B) \wedge x \in (A \cup C).$$

By the definition of \cup , it suffices to prove $x \in A \vee x \in (B \cap C)$. We assume

$$(3) x \notin A$$

and prove $x \in (B \cap C)$. By definition of \cap , it suffices to prove $x \in B \wedge x \in C$.

- We prove $x \in B$. From (2), we know $x \in (A \cup B)$. From the definition of \cup , we thus know $x \in A \vee x \in B$ and with (3) $x \in B$.
- We prove $x \in C$. From (2), we know $x \in (A \cup C)$. From the definition of \cup , we thus know $x \in A \vee x \in C$ and with (3) $x \in C$.

8.2 Universal Formulas in Knowledge

Proposition 29 (Universal Formula in Knowledge) For proving with knowledge $K \cup \{\forall x : F\}$ the goal G , it suffices to prove G with additional knowledge $F[x \leftarrow T]$ for any term T :

$$\frac{K \cup \{\forall x : F\}}{G} \rightsquigarrow \frac{K \cup \{\forall x : F, F[x \leftarrow T]\}}{G}$$

A universal formula $(\forall x : F)$ in the knowledge base is a “machine” that takes any T and produces additional knowledge $F[x \leftarrow T]$. Whenever we are in need of an arbitrary instance of F in our knowledge base, we can start this machine:

We have to prove G . Since we know $(\forall x : F)$, we have $F[x \leftarrow T]$ and thus ... (proof of G with additional knowledge $F[x \leftarrow T]$).

Theorem For every x and y , we have

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

Proof: Take arbitrary x and y . We have to prove

$$x = y \Leftrightarrow (x \subseteq y \wedge y \subseteq x).$$

- We prove $x = y \Rightarrow (x \subseteq y \wedge y \subseteq x)$.

Assume $x = y$, i.e., by definition of '=',

$$(1) \forall w : w \in x \Leftrightarrow w \in y.$$

We have to prove $x \subseteq y \wedge y \subseteq x$.

- We prove $x \subseteq y$, i.e., by definition of ' \subseteq ', $\forall z \in x : z \in y$. Take arbitrary z . We have to prove $z \in x \Rightarrow z \in y$. Assume

$$(2) z \in x.$$

We have to prove $z \in y$ **which follows from (1) and (2)**.

- The proof of $y \subseteq x$ proceeds analogously.

- We prove $(x \subseteq y \wedge y \subseteq x) \Rightarrow x = y$.

Assume $x \subseteq y \wedge y \subseteq x$, i.e., by definition of ' \subseteq '

$$(1) \forall w \in x : w \in y;$$

$$(2) \forall w \in y : w \in x.$$

We prove $x = y$, i.e., by definition of '=', $\forall z : z \in x \Leftrightarrow z \in y$. Take arbitrary z . We have to prove $z \in x \Leftrightarrow z \in y$.

- We prove $z \in x \Rightarrow z \in y$. Assume

$$(3) z \in x.$$

We have to prove $z \in y$ **which follows from (1) and (3)**.

- We prove $z \in y \Rightarrow z \in x$. Assume

$$(4) z \in y.$$

We have to prove $z \in x$ **which follows from (2) and (4)**.

8.3 Existential Formulas in Knowledge

Proposition 30 (Existential Formula in Knowledge) For proving with knowledge $K \cup \{\exists x : F\}$ the goal G , it suffices to prove G with additional knowledge $F[x \leftarrow a]$ for some object constant a that does not appear in K , G , or F :

$$\frac{K \cup \{\exists x : F\}}{G} \rightsquigarrow \frac{K \cup \{\exists x : F, F[x \leftarrow a]\}}{G} \quad (a \text{ not in } K \cup \{G, F\})$$

An existential formula $(\exists x : F)$ in the knowledge base is an “engine” which returns a constant a about the only thing we know is $F[x \leftarrow a]$. Whenever we are in need of such a constant, we can invoke the machine:

We have to prove G . Since we know $(\exists x : F)$, we have some a with $F[x \leftarrow a]$. Thus ... (proof of G with additional knowledge $F[x \leftarrow a]$ for some new constant a).

Theorem There is no square root of 2 in \mathbb{Q} , i.e.,

$$\neg \exists x \in \mathbb{Q} : x^2 = 2.$$

Proof: We assume

$$\exists x \in \mathbb{Q} : x^2 = 2.$$

and show a contradiction. Take $x \in \mathbb{Q}$ such that

$$(1) \quad x^2 = 2$$

We know

$$\mathbb{Q} := \{y : \exists a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\} : y := a/b \wedge a \text{ and } b \text{ are relatively prime}\}.$$

From $x \in \mathbb{Q}$, we know

$$\exists a \in \mathbb{Z}, b \in \mathbb{Z} \setminus \{0\} : x := a/b \wedge a \text{ and } b \text{ are relatively prime.}$$

Thus we have some $a \in \mathbb{Z}$ and $b \in \mathbb{Z} \setminus \{0\}$ such that $x = \frac{a}{b}$ and a and b are relatively prime, i.e., by definition,

$$(2) \neg \exists x \in \mathbb{Z} \setminus \{-1, 1\} : x|a \wedge x|b.$$

...

(The remainder of the proof was shown on page 67.)

Theorem

$$\forall A, B : A = \emptyset \Rightarrow A \cap B = \emptyset.$$

Proof: Take arbitrary A and B and assume

$$A = \emptyset.$$

We show $A \cap B = \emptyset$, i.e., by definition of \cap ,

$$\{x : x \in A \wedge x \in B\} = \emptyset.$$

By the axiom of the empty set and the equality of sets, it suffices to prove $\neg \exists x : x \in A \wedge x \in B$. Assume that this does not hold, i.e.,

$$\exists x : x \in A \wedge x \in B$$

Then we have some x such that

$$x \in A \wedge x \in B$$

and thus $x \in A$ which contradicts $A = \emptyset$.

Theorem The composition of two functions is also a function, i.e., for all A , B , and C , and all $f : A \rightarrow B$ and $g : B \rightarrow C$, we have:

$$f \circ g : A \rightarrow C.$$

Take arbitrary A , B , C , $f : A \rightarrow B$, and $g : B \rightarrow C$. We have to prove $f \circ g : A \rightarrow C$, i.e., by definition of the predicate \rightarrow , that

- (1) $(f \circ g) : A \xrightarrow{\text{partial}} C$;
- (2) $\forall x \in A : \exists y \in C : \langle x, y \rangle \in (f \circ g)$.

- We prove (1), i.e., by definition of $\xrightarrow{\text{partial}}$, that

$$(3) (f \circ g) \subseteq A \times C;$$

$$(4) \forall x, y_0, y_1 : (\langle x, y_0 \rangle \in (f \circ g) \wedge \langle x, y_1 \rangle \in (f \circ g)) \Rightarrow y_0 = y_1.$$

We know (3) from the definition of \circ ; we still have to show (4). Take arbitrary x, y_0, y_1 and assume

$$(5) \langle x, y_0 \rangle \in (f \circ g);$$

$$(6) \langle x, y_1 \rangle \in (f \circ g).$$

We have to show $y_0 = y_1$.

From (5), (6), and the definition of \circ , we know $y_0 \in C, y_1 \in C$, and

$$(7) \exists b \in B : \langle x, b \rangle \in f \wedge \langle b, y_0 \rangle \in g;$$

$$(8) \exists b \in B : \langle x, b \rangle \in f \wedge \langle b, y_1 \rangle \in g.$$

By (7), we have some $b_0 \in B$ such that $\langle x, b_0 \rangle \in f \wedge \langle b_0, y_0 \rangle \in g$; by (8), we have some $b_1 \in B$ such that $\langle x, b_1 \rangle \in f \wedge \langle b_1, y_1 \rangle \in g$. From $\langle x, b_0 \rangle \in f, \langle x, b_1 \rangle \in f$ and the fact that f is a function, we know that $b_0 = b_1$. From this and from $\langle b_0, y_0 \rangle \in g, \langle b_1, y_1 \rangle \in g$, and the fact that g is a function, we know that $y_0 = y_1$.

- ...

(The rest of the proof was shown on page 92.)

8.4 Inferring Additional Knowledge

Proposition 31 (Additional Knowledge) For proving with knowledge K the goal G , it suffices to prove G with additional knowledge F , if F holds in every domain in which (some of) the formulas in K hold.

$$\frac{K}{G} \rightsquigarrow \frac{K \cup \{F\}}{G} \quad (F \text{ holds in every domain in which } K \text{ holds}).$$

This last rule is a “placeholder” for a number of ways to prove

$$\frac{K}{F}$$

(respectively $\frac{S}{F}$ for some $S \subseteq K$):

1. This rule has been shown in a previous proof or is shown as a subproof.
2. This holds because F is a *propositional consequence* of K , i.e., the conclusion holds independently of the truth values of the atomic formulas and quantified formulas contained in K and F .
3. This is an instance of some *quantifier consequence* which give true conclusions in every domain.
4. This is derived by applying *substitution* rules from known equalities and equivalences.

In the first way, we can decompose a proof in a *modular* way into a number of smaller proofs or reuse the knowledge represented by previously proved propositions.

Theorem We prove that there exist infinitely many prime numbers, i.e., that there is no largest prime number:

$$\forall x \in \mathbb{N} : \exists p \in \mathbb{N} : p > x \wedge p \text{ is prime.}$$

Assume that there is some $x \in \mathbb{N}$ such that

$$(1) \forall p \in \mathbb{N} : p \text{ is prime} \Rightarrow p \leq x.$$

(The negation of the goal is $(\exists x \in \mathbb{N} : \forall p \in \mathbb{N} : p \leq x \vee p \text{ is not prime})$ i.e., $\exists x \in \mathbb{N} : \forall p \in \mathbb{N} : p \text{ is not prime} \Rightarrow p \leq x$).

Define $a := 1 + \prod_{1 \leq i \leq x} i$. Now we have two possibilities:

1. a is prime. But from the definition of a we know $a > x$, thus we have a contradiction to (1).

2. a is not prime. Then we can define

$$i := \min_{i \in \mathbb{N}} 1 < i < a \wedge i|a.$$

(because we know $\exists i \in \mathbb{N} : 1 < i < a \wedge i|a$).

From the definition of a , we know

$$\forall i \in \mathbb{N} : 1 \leq i \leq x \Rightarrow i \nmid a.$$

therefore

$$(2) \ i > x.$$

It remains to be shown that

$$(3) \ i \text{ is prime}$$

which together with (2) contradicts (1).

(The proof of (3) follows separately.)

The other ways are explained in the following subsections.

8.4.1 Propositional Consequences

The conclusion

$\neg\neg\forall x : \exists y : p(x, y)$
$\forall x : \exists y : p(x, y)$

holds independently of the interpretation of p and consequently independently of the truth value of $(\forall x : \exists y : p(x, y))$. This is because this conclusion is an instance of the pattern

$\neg\neg G$
G

which is true independently of the truth value of G . We call such a conclusion a *propositional consequence* (*aussagenlogische Folge*).

Proposition 32 (Propositional Consequences) The following conclusions are propositional consequences for every formula A and B :

Negation

$\neg\neg A$	A
A	$\neg\neg A$

And Introduction and Or Elimination

$A \wedge B$	A
A	$A \vee B$

De Morgan

$\neg(A \wedge B)$	$\neg(A \vee B)$	$\neg A \vee \neg B$	$\neg A \wedge \neg B$
$\neg A \vee \neg B$	$\neg A \wedge \neg B$	$\neg(A \wedge B)$	$\neg(A \vee B)$

Modus Ponens

$A, A \Rightarrow B$
B

Contraposition

$A \Rightarrow B$	$\neg A \Rightarrow \neg B$	$A \Leftrightarrow B$	$\neg A \Leftrightarrow \neg B$
$\neg B \Rightarrow \neg A$	$B \Rightarrow A$	$\neg A \Leftrightarrow \neg B$	$A \Leftrightarrow B$

There are (infinitely) many other propositional consequences. A general strategy to show that $\frac{A}{B}$ is a propositional consequence is to show that $A \Rightarrow B$ is a propositional tautology.

A tautology is a thing which is tautological.

Proposition 33 (Tautology) A propositional formula with variables is a *propositional tautology* (*aussagenlogische Tautologie*) if it is true for every assignment of truth values to the variables.

We can show that a propositional formula with variables is a tautology by constructing a *truth table* or by applying the *indirect method*.

Theorem

$$((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C.$$

We show that this formula is a tautology by constructing a truth table:

A	B	C	$A \vee B$	$A \Rightarrow C$	$B \Rightarrow C$	Conjunction	Implication
F	F	F	F	T	T	F	T
F	F	T	F	T	T	F	T
F	T	F	T	T	F	F	T
F	T	T	T	T	T	T	T
T	F	F	T	F	T	F	T
T	F	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	T	T	T	T	T	T	T

Theorem

$$((A \vee B) \wedge (A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow C.$$

We show that this formula is a tautology by assuming that its truth value is false and then deriving a contradiction:

$$\frac{\frac{\frac{\text{false}}{\text{true}}}{\text{true}} \wedge \left(\frac{\text{true}}{\text{false}} \Rightarrow \frac{\text{false}}{\text{false}} \right) \wedge \left(\frac{\text{true}}{\text{false}} \Rightarrow \frac{\text{false}}{\text{false}} \right)}{\text{false}} \Rightarrow \frac{\text{false}}{\text{false}} C$$

Because the implication is false, C is false and the conjuncts are true. Thus A and B must be false (since C is false, the implications cannot be true otherwise). Therefore $A \vee B$ is false, which contradicts above derivation.

8.4.2 Quantifier Consequences

There are various patterns of conclusions that occur in proofs so frequently that they are considered as “basic knowledge”.

Proposition 34 (Quantifier Consequences) For every formula A and B , the following conclusions hold:

Universal Quantification and Conjunction

$(\forall x : A \wedge B)$	$(\forall x : A) \wedge (\forall x : B)$
$(\forall x : A) \wedge (\forall x : B)$	$(\forall x : A \wedge B)$

Existential Quantification and Disjunction

$(\exists x : A \vee B)$	$(\exists x : A) \vee (\exists x : B)$
$(\exists x : A) \vee (\exists x : B)$	$(\exists x : A \vee B)$

Universal and Disjunction, Existential and Conjunction

$(\forall x : A) \vee (\forall x : B)$	$(\exists x : A \wedge B)$
$(\forall x : A \vee B)$	$(\exists x : A) \wedge (\exists x : B)$

Universal and Existential Quantification

$\exists x : \forall y : A$
$\forall y : \exists x : A$

De Morgan Laws

$\neg \forall x : A$	$\exists x : \neg A$	$\neg \exists x : A$	$\forall x : \neg A$
$\exists x : \neg A$	$\neg \forall x : A$	$\forall x : \neg A$	$\neg \exists x : A$

Such Quantifier

$\exists x : A$
$A[x \leftarrow \mathbf{such} \ x : A]$
$(\forall y_0, y_1 : (A[x \leftarrow y_0] \wedge A[x \leftarrow y_1]) \Rightarrow y_0 = y_1)$
$(\forall x : A \Rightarrow x = \mathbf{such} \ x : A)$

The last two facts about the such quantifier tell us that

1. there must exist some x such that A holds before we are allowed to conclude that $(\mathbf{such} \ x : A)$ satisfies A ;
2. to prove that some y with $A[x \leftarrow y]$ equals $(\mathbf{such} \ x : A)$, we also have to show that only one such y exists.

These conclusions can be proved by the proving techniques explained before; an example is given below.

Proof We show for arbitrary formula A

$$(\neg \forall x : A) \Rightarrow (\exists x : \neg A)$$

by showing (contraposition)

$$(\neg \exists x : \neg A) \Rightarrow (\neg \neg \forall x : A)$$

i.e. (propositional consequence and substitution, see next subsection)

$$(\neg \exists x : \neg A) \Rightarrow (\forall x : A).$$

We assume

$$(*) \neg \exists x : \neg A$$

and show $\forall x : A$. Take arbitrary and assume $\neg A$. Then we have $(\exists x : \neg A)$ which contradicts (*).

8.4.3 Substitutions

The equality axioms and the properties of logical equivalence allow us to draw the following conclusions.

Proposition 36 (Substitutions) For all terms S and T , formulas A and B , variables x and formula patterns C with variable F , the following holds:

Equality Substitutions

$S = T \wedge A[x \leftarrow S]$
$A[x \leftarrow T]$

Equivalence Substitutions

$A \Leftrightarrow B \wedge C[F \leftarrow A]$
$C[F \leftarrow B]$

The substitution laws allow us to replace “terms by equal terms” and “formulas by equivalent formulas”. These laws are applied when we say

We know $A[x \leftarrow S]$. Because $S = T$, we know $A[x \leftarrow T]$.
 We know $C[F \leftarrow A]$. Because $A \Leftrightarrow B$, we know $C[F \leftarrow B]$.

which is frequently done when we have a formula in our knowledge base that refers to a defined function or predicate constant.

Theorem $\forall A, B, C : A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Proof: Take arbitrary A, B, C . By the axiom of the equality of sets, it suffices to prove

$$\forall x : x \in A \cup (B \cap C) \Leftrightarrow x \in (A \cup B) \cap (A \cup C).$$

Take arbitrary x .

- We prove $x \in A \cup (B \cap C) \Rightarrow x \in (A \cup B) \cap (A \cup C)$.

Assume $x \in A \cup (B \cap C)$, i.e., **by the definition of \cup** ,

$$(1) \ x \in A \vee x \in B \cap C.$$

We prove $x \in (A \cup B) \cap (A \cup C)$. By definition of \cap , it suffices to prove $x \in A \cup B \wedge x \in A \cup C$.

- We prove $x \in A \cup B$. From (1), we have two cases:
 - * Case $x \in A$: by definition of \cup , we are done.
 - * Case $x \in B \cap C$: from definition of \cap , we know $x \in B$ and, by definition of \cup , are done.
- We prove $x \in A \cup C$. From (1), we have two cases:
 - * Case $x \in A$: we are done.
 - * Case $x \in B \cap C$: from definition of \cap , we know $x \in C$ and are, by definition of \cup , done.

- We prove $x \in (A \cup B) \cap (A \cup C) \Rightarrow x \in A \cup (B \cap C)$.

Assume $x \in (A \cup B) \cap (A \cup C)$, i.e., **by the definition of \cup** ,

$$(2) \ x \in (A \cup B) \wedge x \in (A \cup C).$$

By the definition of \cup , it suffices to prove $x \in A \vee x \in (B \cap C)$. We assume

$$(3) \ x \notin A$$

and prove $x \in (B \cap C)$. By definition of \cap , it suffices to prove $x \in B \wedge x \in C$.

- We prove $x \in B$. From (2), we know $x \in (A \cup B)$. **From the definition of \cup** , we thus know $x \in A \vee x \in B$ and with (3) $x \in B$.
 - We prove $x \in C$. From (2), we know $x \in (A \cup C)$. **From the definition of \cup** , we thus know $x \in A \vee x \in C$ and with (3) $x \in C$.
-
-

Chapter 9

Example

We conclude these notes by demonstrating in detail the proof of a proposition which is due to the Greek mathematician Euclid and which is the basis for *Euclid's algorithm* for efficiently computing the greatest common divisor of two natural numbers.

First we give the definitions

$$x|y :\Leftrightarrow \exists z \in \mathbb{N} : x * z = y;$$
$$\text{gcd}(x, y) := \mathbf{such} \ z \in \mathbb{N} : z|x \wedge z|y \wedge (\forall g : (g|x \wedge g|y) \Rightarrow g \leq z).$$

Our goal is to prove

$$\forall m \in \mathbb{N}, n \leq m : \text{gcd}(m, n) = \text{gcd}(m - n, n),$$

i.e., the greatest common divisor of m and n is also the greatest common divisor of $m - n$ and n .

An argument as it can be found in a typical text book is the following.

Proof Let g be the gcd of $m - n$ and n . Since g divides $m - n$ and g divides n , g also divides $(m - n) + n = m$.

Take h such that h divides m and h divides n . Then h also divides $m - n$. Since g is the gcd of m and $m - n$, h is less than or equal g .

This “proof skeleton” captures on a very high-level the *key idea* that represents the basis of a “real” proof. However, this idea is only understandable

with the knowledge of the rules introduced in the previous sections. Then it becomes also clear that the skeleton actually reflects the structure of a more detailed proof such as the one given below (with additional explanations that are not part of the proof added in *italics*).

Proof Take arbitrary $m \in \mathbb{N}$ and $n \leq m$.

(We have to prove a universally quantified implication. We introduce arbitrary constants m and n and add the implication antecedent $(m \in \mathbb{N} \wedge n \leq m)$ to our knowledge. We now have to show the goal $\text{gcd}(m, n) = \text{gcd}(m - n)$.)

If $m = 0 \wedge n = 0$, we are done.

(We proceed by case distinction. In the first case $m = 0 \wedge n = 0$, we know $\text{gcd}(m, n) = \text{gcd}(0, 0) = \text{gcd}(0 - 0, 0) = \text{gcd}(m - n)$. By transitivity of $=$, we have the goal.)

Otherwise, $m \neq 0 \vee n \neq 0$. We can show that in this case there exists a unique $z \in \mathbb{N}$ with

$$z|m \wedge z|n \wedge (\forall g : (g|m \wedge g|n) \Rightarrow g \leq z).$$

(The rest of the proof deals with the second case $m \neq 0 \vee n \neq 0$. We claim that in this case above formula holds (which requires a separate proof) and use this as knowledge from now on.)

For every $z' \in \mathbb{N}$, in order to prove

$$z' = \text{gcd}(m, n),$$

it therefore suffices to prove

$$z'|m \wedge z'|n \wedge (\forall g : (g|m \wedge g|n) \Rightarrow g \leq z').$$

(We justify a reasoning step that will be used below: we have to prove that a particular object equals $\text{gcd}(m, n)$. By the rule of substitution, we may insert the definition of gcd and thus have to prove that the object equals the value of a “such term”. We thus remind the reader of the proof rule for this kind of terms on page 107 and instantiate the rule with the concrete formula. The application of this rule requires that only one object satisfies the formula in the “such term”, which is exactly what we have added as knowledge above.)

Let $g := \text{gcd}(m - n, n)$. By $n \leq m$, we know

$$(1) \quad g|(m - n) \wedge g|n \wedge (\forall h : (h|(m - n) \wedge h|n) \Rightarrow h \leq g).$$

(We define an abbreviation g for $\text{gcd}(m - n, n)$. Since $n \leq m$, we have $m - n \in \mathbb{N}$. Since $m \neq 0 \vee n \neq 0$, also $m - n \neq 0 \vee n \neq 0$ (which has to be shown in a small subproof). By the knowledge added above, we thus have an object that satisfies the formula in the definition of $\text{gcd}(m - n, n)$. Therefore we may apply the corresponding rule for “such terms” and assume that g satisfies this formula as well.)

We have to show $g = \text{gcd}(m, n)$, i.e., by the explanation above

$$(2) \ g|m \wedge g|n \wedge (\forall h : (h|m \wedge h|n) \Rightarrow h \leq g).$$

(We apply the reasoning step explained above. Now we have to prove the conjunction by showing each conjunct in turn.)

1. We show (3) $g|m$. Because of (1), we have some a and b with $ga = m - n$ and $gb = n$. Then we know

$$g(a + b) = ga + gb = (m - n) + n = m;$$

which implies (3).

(We insert the definition of $|$ into knowledge (1), which gives us an existential formula $(\exists a : ga = m)$ and $(\exists b : gb = n)$. The application of these existential formulas gives us new constants a and b with the corresponding properties. Then we apply equational reasoning which gives us our goal by the transitivity of $=$.)

2. We know (4) $g|n$ from (1).

(The goal is part of our knowledge.)

3. We show (5) $\forall h : (h|m \wedge h|n) \Rightarrow h \leq g$. Take arbitrary h and assume

$$(6) \ h|m \wedge h|n.$$

We show (7) $h \leq g$.

(We have to show a universally quantified implication. We introduce an arbitrary constant h and assume the implication antecedent. We now have to show the new goal (7).)

By (6) we have some a' and b' with $ha' = m$ and $hb' = n$.

(We insert the definition of $|$ into knowledge (6) which gives two existential formulas that we can apply to introduce the constants a' and b' with the knowledge denoted above.)

Then we know

$$h(a' - b') = ha' - hb' = m - n$$

and thus (8) $h|(m - n)$.

(We apply equational reasoning to yield a formula in our knowledge that is equivalent to the definition of $|$ for arguments h and $m - n$.)

With (6) and (8), (1) implies (7).

(We apply the rule of “universal quantification in knowledge” to instantiate the universally quantified implication in (1) by term h . Then we apply the rule of “modus ponens” to this implication and to the conjunction of (6) and (8), which gives us the goal.)

While above proof is very detailed, it still relies on the claim that the greatest common divisor is uniquely defined (which has to be shown in an extra proof). The verbosity of the proof stems from the fact that it considers details like “Is $m - n$ actually defined?” or “Is the gcd unique?” that were neglected in the high-level proof skeleton.

Whether one is willing to bother with details is a matter of whether one is actually interested in the truth of a formula *as it stands* or whether one just wants to know whether the “intention is correct” (whatever this means). For a computer scientist or engineer, exactness *is* important (a “roughly” correct program may be the cause of an airplane crash); she should not be *anxious* but *eager* to find an error in a proposition and thus also check all the details that are often neglected in typical mathematical texts.

Chapter 10

Further Proving Techniques

Popular proving techniques not covered in this text are:

Proof by example: The author gives only the case $n = 2$ and suggests that it contains most of the ideas of the general proof.

Proof by intimidation: "Trivial."

Proof by vigorous handwaving: Works well in a classroom setting.

Proof by cumbersome notation: Best done with access to at least four alphabets and special symbols.

Proof by exhaustion: An issue or two of a journal devoted to your proof is useful.

Proof by omission: "The reader may easily supply the details".
"The other 253 cases are analogous."
"..."

Proof by obfuscation: A long plotless sequence of true and/or meaningless syntactically related statements.

Proof by wishful citation: The author cites the negation, converse, or generalization of a theorem from the literature to support his claims.

Proof by funding: How could three different government agencies be wrong?

Proof by eminent authority: "I saw Karp in the elevator and he said it was probably NP-complete."

Proof by personal communication: “Eight-dimensional colored cycle stripping is NP-complete [Karp, personal communication].”

Proof by reduction to the wrong problem: “To see that infinite-dimensional colored cycle stripping is decidable, we reduce it to the halting problem.”

Proof by reference to inaccessible literature: The author cites a simple corollary of a theorem to be found in a privately circulated memoir of the Slovenian Philological Society, 1883.

Proof by importance: A large body of useful consequences all follow from the proposition in question.

Proof by accumulated evidence: Long and diligent search has not revealed a counterexample.

Proof by cosmology: The negation of the proposition is unimaginable or meaningless. Popular for proofs of the existence of God.

Proof by mutual reference: In reference A, Theorem 5 is said to follow from Theorem 3 in reference B, which is shown to follow from Corollary 6.2 in reference C, which is an easy consequence of Theorem 5 in reference A.

Proof by metaproof: A method is given to construct the desired proof. The correctness of the method is proved by any of these techniques.

Proof by picture: A more convincing form of proof by example. Combines well with proof by omission.

Proof by vehement assertion: It is useful to have some kind of authority relation to the audience.

Proof by ghost reference: Nothing even remotely resembling the cited theorem appears in the reference given.

Proof by forward reference: Reference is usually to a forthcoming paper of the author, which is often not as forthcoming as at first.

Proof by semantic shift: Some of the standard but inconvenient definitions are changed for the statement of the result.

Proof by appeal to intuition: Cloud-shaped drawings frequently help here.

Appendix A

Logic Evaluator Definitions

A.1 Natural Numbers

```
// -----  
// $Id: evaluator.tex,v 1.2 2003/02/05 16:27:35 schreine Exp $  
// the natural numbers  
//  
// (c) 1999, Wolfgang Schreiner, see file COPYRIGHT  
// http://www.risc.uni-linz.ac.at/software/formal  
// -----  
  
// we use the builtin type  
pred N(x) <=> Nat(x);  
  
// the constructors  
fun NO = 0;  
fun '(x: N) = +(x, 1);  
  
// predecessor  
fun ^-(x: N) = such(n in nat(0, x): =(x, '(n)), n);  
  
// constants  
fun N1 = '(NO);  
fun N2 = '(N1);  
  
// addition, multiplication, comparison  
fun +N(x: N, y: N) recursive y =  
  if(=(y, NO), x, '(+N(x, ^-(y))));  
fun *N(x: N, y: N) recursive y =  
  if(=(y, NO), NO, +N(x, *N(x, ^-(y))));  
pred <=N(x: N, y: N) recursive y <=>  
  if(=(x, NO), true, if(=(y, NO), false, <=N(^-(x), ^-(y))));
```

```

// difference
fun -N(x: N, y: N) = such(z in nat(0, x): =(x, +N(z, y)), z);

// quotient and remainder
fun divN(x: N, y: N) =
  such(q in nat(0, x), r in nat(0, ^-(y)):
    =(x, +N(*N(q, y), r)), q);
fun modN(x: N, y: N) =
  such(q in nat(0, x), r in nat(0, ^-(y)):
    =(x, +N(*N(q, y), r)), r);

// exponentiation
fun ^N(x: N, n: N) recursive n =
  if(=(n, N0), N1, *N(x, ^N(x, ^-(n))));

// more notions
pred divides(x, y) <=> exists(z in nat(N0, y): =( *N(x, z), y));
fun gcd(x, y) =
  let(m = if(=(x, N0), y, x):
    such(z in nat(N0, m):
      and(divides(z, x), divides(z, y),
        forall(w in nat(+N(z, N1), m):
          or(not(divides(w, x)), not(divides(w, y))))),
      z));
fun lcm(x, y) = such(z in nat(N1, *N(x, y)):
  and(divides(x, z), divides(y, z),
    forall(w in nat(x, -(z, N1)):
      or(not(divides(x, w)), not(divides(y, w))))),
  z);
pred relprime(x, y) <=> =(gcd(x, y), N1);
pred isprime(x) <=>
  and(not(<=N(x, N1)),
    forall(y in nat(N0, x):
      implies(divides(y, x), or(=(y, N1), =(y, x))));

// -----
// $Id: evaluator.tex,v 1.2 2003/02/05 16:27:35 schreine Exp $
// -----

```

Index

- ackerman's function, 41
- additional knowledge, 102
- backus naur form, 53
- base formula, 42
- base term, 40
- binomial coefficient, 34
- binomial identities, 35
- bnf, 53
- complete induction, 48
- conditional formula, 12
- constructors, 51
- continuous, 17
- contradiction, 66
- contraposition, 105
- de morgan, 105
- decomposition of conjunctions, 87
- decomposition of disjunctions, 88
- decomposition of equivalences, 82
- decomposition of existential formulas, 78
- decomposition of implications, 84
- decomposition of universal formulas, 74
- difference, 24
- direct proof, 65
- discrete, 17
- divides, 26
- existential formula in knowledge, 100
- explicitly defined functions, 91
- explicitly defined predicates, 90
- exponentiation, 25
- factorial, 34
- factorial function, 6
- fibonacci numbers, 41
- grammars, 53
- greatest common divisor, 26
- indirect proof, 66
- induction axioms, 18
- induction base, 44
- induction hypothesis, 44
- induction step, 44
- inductive definition, 40
- inductive function definition, 40
- inductive predicate definition, 42
- inductive set definition, 50
- least common multiple, 26
- lexicographic ordering, 42
- mathematical induction, 43
- matrices over the reals, 36
- matrix operations, 36
- minimum and maximum functions, 28
- minimum and maximum quantifier, 27
- modus ponens, 105
- natural number laws, 22
- natural number operations, 26
- natural number subsets, 20
- natural numbers, 19
- natural numbers from set, 18
- natural numbers operations, 20, 21

- null matrix, 36
- number of permutations, 47
- order predicates, 22
- pascal's triangle, 35
- peano arithmetic, 17
- predecessor, 20
- prime, 26
- prime number factorization, 33
- product quantifier, 32, 33
- proof, 58
- proof by case distinction, 95
- proof by contradiction, 66
- proof completion, 64
- proof rule, 62
- proposition, 58
- propositional consequence, 104
- propositional consequences, 104
- propositional tautology, 105

- quantifier consequences, 106
- quotient, 24
- quotient and remainder, 24

- real matrices, 36
- recursion formula, 42
- recursion term, 40
- recursive function definition, 8
- recursive predicate definition, 12
- reduction, 14
- regularity, 19
- relatively prime, 26
- remainder, 24

- set reduction, 14
- size of powerset, 69
- structural induction, 54
- substitutions, 108
- successor, 17
- such quantifier, 107
- sum quantifier, 29, 31

- tautology, 105
- term algebra, 53
- termination function, 9
- termination term, 41
- theorem, 58

- unity matrix, 36
- universal formula in knowledge, 98

- well-founded ordering, 7
- witness, 78

- zero, 17