Chapter 1:

Syntax and Informal Semantics of Predicate Logic

1.1 Why Syntax?

Mathematics, in particular a mathematical proof, always reflects processes in the human mind. We consider something as true, *therefore* something else must be true. Let's consider the following statements:

A: Increasing demand increases the price.

B: Raising the taxes increases the price.

Consider A and B to be true and imagine this is all what we know about "demand", "price", and "tax". (Either A and B had been proven before or we consider them as axioms—as "laws of the market"—this does not make a difference in what we want to investigate now!)

Question 1: We observe increasing demand. Can we conclude that the price must increase?
Answer: YES. ("Because of" A)

Question 2: We observe increasing price. Can we conclude that the demand has increased?
Answer: NO. (It could also be due to raising the taxes, i.e. "because of" B.)

Question 3: We observe increasing price. Can we conclude that the demand has not increased?
Answer: NO. (Of course not, the increase in price can be just the effect of increasing demand.)

Question 4: We observe decreasing price. Can we conclude that the demand has not increased?

Answer: YES. (If it had increased then, "because of" A, the price would increase.)

Question 5: We observe decreasing price. Can we conclude that the taxes have not been raised?
Answer: YES. (If taxes had been raised then, "because of" B, the price would increase.)

The justifications for the above answers can be given without any understanding of the concepts "demand", "price", and "tax". This means, the logical rules in our brains that make us answer "YES" or "NO", respectively, only depend on the syntactical structure of the statements under consideration. In fact, there is a tight correspondence between "thoughts" and the way we express them. A clear understanding of the syntactical structure helps to understand lots of the "reasoning power" contained in a statement. On the other hand, the correctness of logical reasoning can be checked merely on a syntactical basis without understanding the meaning of the statement.

> We should, thus, learn to concentrate on *syntax rather than semantics*—on *form rather than content*—not only when analyzing the correctness of reasoning steps but also when doing one's own proofs!

This does neither dispel "having good ideas" nor "creativity" nor "intuition" from mathematical proving. In some proofs it will be valuable to have some intuition in order to find the right track, on which to proceed with the proof. Ideas and intuition are helpful for finding the appropriate proof rule to apply next, but the actual application of the rule is safer when being done on a syntactical basis. Often mathematical concepts become too complicated or too abstract for reasoning about their properties just based on intuition, and often the intuitive picture of some mathematical concept is only a rough approximation to the real meaning and can lead into error.

Exercise 1

Question 4 is often answered "NO" with the following argument:

> The demand could have increased. If the taxes were lowered sufficiently, then one would still observe desreasing price, therefore one *cannot conclude* that demand has *necessarily* not increased.

Analyse this argument and compare it to the argument given as justification for "YES".

Solution →

In the rest of Chapter 1, we will study the syntactical structure of statements—such as A and B—in predicate logic. Based on this structure we will then investigate in Chapter 2, how certain statements can be used in order to derive other statements, which we then know to be true as long as the original statements were true. These rules will be called "inference rules" and they are just what is hidden behind the "because of" in the above example. In mathematical proofs, it is common to just phrase the application of inference rules as "therefore", "because of", or often just "⇒", and it is assumed that the reader is familiar with the rules that justify the respective proof step.

1.2 What Is Syntax?

Syntax describes the *form* of expressions in a language. We will describe predicate logic by its *abstract syntax*, i.e. one particular standard form of expressions in the language. Expressions that conform to the rules of syntax are called "well-formed expressions". Compare this to programming languages: The syntax tells, how programs must be written, where blanks are allowed, where semi-colons must be written, whether to use brackets or parentheses, etc.), a well-formed expression in a programming language is a program that passed the compiler without "syntax error". There are, however, many different "concrete ways" of writing the concepts described in the abstract syntax of predicate logic, mainly to allow different appearances of mathematical expressions reflecting different taste and style as it developed over the years. We call this *concrete syntax*, *external syntax*, or *notation*. (I don't know a comparable concept in programming languages.)

However, whatever the concrete notation is, the syntactical structure of an expression in predicate logic (and other langauges) is defined by the answers to three questions:

- To which class of expression does the expression belong? In concrete notation, this question is answered by determining the "outermost symbol" or the "classifier" of the expression.
- What are the subexpressions of the expression? This question is answered by "selector" functions that decompose ("parse") the expression.
- How can we form new expressions in a certain class from given expressions? This question is answered by "constructor" functions that compose new expressions.

The abstract syntax of a language is defined by the classifiers, the constructors and the selectors and their interaction.

Example

$$\underset{\substack{x \\ x^2>a}}{\forall} \left(x > b \bigwedge \left|\frac{1}{x}\right| < \epsilon\right)$$

The outermost symbol of this expression is the symbol '$\forall$' (universal quantifier).

The ingredient subexpressions are:

the quantified variable 'x',

the condition expression '$x^2 > a$', and

the body expression '$(x > b \bigwedge |\frac{1}{x}| < \epsilon)$'.

□

The syntactical structure of an expression can be determined without knowing anything about the meaning of the symbols. We must be able to recognize the concepts of predicate logic in all possible notational variants. The most extreme case being the recognition of predicate logic concepts in natural language, which is very important, since much of mathematics conversation is done in natural language. Don't believe mathematics starts where one starts to introduce fancy symbols and complicated formulae! The really distinctive feature of mathematics is not the substitution of natural language by formulae but it is its formal rigor.

Moreover, it should be seen, that the validity of the rules of mathematical logic is not restricted to mathematics. The rules of logic have developed over centuries and they only reflect what humans have observed in their every-day behavior. Therefore, it can also be of help in daily conversation to recognize the logic structure in statements such as "There is always …" or "If we … then … we will …" and to know what conclusions are allowed from these. It can often be observed, that wrong conclusions are drawn by wrong application of logical rules.

1.3 What is Semantics?

Semantics refers to the *meaning* of expressions in a language. The semantics of an expression is based on its syntax, and also the actions that can be attached to an expression—such as proving, computing, and solving—are guided by the syntactical structure of expressions. Hence, a clear understanding of the syntax is the basis, and often the key, for understanding mathematics.

Example

$$\forall_{\substack{x \\ x^2>a}} \left(x > b \bigwedge \left|\frac{1}{x}\right| < \epsilon\right)$$

has the meaning that

for all values 'x',

satisfying the condition '$x^2 > a$'

'$(x > b \bigwedge |\frac{1}{x}| < \epsilon)$' holds.

□

1.4 A Standard Syntax: *Theorema*

Theorema is a software system implemented by the *Theorema* group at RISC-Linz under the direction of B. Buchberger, see www.theorema.org. *Theorema* contains a computer-supported version of predicate logic and other useful language constructs, such as numbers, sets, and tuples. The external syntax, i.e. the appearance of the language, is as close as possible to the syntax commonly used in mathematical textbooks. The internal representation of expression in the *Theorema* system, on the other hand, is very close to the abstract syntax of predicate logic. For getting a quick and deep introduction to predicate logic syntax, we will display predicate logic expression in the internal format of *Theorema*. The internal syntax will uniquely reveal the complete syntax of any expression in predicate logic.

- This loads *Theorema* during a Mathematica session. A valid license for *Theorema* is necessary.

```
Needs["Theorema‘"]
```

- To obtain the internal format of the above predicate logic expression, enter

```
•[ ∀_{x, x^2>a} (x > b ⋀ |1/x| < ϵ) ] // InputForm
```

```
™ForAll[•range[•simpleRange[•var[x]]], ™Greater[™Power[•var[x], 2], a],
 ™And[™Greater[•var[x], b], ™Less[™BracketingBar[™Divide[1, •var[x]]], ϵ]]]
```

- (Wrapping an expression into •[…] tells Mathematica to parse the expression as a *Theorema* expression, the suffix //InputForm shows the internal representation instead of the standard external form.)

The internal representation shows the syntactical structure of the expression in a systematic way, namely as a *nested expression* made up from a *head symbol* (outermost symbol, leftmost symbol) and *subexpressions*. The head symbol determines the "type of the expression", i.e. the language category, to which the expression belongs. In the example above, the head symbol '™ForAll' indicates that the expression is a "universally quantified formula" containing the three subexpressions

•range[•simpleRange[•var[x]]]

™Greater[™Power[•var[x], 2], a]

™And[™Greater[•var[x],b],™Less[™BracketingBar[™Divide[1,•var[x]]],ϵ]]

The application of head symbols to subexpressions is written using brackets ('[', ']'). Each of the subexpressions again has a head symbol and subexpressions. Some expressions do not contain subexpressions anymore, we call these expressions *atomic expressions*. The actual names used as head symbols ('™ForAll', '™Greater', •range, •var, etc.) do not matter, you can choose other—more telling—names. Even symbols may be used, for instance the third subexpression above could also be written as

∧ [> [•var[x], b], < [|| [÷ [1, •var[x]]], ϵ]]

It is important to see, how complex expressions are built-up from simpler expressions. Again there are very similar concepts in programming languages: The first step during the compilation of a program, i.e. the transformation to the machine code of the particular computer, is the "lexical analysis", where the stream of symbols is divided into "tokens", i.e. separate pieces, words. During lexical analysis it is decided, for instance, whether "$x > b$" consists of the three tokens 'x', '$>$', and 'b' or just one token '$x > b$'. This may seem trivial, but already the next example will show that lexical analysis may become tricky: the input stream "$3\,n$" is usually divided into the two tokens '3' and 'n', whereas "in" is usually recognized as only one token 'in'.

The hierarchical inter-relation of language chunks left after lexical analysis is established by the "parsing" of the expression, i.e. the grammatical analysis of the expression. During parsing, the sequence of tokens 'x', '$>$', and 'b' is recognized as "the operation '$>$' applied to the arguments 'x' and 'b'". This can be established since the token '$>$' is recognized as a symbol, which may occur *between* its subexpressions. It stands for the known head symbol ™Greater, thus "$x > b$" is parsed as $\text{™Greater}[x, b]$. The result of parsing is best visualized in the so-called "parse tree" of the expression.

- In Mathematica, the parse tree of an expression can be displayed using the command TreeForm.

```
•[ ∀_{x, x^2>a} (x > b ⋀ |1/x| < ϵ) ] // TreeForm

™ForAll[ | , | , |
    •range[ | ]  ™Greater[ | , a]  ™An
      •simpleRange[ | ]  ™Power[ | , 2]
        •var[x]  •var[x]
```

From this it becomes evident that the most appropriate data structure for storing expressions is a *tree structure*, where each node contains the head of the corresponding subexpression and each subtree of a node corresponds to a subexpression. The root of the tree would contain the outermost symbol of the whole expression, the leaves of the tree would contain only atomic expressions.

- The use of symbols starting with ™ as head symbols in *Theorema* has technical reasons. The same is true for head symbols starting with •, such as •var or •range.

1.5 The Standard Language Constructs of Predicate Logic

We will now introduce the standard language constructs available in predicate logic. For most language constructs, we will give the standard syntax as supported in *Theorema* and several notational variants that are commonly used in mathematics.

1.5.1 Example from Analysis 1: The Definition of 'converges'

We want to define—for all sequences f and all real numbers a—what it means that f converges to a. In *Theorema*'s version of predicate logic, this would be written as

```
•[ ∀_f ∀_a ( converges[f, a] :⇔ ∀_{ϵ∈ℝ⋀ϵ>0} ∃_{n∈ℕ} ∀_{m∈ℕ⋀m>n} |f_m − a| < ϵ ) ] // InputForm
```

```
™ForAll[
  •range[•simpleRange[•var[f]]],
  True,
  ™ForAll[
    •range[•simpleRange[•var[a]]],
    True,
    ™Iff[
      converges[•var[f], •var[a]],
      ™ForAll[
        •range[•simpleRange[•var[ϵ]]],
        ™And[™Element[•var[ϵ], ™ℝ], ™Greater[•var[ϵ], 0]],
        ™Exists[
          •range[•simpleRange[•var[n]]],
          ™Element[•var[n], ™ℕ],
          ™ForAll[
            •range[•simpleRange[•var[m]]],
            ™And[™Element[•var[m], ™ℕ], ™Greater[•var[m], •var[n]]],
            ™Less[
              ™BracketingBar[
                ™Minus[
                  ™Subscript[•var[f], •var[m]],
                  •var[a]]],
              •var[ϵ]]]]]]]]
```

This is a definition of a *new property*, namely that f converges to a. In the internal syntax, this is reflected by introducing a *new head symbol* ‘converges’, which is applied to its argument expressions. A definition only introduces an *abbreviation* for usually more complex expressions of predicate logic, i.e. it is not necessary to view definitions as a core component of predicate logic, it is just that it is *convenient* to allow the introduction of new symbols that are defined in terms of known symbols. We will learn more about the rules, how to give explicit definitions, in Section 1.8 later. In general, the structure of a definition is

$$\text{new}[v_1, \ldots, v_n] \;:\Leftrightarrow\; \text{expression}_{v_1,\ldots,v_n}$$

or

$$\text{new}[v_1, \ldots, v_n] \;:=\; \text{expression}_{v_1,\ldots,v_n}$$

and it allows to substitute each occurrence of ‘$\text{new}[v_1, \ldots, v_n]$’ by ‘$\text{expression}_{v_1,\ldots,v_n}$’, where ‘$\text{expression}_{v_1,\ldots,v_n}$’ is a predicate logic expression involving the variables $v_1, \ldots, v_n$. The expression to the right of $:\Leftrightarrow$ in the definition above contains most of the important categories of language constructs of predicate logic.

- The “wrappers” starting with • in the internal form of the expression are used for clarification of syntactical structure in the *Theorema* system. They are not part of predicate logic. For instance, •var is used to explicitely mark symbols to denote

variables in order to be able to distinguish them from symbols denoting constants, see below.

For an intuitive understanding of predicate logic it is helpful to distinguish expressions into *terms* and *formulae*. A term describes an object, whereas a formula describes a statement about (a property of) objects.

1.5.2 Terms

1.5.2.1 Constants

- Object constants: stand for concrete objects in the "universe of discourse".

  In the example above: 0, ™ℝ, ™ℕ .

- Function constants: stand for concrete functions (operations, processes, algorithms, etc.) on objects in the "universe of discourse". The "arity" of a function constant is the number of arguments that it can take.

  In the example above: ™BracketingBar (arity 1), ™Minus (arity 2), ™Subscript (arity 2).

- Predicate constants: stand for concrete predicates (attributes, relations, properties, etc.) on objects in the "universe of discourse". The "arity" of a predicate constant is the number of arguments that it can take.

  In the example above: converges (arity 2), ™Element (arity 2), ™Greater (arity 2), ™Less (arity 2).

1.5.2.2 Variables

- Object (Ordinary) variables: place-holders, for which terms can be substituted, see Section 1.6, or which can be quantified, see Section 1.5.3.3.

  In the example above: f, a, ϵ, n, m. (In *Theorema*'s internal form, they are represented as •var[f], •var[a], •var[ϵ], etc.)

- Function variables:

  In the example above: None.

- Predicate variables:

  In the example above: None.

These types of variables are available in *first-order logic*. Note, that function and predicate variables are available, but may neither be substituted nor quantified in first-order logic. In higher-order logic, function and predicate variables may

be both substituted and quantified. We will not embark on the advantages and disadvantages of first-order versus higher-order. The approach chosen mostly in mathematics is to use first-order logic and introduce *set theory* on top of first-order predicate logic. Both functions and relations can be represented *within* set theory, which makes them first-order objects, thus making function and predicate variables superfluous. These topics will be discussed in all necessary detail in courses on logic (e.g. Logik 1, Logik 2).

1.5.2.3 Compound Terms

Terms are built up inductively from object and function constants and variables by "application". Application, in *Theorema*, is denoted by brackets '[' and ']'. (In almost all other notations, application is denoted by parantheses '(' and ')'. The bracket notation, which is borrowed from Mathematica, avoids certatin ambiguities that may arise with the parantheses notation.)

- Constants and ordinary variables are terms.
- If $t_1, \ldots, t_n$ are terms and f is an n-ary function constant or variable then $f[t_1, \ldots, t_n]$ is again a term.

  In the example above:

  ™Subscript[•var[f], •var[m]], i.e. f_m

  ™Minus[™Subscript[•var[f], •var[m]], •var[a]], i.e. $f_m - a$

  ™BracketingBar[™Minus[™Subscript[•var[f], •var[m]], •var[a]]], i.e. $|f_m - a|$

Usually, the arity of a function constant is fixed, i.e. in the same context a function constant must always take the same number of arguments. There are cases, where it is convenient to allow *flexible arity* constants, i.e. function constants that can take arbitrarily many arguments. In most of the cases, however, these are only abbreviations for more complicated expressions using fixed arity constants.

```
•[ a + b ] // InputForm
```

```
™Plus[a, b]
```

```
•[ a + b + c ] // InputForm
```

```
™Plus[a, b, c]
```

Alternatively, ™Plus[a, b, c] could be interpreted as an abbreviation for ™Plus[a, ™Plus[b, c]].

Notation: The standard way of denoting function application is $f[t_1, \ldots, t_n]$, where f is a function constant, i.e. the *name* of the function, and $t_1, \ldots, t_n$ are

the argument terms. Many functions in every-day mathematics have *associated operators*, i.e. symbols and symbolics, for abbreviating the names. In many cases, symbols and symbolics is chosen such that an intuitive meaning of the function is captured in the notation. Operators can be written in many different ways, notably *infix*, *prefix*, *postfix*, *matchfix*, *sub-*, *super-*, *under-*, *over-script*, and many other fancy two-dimensional patterns. For correctly parsing an expression it is absolutely necessary to know the behavior of operands in order to uniquely translate any fancy notation into its underlying standard form of predicate logic. "Good" notation is an absolutely crucial feature of mathematics and it is heavily used in all areas of mathematics, therefore it is inevitable to understand the logic structure of mathematical notation in all its variations. Table 1.1 shows commonly used expressions in mathematics with one possible translation to standard syntax:

Expression	**Operator**	**Type**	**Standard syntax**
$a + b$	$+$	Infix	™Plus[a, b]
$a - b$	$-$	Infix	™Minus[a, b]
$-a$	$-$	Prefix	™Minus[a]
$a!$	$!$	Postfix	™Factorial[a]
$\lvert a \rvert$	$\lvert\ \rvert$	Matchfix	™BracketingBar[a]
$\lVert a \rVert$	$\lVert\ \rVert$	Matchfix	™DoubleBracketingBar[a]
a_n	$\square_\square$	Subscript	™Subscript[a, n]
a_n	$\square_\square$	Subscript	a[n]
a^2	$\square^2$	Superscript	™Square[a]
a^n	$\square^\square$	Superscript	™Power[a, n]
$\sqrt{a}$	$\sqrt{\ }$	2 D	™Sqrt[a]
f'	$'$	Postfix	™Derivative[f]
$f^{(n)}$	$\square^{(\square)}$	2 D	™Derivative[f, n]
$\frac{a}{b}$	$-$	2 D / Infix	™Divide[a, b]
$\binom{a}{b}$	$\binom{\square}{\square}$	2 D / Matchfix	™Binomial[a, b]
$\begin{pmatrix} a \\ b \end{pmatrix}$	$\begin{pmatrix} \square \\ \square \end{pmatrix}$	2 D / Matchfix	™Vector[a, b]
$F\ \rvert_a^b$	$\rvert$	2 D/ Postfix	™EvalUpperLower[F, a, b]

Table 1.1:Frequently used operators associated with function symbols.

Ambiguities: As one can already see from Table 1.1, mathematical notation is not unique. In most of the cases, interpretation is at least unique in a particular context assumed to be known by the reader: in a book on combinatorics $\binom{a}{b}$ will certainly denote the binomial coefficient, in a book on geometry probably the vector. Lecture notes Linear Algebra: Page 12: used as binomial coefficient, page 58 and 59 used as vector.
Lecture notes Analysis: Page 236: in Exercise 7 used as binomial coefficient, on page 244 used as vector.
For humans, it is usually ‘‘clear from the context’’ what certain ambiguous notations actually mean.

Precedence: The question of precedence arises as soon as there is more than one operator, e.g.

$a + b!$.?. ™Factorial[™Plus[a, b]]

$a + b!$.?. ™Plus[a, ™Factorial[b]]

Parentheses ‘(’ and ‘)’ are used in order to indicate which operator to use first, i.e.

$(a + b)!$ … ™Factorial[™Plus[a, b]]

$a + (b!)$ … ™Plus[a, ™Factorial[b]]

Note that in the standard syntax there is no need for parentheses! In order to avoid parentheses in certain cases one introduces *operator precedence*. In the example above, interpretation 1 would be achieved without need of parentheses by assigning a higher precedence for ‘+’ than to ‘!’.

General rule: If unsure, use parentheses!

1.5.3 Formulae

Formulae can be atomic formulae, propositional formulae, or quantifier formulae.

1.5.3.1 Atomic Formulae

Atomic formulae are built up inductively from predicate constants and variables by ‘‘application’’ using brackets ‘[’ and ‘]’.

- If $t_1, \ldots, t_n$ are terms and p is an n-ary predicate constant or variable then $p[t_1, \ldots, t_n]$ is an atomic formula.

In the example above:

TM Element[•var[ϵ], TM ℝ], TM Element[•var[n], TM ℕ], TM Element[•var[m], TM ℕ], i.e. $\epsilon \in \mathbb{R}$, $n \in \mathbb{N}$, $m \in \mathbb{N}$

TM Greater[•var[ϵ], 0], TM Greater[•var[m],•var[n]], i.e. $\epsilon > 0$, $m > n$

TM Less[TM BracketingBar[TM Minus[TM Subscript[•var[f], •var[m]], •var[a]]], •var[ϵ]], i.e. $| f_m - a | < \epsilon$

Like with function constants, we allow flexible arity also for some predicate constants.

```
•[  a < b  ] // InputForm
```

```
TM Less[a, b]
```

```
•[  0 < n < M  ] // InputForm
```

```
TM Less[0, n, M]
```

Alternatively, TM Less[0, n, M] could be interpreted as abbreviation for TM Less[0, n] $\wedge$ TM Less[n, M].

Notation: Predicate symbols can be denoted—like function symbols—using associated operators. Table 1.2 shows commonly used expressions in mathematics with one possible translation to standard syntax:

Expression	Operator	Type	Standard syntax
$a = b$	$=$	Infix	™Equal[a, b]
$a \leq b$	$\leq$	Infix	™LessEqual[a, b]
$a \geq b$	$\geq$	Infix	™GreaterEqual[a, b]
$a \mid b$	$\mid$	Infix	™VerticalBar[a, b]
$a \equiv_n b$	$\equiv_n$	2 D / Infix	™Congruent[a, b, n]
$a \in A$	$\in$	Infix	™Element[a, A]
$A \subseteq B$	$\subseteq$	Infix	™SubsetEqual[A, B]
$a \sim b$	$\sim$	Infix	™Tilde[a, b]
$f \longrightarrow a$	$\longrightarrow$	Infix	™LongRightArrow[f, a]
$g \perp h$	$\perp$	Infix	™UpTee[g, h]
$g \parallel h$	$\parallel$	Infix	™DoubleVerticalBar[g, h]
$f : A \to B$	$: \to$	Infix	™IsFunctionFromTo[f, A, B]
$f \nearrow$	$\nearrow$	Postfix	™UpperRightArrow[f]

Table 1.2: Frequently used operators associated with predicate symbols.

Ambiguities and Precedence: see terms.

1.5.3.2 Propositional Formulae

Propositional formulae are built up inductively from formulae by the propositional connectives.

- If A and B are formulae then "negations", "disjunctions", "conjunctions", "implications", and "equivalences" are propositional formulae.

Connective	Syntax	Name	speak
$\neg$	$\neg A$	Negation	"not" A
$\vee$	$A \vee B$	Disjunction	A "or" B
$\wedge$	$A \wedge B$	Conjunction	A "and" B
$\Rightarrow$	$A \Rightarrow B$	Implication	A "implies" B
$\Leftrightarrow$	$A \Leftrightarrow B$	Equivalence	A "if and only if" B

Table 1.3: Propositional connectives.

In the example above:

TMAnd[TMElement[•var[ϵ], TM$\mathbb{R}$], TMGreater[•var[ϵ], 0]], i.e. $\epsilon \in \mathbb{R} \wedge \epsilon > 0$

TMAnd[TMElement[•var[m], TM$\mathbb{N}$], TMGreater[•var[m], •var[n]]], i.e. $m \in \mathbb{N} \wedge m > n$

TMIff[converges[•var[f], •var[a]], TMForAll[…]], i.e. converges$[f, a] \Leftrightarrow \forall \ldots$

In written text, "if and only if" is often abbreviated "iff".
Connectives can also be hidden in quantifier expressions, see Table 1.5 below, or in flexible arity operators, see above.

1.5.3.3 Quantifier Formulae

Quantifier formulae are built up inductively from formulae by the quantifiers $\forall$ and $\exists$. In correspondence with the outermost quantifiers, such formulae are called "universal formulae" and "existential formulae", respectively. The distinctive feature of quantifiers is that quantifiers "bind" a variable, see Section 1.6.1.

- If A and C are formulae and x is an object variable then $\underset{x}{\underset{C}{\forall}} A$, $\underset{x}{\underset{C}{\exists}} A$ are quantified formulae.

  In the example above:

  TMForAll[
  •range[•simpleRange[•var[m]]],
  TMAnd[TMElement[•var[m], TM$\mathbb{N}$],TMGreater[•var[m], •var[n]]],
  TMLess[TMBracketingBar[TMMinus[TMSubscript[•var[f], •var[m]], •var[a]]], •var[ϵ]]]
  …
  In external form:

$$\underset{\substack{m \\ m\in\mathbb{N}\wedge m>n}}{\forall} |f_m - a| < \epsilon$$

$$\underset{n\in\mathbb{N}}{\exists} \underset{\substack{m \\ m\in\mathbb{N}\wedge m>n}}{\forall} |f_m - a| < \epsilon$$

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall} \underset{n\in\mathbb{N}}{\exists} \underset{\substack{m \\ m\in\mathbb{N}\wedge m>n}}{\forall} |f_m - a| < \epsilon$$

$$\underset{a}{\forall} \left(\text{converges}[f, a] \Leftrightarrow \underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall} \underset{n\in\mathbb{N}}{\exists} \underset{\substack{m \\ m\in\mathbb{N}\wedge m>n}}{\forall} |f_m - a| < \epsilon \right)$$

$$\forall_f \forall_a \left(\text{converges}[f, a] \Leftrightarrow \forall_{\epsilon \in \mathbb{R} \wedge \epsilon > 0} \exists_{n \in \mathbb{N}} \forall_{m \in \mathbb{N} \wedge m > n} |f_m - a| < \epsilon \right)$$

The internal representation reveals the three selectors, that every quantifier formulae has:

- the quantified variable: •range[•simpleRange[•var[m]]]
- the condition (on the variable): TMAnd[TMElement[•var[m], TMℕ], TMGreater[•var[m], •var[n]]]
- the body formula: TMLess[TMBracketingBar[TMMinus[TMSubscript[•var[f], •var[m]], •var[a]]], •var[ϵ]]]

In *Theorema*'s external syntax, the quantified variable is always written *under* the quantifier, the condition is written under both quantifier and quantified variable. In other mathematical texts, the quantified variable might be written as subscript to the quantifier or just to the right of the quantifier. The above example would then be written as

$$\forall\, (\epsilon \in \mathbb{R} \wedge \epsilon > 0)\ \ \exists\, n \in \mathbb{N}\ \ \forall\, (m \in \mathbb{N} \wedge m > n) :\ |f_m - a| < \epsilon\ .$$

Special forms

- The quantified variable and the condition are often combined into "special ranges". Let Q stand for $\forall$ or $\exists$:

Long Form	Special Range
$\underset{x}{Q}\ A$ $x \in S$	$\underset{x \in S}{Q}\ A$
$\underset{x}{Q}\ \ A$ $x \in \mathbb{Z} \wedge l \le x \le u$	$\underset{x=l,\ldots,u}{Q}\ \ A$

Table 1.4: Special ranges.

Note, however, that the range specification must allow to determine the quantified variable! A human mathematician can often decide this again "in the context", when communicating with a machine this might sometimes need more care. It is, for instance, common to write

$$\ldots \underset{m>n}{\forall} |f_m - a| < \epsilon$$

and it is "obvious" that the variable quantified by the universal quantifier is 'm' and not 'n'.

- Subsequent quantifiers of the same type are often combined into one quantifier with "multiple range", i.e.

$$\underset{x}{\forall}\,\underset{y}{\forall}\,\underset{z}{\forall}\,A$$

can be written as

$$\underset{x,y,z}{\forall}\,A$$

- Conditions in a quantifier can, in fact, be viewed as abbreviations as shown in Table 1.5.

Quantifier with Condition	Long Form
$\underset{\substack{x\\C}}{\forall}\,A$	$\underset{x}{\forall}\,C \Rightarrow A$
$\underset{\substack{x\\C}}{\exists}\,A$	$\underset{x}{\exists}\,(C \wedge A)$

Table 1.5: Conditions in quantifiers.

- The condition in a quantifier can be omitted, in the internal form it is represented as 'True'.

All special forms of quantifiers reduce—by the rules above—to quantifier formulae without condition! We could even "survive" with only the universal quantifier and define $\underset{x}{\exists}\,A$ to stand for $\neg\,\underset{x}{\forall}\,\neg\,A$. Since we introduce both quantifiers, the "de Morgan laws"

$$\neg\,\underset{x}{\forall}\,A \Leftrightarrow \underset{x}{\exists}\,\neg\,A$$

$$\neg\,\underset{x}{\exists}\,A \Leftrightarrow \underset{x}{\forall}\,\neg\,A$$

would have to be *proven* based on the semantics for quantifier formulae given in Section 1.7.

If x is the quantified variable in a formula A then we call an occurrence of x a *bound occurrence* and we call x a *bound variable* in A. Otherwise, variables and their occurrences are called *free*. The distinction between free and bound occurrences of variables is of particular importance when it comes to the crucial operation on variables, namely *substitution*.

1.6 Substitution and Replacement

1.6.1 Free and Bound Occurences of Variables

Variables are the essence of mathematical language, which make the mathematical language really powerful. Variables are atomic (not decomposable) parts of expressions for which potentially infinitely many different expressions may be substituted. An expression containing a free variable has many meanings, one for each assignment of a concrete object as the meaning of the variable. Hence, an expression containing a free variable, in some sense, stands for potentially infinitely many expressions and hence, in one stroke, can express potentially infinitely many meanings depending on which expressions are substituted for the variables. Expressions containing free variables can be viewed as "general" statements that can be formulated once but can be unfolded in potentially infinitely many situations.

(A formula containing free variables is exactly what in the Analysis lecture notes is called "Aussageform" and in the Linear Algebra lecture notes is called "Prädikat" !)

Most of what has been remarked above about free variables is not true for bound variables, such as the x in the formula $\underset{x\in\mathbb{N}}{\forall}(x=0)$. This formula does not have various meanings, it means exactly the statement that all natural numbers are equal to 0, which is false (it should intuitively be clear that this is false; however, we can only say so after having discussed the *semantics* of $\forall$, which has not been done yet!). Substituting 0 or 7 for x in the formula $x=0$ makes it a true $(0=0)$ or false $(7=0)$ statement, respectively. Substituting 0 or 7 for x in the formula $\underset{x\in\mathbb{N}}{\forall}(x=0)$ makes it into the meaningless formulae $\underset{0\in\mathbb{N}}{\forall}(0=0)$ or $\underset{7\in\mathbb{N}}{\forall}(7=0)$, respectively. Therefore, bound variables should be considered "invisible" for the substitution operation.

Substituting the variable y for x in $\underset{x\in\mathbb{N}}{\forall}(x=0)$ does not change the meaning of the formula, whereas substituting y for x in $x=0$ might change the meaning. It depends on the assignments for x and y, whether $x=0$ and $y=0$ have the same meaning.

Note that there is no possibility to decide whether an occurrence of a symbol in a formula is a free occurrence of a variable or a constant without knowing the context of the formula. For example, just looking at

$$\underset{n\in\mathbb{N}}{\exists}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall}\ |f_m - a| < \epsilon$$

it is not possible to tell whether 'f', 'a', 'ϵ' are free occurrences of variables or (function or object) constants.

We will discuss this subtle point in more detail later. In fact, wrapping symbols by •var[…] in the internal *Theorema* representation is done in order to be able to distinguish the usage of symbols as free variables and as constants. Alternatively, one would have to declare, once and for all, which symbols will be used as variables and which symbols will be used as constants. However, this is not attractive for practical purposes because it may well be that, for example, 'π' in some texts is used as a constant and in some other text is used as a variable.

On the meta level—when we speak *about* expressions—we will often indicate free variables of an expression as subscripts of the expression, i.e. A_x should indicate that x occurs free in A. We do not fix in general, whether A_x should mean that x is *among* the free variables of A or whether x is *the only* free variable in A. On the object level, however, subscripting is often used as a hidden function symbol like in the example of f_m above. In other situations subscripting is used to distinguish between constants or variables of similar nature, e.g. if two constants for polynomials are required one often likes to name them p_1 and p_2. The meaning of subscripting must be understood from the context. The same is true for similar notations.

1.6.2 More Quantifiers

The different behavior w.r.t. substitution of free and bound variables as illustrated above should guide us in distinguishing free and bound variables. Of course, the quantifiers $\forall$ and $\exists$ turn free variables into bound variables. We will see that also other language constructs show this behavior. We will classify all language constructs, which bind variables, as "quantifiers". Examples of such quantifiers are given in Table 1.6.

Quantifier	Meaning	Binds
$\{x \mid A\}$	"the set of all x satisfying A"	x
$\underset{x}{\{T \mid A\}}$	"the set of all T with x satisfying A"	x
$\sum_{k=1}^{n} A$	"the sum of all A when k ranges from 1 to n"	k
$\int f[x]\, dx$	"the integral of f"	x
$\underset{x}{\lambda}\, T$	"the function mapping x to T"	x
$\underset{x}{\ni}\, A$	"such an x satisfying A"	x
$F[x] \mid_{x=5}$	"$F[x]$ where x equals 5"	x

Table 1.6: Special quantifiers.

The last two quantifiers in Table 1.6—the "such a" and the "where" quantifier—are often used in their natural language form and only rarely using the symbols "∋" and "|".

If you look through mathematical texts, you will probably find many more quantifiers. For many of these quantifiers, special forms as described in Section 1.5.3.3 (special ranges, omitting conditions, etc.) are available. In most of the cases, the meaning of quantifiers is explained when they are introduced in a theory. The crucial thing is to *recognize* when certain language constructs bind variables!

Exercise 2

Analyze the syntactical structure of the following axiom of the real numbers (as given in some lecture notes):

$$\exists\, 0 \in K\ \forall\, x \in K:\ x + 0 = x$$

$$\exists\, 1 \in K\ \forall\, x \in K:\ x \cdot 1 = x$$

Solution →

Exercise 3

Analyze the syntactical structure of the following theorem:

Every field $(K, +, \cdot)$ is a K-vector-space.

Solution →

Exercise 4

Analyze the syntactical structure of the following theorem:

> In a field $(K, +, \cdot)$ the inverse elements are unique.

Solution →

Exercise 5

Analyze the syntactical structure of the following definitions (as taken from existing lecture notes):

> Let $M \neq \emptyset$. A mapping $x : \mathbb{N} \to M$ is called *a sequence (in M)*. For $x[n]$ we write x_n, for x we often write (x_n) or $(x_n)_{n\in\mathbb{N}}$.
> (x_n) *converges to* x if and only if for all $\epsilon > 0$ there exists a natural number n_0 such that for all $n \geq n_0 : \|x_n - x\| < \epsilon$. In this case we write $x_n \to x$ or $x_n \underset{n\in\mathbb{N}}{\to} x$ or $x = \lim\limits_{n\to\infty} x_n$.

Solution →

Exercise 6

Analyze the syntactical structure of

$$\sum_{k=1}^{\infty} b_k = \sum_{k=1}^{\infty} a_k$$

using all knowledge on notation for series available from Analysis.

Solution →

Exercise 7

Analyze the syntactical structure of the following theorem:

> Let $\sum\limits_{n=1}^{\infty} a_n$ and $\sum\limits_{n=1}^{\infty} b_n$ be two absolutely convergent series. Then their Cauchy-product $\sum\limits_{n=1}^{\infty} c_n$ is absolutely convergent and

$$\sum_{n=1}^{\infty} c_n = \left(\sum_{n=1}^{\infty} a_n\right) \cdot \left(\sum_{n=1}^{\infty} b_n\right).$$

Solution →

Exercise 8

Analyze the syntactical structure of the following definition (as taken from existing lecture notes):

Let $A, B \subseteq \mathbb{R}$ and $x \in A$. A function f is *continuous at* x if and only if for all sequences (x_n) in A:

$$x_n \to x \bigwedge_{n\in\mathbb{N}} \forall\, x_n \geq x \Longrightarrow f[x_n] \to f[x]\,.$$

Solution →

Exercise 9

Analyze the syntactical structure of the following theorem:

Let $f : A \to B$ bijective. Then there is a function $g : B \to A$ such that
(i) $g \circ f = \mathrm{id}_A$
(ii) $f \circ g = \mathrm{id}_B$.

Solution →

1.6.3 Substitution

The operation of substitution has three arguments:

- an expression e in which the substitution takes place,
- a (free) variable x for which a term should be substituted, and
- a term t which is substituted for the variable.

We will write

```
SUBSTITUTED[e, x, t]
```

for the expression that results from e by substituting t for x at *all free occurrences* of x.

Note that t is substituted for *all* free occurrences of x in e. This is in sharp contrast to the operation of "replacement", which we will discuss in Section 1.6.5.

We have already discussed an other operation that can be applied to expressions containing variables, namely "quantification". Quantifiers "bind" variables. Variables bound by quantifiers are not any more free to the outside. More precisely, the operation of substitution does not substitute terms for bound variables.

The interaction of quantification and substitution has one more subtle aspect: If a free variable x occurs inside the scope of a quantifier that quantifies variable y then, by substituting for x a term t containing a free variable y, the free variable y in t would be "caught", i.e. turned into a bound variable! This must be avoided because it would drastically change the meaning of the expression. Hence, in such a case, before the actual substitution is carried out the

bound variable y must be "renamed". As we discussed above, it is a characteristic feature of bound variables that renaming them does not affect the meaning of the expression. This renaming of bound variables as a part of the substitution process should be well understood. It is an aspect of substitution that is less known.

If, for some x and t,

```
f = SUBSTITUTED[e, x, t]
```

then we say that "f is an instance of e".

We will train the operation of substitution, including the subtle aspect of renaming of bound variables, in a couple of examples.

Example: Substitution of a Constant for a Variable

$$\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{n\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall_{m}}\ |f_m - a| < \epsilon,\ a,\ 5\Big]$$

is

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{n\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall_{m}}\ (|f_m - 5| < \epsilon)$$

Example: Substitution of a Term for a Variable

$$\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{n\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall_{m}}\ |f_m - a| < \epsilon,\ a,\ (u+v)^2\Big]$$

is

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{n\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall_{m}}\ (|f_m - (u+v)^2| < \epsilon)$$

Example: A Free Variable in the Substitution Term Caught by a Quantifier

If we substituted n for a in

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{n\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall_{m}}\ |f_m - a| < \epsilon$$

just blindly we would obtain

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall_{\epsilon}}\ \underset{\boldsymbol{n}\in\mathbb{N}}{\exists_{n}}\ \underset{m\in\mathbb{N}\wedge m>\boldsymbol{n}}{\forall_{m}}\ |f_m - \boldsymbol{n}| < \epsilon$$

Carefully think about why this is undesirable:

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |f_m - a| < \epsilon$$

means that f converges to a, whereas

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |f_m - n| < \epsilon$$

means something completely different. Hence,

$$\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |f_m - a| < \epsilon,\ a,\ n\Big]$$

must be defined to be something like

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{N\in\mathbb{N}}{\exists N}\ \underset{m\in\mathbb{N}\wedge m>N}{\forall m}\ |f_m - n| < \epsilon$$

i.e. we must introduce a new bound variable N, which should not appear elsewhere in the expression in order to avoid dangerous "variable clashes". Note that the last formula has the appropriate meaning again, namely it means that f converges to n.

Example: Simultaneous Substitution Versus Successive Substitution

Often, terms are substituted for various free variables simultaneously: Let us read

$$\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |f_m - a| < \epsilon,\ f,\ a+b,\ a,\ f[a]+f[b]\Big]$$

as the result of substituting $a+b$ for f and $f[a]+f[b]$ for a simultaneously yielding

$$\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |(a+b)_m - (f[a]+f[b])| < \epsilon\ .$$

meaning "the sum sequence $a+b$ converges to $f[a]+f[b]$. This has to be carefully distinguished from successive substitution:

$$\text{SUBSTITUTED}\Big[\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |f_m - a| < \epsilon,\ f,\ a+b\Big],\ a,\ f[a]+f[b]\Big]$$

yields

$$\text{SUBSTITUTED}\Big[\underset{\epsilon\in\mathbb{R}\wedge\epsilon>0}{\forall\epsilon}\ \underset{n\in\mathbb{N}}{\exists n}\ \underset{m\in\mathbb{N}\wedge m>n}{\forall m}\ |(a+b)_m - a| < \epsilon,\ a,\ f[a]+f[b]\Big]$$

which is

$$\forall_{\epsilon \in \mathbb{R} \wedge \epsilon > 0} \; \exists_{n \in \mathbb{N}} \; \forall_{m \in \mathbb{N} \wedge m > n} \; |(f[a] + f[b] + b)_m - (f[a] + f[b])| < \epsilon$$

This is again different from

$$\text{SUBSTITUTED}\Big[\text{SUBSTITUTED}\Big[\forall_{\epsilon \in \mathbb{R} \wedge \epsilon > 0} \; \exists_{n \in \mathbb{N}} \; \forall_{m \in \mathbb{N} \wedge m > n} \; |f_m - a| < \epsilon,\, a,\, f[a] + f[b]\Big],\, f,\, a + b\Big]$$

which is

$$\text{SUBSTITUTED}\Big[\forall_{\epsilon \in \mathbb{R} \wedge \epsilon > 0} \; \exists_{n \in \mathbb{N}} \; \forall_{m \in \mathbb{N} \wedge m > n} \; |f_m - (f[a] + f[b])| < \epsilon,\, f,\, a + b\Big]$$

and, hence,

$$\forall_{\epsilon \in \mathbb{R} \wedge \epsilon > 0} \; \exists_{n \in \mathbb{N}} \; \forall_{m \in \mathbb{N} \wedge m > n} \; |(a + b)_m - ((a + b)[a] + (a + b)[b])| < \epsilon \;\; .$$

1.6.3.1 A Notation for Substitution

In the sequel, let us write

$$e_{x,y,z \to s,t,u}$$

for

$$\text{SUBSTITUTED}[e,\, x,\, s,\, y,\, t,\, z,\, u].$$

Note that parallel substitution

$$e_{x,y \to s,t}$$

is in general not identical to successive substitution

$$(e_{x \to s})_{y \to t}$$

and that this may be also different from

$$(e_{y \to t})_{x \to s} \;\; .$$

1.6.4 Declaration of (Free) Variables

In mathematical texts, there are (at least) two ways of declaring which symbols in a formula are considered to be (free) variables and which symbols are considered to be "constants" (which do not allow any substitution):

- explicit declaration of variables at the beginning of the text; this declaration is then valid for all subsequent formulae;
- explicit declaration of variables at the beginning of each formula.

(In *Theorema*, we use the second alternative. We will explain this in more detail later.)

In most math text books and publications, the first alternative is chosen. In fact, in most texts the authors assume that the reader makes the appropriate guesses "from the context". For example, if the group axioms

$$x \circ (y \circ z) = (x \circ y) \circ z$$

$$1 \circ x = x$$

$$x \circ x^{-1} = 1$$

are given, it is tacitly assumed that one considers 'x', 'y', and 'z' as (free) variables.

Alternatively, instead of considering formulae with free variables, we may just consider the formulae with all free variables quantified by a universal quantifier. This is particularly appropriate for formulae in "knowledge bases", see below. Formulae without free variables are called "closed". For example, the above axioms could also be written in the form

$$\forall_{x,y,z} \; x \circ (y \circ z) = (x \circ y) \circ z$$

$$\forall_{x} \; 1 \circ x = x$$

$$\forall_{x} \; x \circ x^{-1} = 1$$

Universal quantification of free variables is, however, not appropriate for formulae with free variables that appear as goals in solution or simplification situations, see below. Therefore, we *do not* want to introduce it as a *general rule*, that free variables should always be considered as universally quantified!

1.6.5 Replacement

The operation of replacement has three arguments:

- an expression e in which a replacement takes place,
- a position p at which the replacement takes place,
- and an expression f which replaces the expression in e at position p.

We will write

```
REPLACED[e, p, f]
```

for the expression that results from e by replacing the expression at position p in e by the expression f.

Note that, in replacement, f is replacing just one occurrence of a subexpression of e, namely the subexpression of e at one particular position p and that the subexpression at position p need not be a variable but may be an arbitrary expression. This is in sharp contrast to the operation of "substitution", which we discussed in Section 1.6.3.

Again one must be careful with replacing subexpression by expressions in expressions that contain quantifiers. We assume that the replacement operation takes care of the necessary renaming of bound variables if name clashes occur during replacement.

1.6.5.1 Positions

Positions of subexpressions in expressions can be described uniquely by tuples of natural numbers taking into account the nested structure of expressions in standard syntax. For example, in the formula

$$(x + y) * (x + y) = x^2 + 2 * x * y + y^2 \tag{1}$$

there are two positions at which the subexpression

$$x + y$$

occurs. These two positions could be described by the tuples $\langle 1, 1\rangle$ and $\langle 1, 2\rangle$, respectively, because in the standard syntax representation of (1)

```
•[  (x + y) * (x + y) = x^2 + 2 * x * y + y^2  ] // InputForm
```

```
TM Equal[TM Times[TM Plus[x, y], TM Plus[x, y]], TM Plus[TM Power[x, 2], TM Times[2, x, y],
 TM Power[y, 2]]]
```

the first occurrences of ™Plus[x, y] can be visited by going to the 1-st and then again 1-st subexpression of (1) and the second occurrence can be visited by going to the 1-st and then to the 2-nd subexpression of (1).

We will rarely use this explicit description of positions in this practical introduction to predicate logic. Most times it will suffice to say something like "the first occurrence" or the "third occurrence" of a subexpression.

- Mathematica offers an operation ReplacePart, which does exactly what REPLACED does, i.e. we can implement REPLACED using basic Mathematica operations. The notion of "position", which we introduced above, corresponds exactly to the notion of position in Mathematica, except that Mathematica uses braces '{' and '}' to denote tuples instead of angle brackets '⟨' and '⟩'.

```
Position[•[  (x + y) * (x + y) = x^2 + 2 * x * y + y^2  ], •[  x + y  ]]
{{1, 1}, {1, 2}}
```

```
REPLACED[e_, p_, f_] := ReplacePart[e, f, p /. AngleBracket → List]
```

We will train the operation of replacement in a couple of examples.

Examples

```
REPLACED[•[  (x + y) * (x + y) = x^2 + 2 * x * y + y^2  ], ⟨1, 1⟩, •[  √(u^2)  ]]
(√(u^2)) * (x + y) = x^2 + 2 * x * y + y^2
```

whereas

```
REPLACED[•[  (x + y) * (x + y) = x^2 + 2 * x * y + y^2  ], ⟨1, 2⟩, •[  √(u^2)  ]]
(x + y) * (√(u^2)) = x^2 + 2 * x * y + y^2
```

Also,

```
REPLACED[•[  (x + y) * (x + y) = x^2 + 2 x * y + y^2  ], ⟨2, 1⟩, •[  √(u^2)  ]]
(x + y) * (x + y) = (√(u^2)) + 2 * x * y + y^2
```

(Note that, at this moment, we are not yet concerned with whether such replacements are "reasonable" or "valid"! We will be concerned with "valid" transformations later, when we start to discuss proving, solving, and simplifying within predicate logic.)

1.6.5.2 A Notation for Replacement

We will adopt the notation

$e_{p \to\!\!\!| f}$

for

REPLACED[e, p, f]

i.e.

$e_{\_p\_ \to\!\!\!| f\_} := \text{REPLACED}[e, p, f]$

$\bullet[\ (x+y)*(x+y) = x^2 + 2\,x*y + y^2\ ]_{\langle 2,2\rangle \to\!\!\!| \bullet[\sqrt{u^2}\,]}$

$(x+y)*(x+y) = x^2 + \left(\sqrt{u^2}\right) + y^2$

1.7 An Informal Semantics

The semantics of expressions in predicate logic is defined inductively following the syntactical structure of the expression. We will not give a formal definition of semantics, rather we will try to give an idea how *meaning* is attached to expressions in predicate logic. For assigning semantics to expressions, we need

- a non-empty "universe of discourse" U,
- concrete functions on U (of arity 1, 2, 3, etc.) and concrete predicates (i.e. relations) on U (of arity 1, 2, 3, etc.),
- an "interpretation", which interprets
  - □ object constants as distinct concrete objects in U,
  - □ each n-ary function constant as some n-ary function on U, and
  - □ each n-ary predicate constant as some n-ary relation on U,
- an assignment mapping for variables, which maps each variable to a concrete object in U.

1.7.1 Semantics for Terms

Terms mean some concrete object from U. A term without free variables has a unique meaning, a term containing free variables typically has different meanings depending on the assignment for the variables.

Expression	Meaning
constant	the interpretation of the constant
variable	the object assigned to the variable
$f[t_1, \ldots, t_n]$	the interpretation of f applied to the interpretations of the t_i

Table 1.7: Semantics for terms.

Important: Distinct constants denote distinct objects, whereas distinct variables may denote the same object, i.e. in case of constants 'x' and 'y' we know $x \neq y$, in case of variables 'x' and 'y' it can be that $x = y$.

Example

Consider the universe of discourse the real numbers and consider the functions "absolute value", "subtraction", "reciprocal", and "function evaluation". We interpret ™BracketingBar, ™Minus, ™Subscript, and f as the absolute value function, the subtraction function, function evaluation, and the reciprocal function, respectively. The meaning of the compound term $|f_m - a|$, i.e.

™BracketingBar[™Minus[™Subscript[f, •var[m]], •var[a]]] ,

is then

"the absolute value of the difference between the reciprocal of m and a"

(with m and a free variables). If we assign to m the real number 5 and to a the real number 0 then the meaning is $\frac{1}{5}$, if we assign to m the real number π and to a the real number $\frac{3}{\pi}$ then the meaning is $\frac{2}{\pi}$.

Example

Consider the universe of discourse the real numbers and consider the functions "absolute value", "subtraction", "reciprocal", and "function evaluation". We interpret ™BracketingBar, ™Minus, ™Subscript, and f as the square root function, the multiplication function, function evaluation, and the squaring function, respectively. The meaning of the compound term $|f_m - a|$, i.e.

™BracketingBar[™Minus[™Subscript[f, •var[m]], •var[a]]] ,

is then

"the square root of the product of m squared and a"

(with m and a free variables). If we assign to m the real number 5 and to a the real number 0 then the meaning is 0, if we assign to m the real number π and to a the real number $\frac{3}{\pi}$ then the meaning is (the real number) $\sqrt{3\pi}$.

1.7.2 Semantics for Formulae

Formulae without free variables mean "true" or "false". Truth or falsity of a formula containing free variables may depend on the assignment for the variables.

Expression	**Meaning**
$p[t_1, \ldots, t_n]$	the interpretation of p applied to the interpretations of the t_i
$\neg A$	interpret A; then use truth table for $\neg$
$A \vee B$	interpret A and B; then use truth table for $\vee$
$A \wedge B$	interpret A and B; then use truth table for $\wedge$
$A \Rightarrow B$	interpret A and B; then use truth table for $\Rightarrow$
$A \Leftrightarrow B$	interpret A and B; then use truth table for $\Leftrightarrow$
$\underset{x}{\forall}\, A$ C	"true" if the interpretation of A is "true" **for each** assignment for x, which makes the interpretation of C "true" "false" otherwise
$\underset{x}{\exists}\, A$ C	"true" if the interpretation of A is "true" **for some** assignment for x, which makes the interpretation of C "true" "false" otherwise

Table 1.8: Semantics for formulae.

The meaning of atomic formulae is decided by interpretation, propositional formulae are decided by truth tables, whereas the meaning of quantified formulae must be decided by following the "recipe" given in Table 1.8. It is important to understand that the formulations "for each" and "for some" in the right column of Table 1.8 are formulations on the *meta level*, i.e. on the level where one speaks *about* the language, whereas "$\forall$" (speak: "for all") and "$\exists$" (speak: "there exists") are on the object level, i.e. the level of the language! This means that for $\underset{x}{\forall}\, A$ to be "true" we must ensure that A is "true" *for all possible values* in the universe of discourse substituted for x, and for $\underset{x}{\exists}\, A$ to be "true" we must argue that we can substitute an object from the universe of discourse for x in order to make A "true".

Note in particular, that quantifiers with finite ranges have the same meaning as $\wedge$ and $\vee$, e.g.

$$\forall_{x\in\{1,\ldots,n\}} \mathrm{A[x]} \text{ is the same as } A[1] \wedge \ldots \wedge A[n] \text{ and}$$

$$\exists_{x\in\{1,\ldots,n\}} \mathrm{A[x]} \text{ is the same as } A[1] \vee \ldots \vee A[n] .$$

It is therefore appropriate to view the universal quantifier as a generalization of "$\wedge$" and the existential quantifier as a generalization of "$\vee$" for cases of arbitrary number or infinitely many operands. This is also the reason, why $\forall_x A$ and $\exists_x A$ is sometimes written as $\bigwedge_x A$ and $\bigvee_x A$.

Example

Consider the universe of discourse the real numbers and consider the functions "absolute value", "subtraction", "reciprocal", "function evaluation" and the relation "being less than". We interpret ™BracketingBar, ™Minus, ™Subscript, and f as the absolute value function, the subtraction function, function evaluation, and the reciprocal function, respectively. We interpret ™Less as the "less than" relation. The meaning of the atomic formula $|f_m - a| < \epsilon$, i.e.

™Less[™BracketingBar[™Minus[™Subscript[f, •var[m]], •var[a]]], •var[ϵ]] ,

is the property that

> "the absolute value of the difference between the reciprocal of m and a is less than ϵ"

(with m, a, and ϵ free variables). If we assign to m the real number 5, to a the real number 0, and to ϵ the real number $\frac{1}{3}$, then the meaning is "$\frac{1}{5}$ is less than $\frac{1}{3}$", which is "true", if we assign to m the real number π, to a the real number $\frac{3}{\pi}$, and to ϵ the real number $\frac{1}{\pi}$, then the meaning is "$\frac{2}{\pi}$ is less than $\frac{1}{\pi}$", which is "false".

Example

Consider the universe of discourse the real numbers and "all sets". Consider the object "the set of natural numbers" and the functions "absolute value", "subtraction", "reciprocal", "function evaluation" and the relations "being less than", "being greater than", and "membership". We interpret ™ℕ as the set of natural numbers, ™BracketingBar, ™Minus, ™Subscript, and f as the absolute value function, the subtraction function, function evaluation, and the reciprocal function, respectively. We interpret ™Less, ™Greater, and ™Element as the "less than", the "greater than", and the "membership" relation, respectively. The meaning of the quantified formula $\underset{m\in\mathbb{N}\wedge m>n}{\forall_m} |f_m - a| < \epsilon$, i.e.

```
™ForAll[•range[•simpleRange[•var[m]]],
  ™And[™Element[•var[m], ™ℕ], ™Greater[•var[m], •var[n]]],
  ™Less[™BracketingBar[™Minus[™Subscript[f, •var[m]], a]], •var[ϵ]]]
```

is the property that

> "for all m
> with the property, that m is a member of the set of natural numbers and m is greater than n,
> the absolute value of the difference between the reciprocal of m and a is less than ϵ"

(with n, a and ϵ free variables). If we assign to n the real number 3, to a the real number 0, and to ϵ the real number $\frac{1}{3}$, then the meaning is: "for all natural numbers m greater than 3 the absolute value of the reciprocal of m is less than $\frac{1}{3}$", which is "true". Why is this true? Suppose m is a natural number greater than 3, then, by laws for inequalities, we know that $\frac{1}{m}$ is less than $\frac{1}{3}$, and from this, since $\frac{1}{m}$ is positive, we know that also the absolute value of $\frac{1}{m}$ is less than $\frac{1}{3}$.

If we assign to n the real number 1, to a the real number 0, and to ϵ the real number $\frac{1}{3}$, then the meaning is: "for all natural numbers m greater than 1 the absolute value of the reciprocal of m is less than $\frac{1}{3}$", which is "false". Why is this false? Because the statement "absolute value of the reciprocal of m is less than $\frac{1}{3}$" is *not* true *for all* natural numbers m greater than 1. Substitute e.g. 2 for m, then we are left with "absolute value of $\frac{1}{2}$ is less than $\frac{1}{3}$", which is false.

Example

Interpret symbols as in the previous example. The meaning of the quantified formula $\underset{n\in\mathbb{N}}{\exists_n} \underset{m\in\mathbb{N}\wedge m>n}{\forall_m} |f_m - a| < \epsilon$, i.e.

```
TMExists[•range[•simpleRange[•var[n]]],
 TMElement[•var[n], TMℕ],
 TMForAll[•range[•simpleRange[•var[m]]],
  TMAnd[TMElement[•var[m], TMℕ], TMGreater[•var[m], •var[n]]],
  TMLess[TMBracketingBar[TMMinus[TMSubscript[f, •var[m]], a]], •var[ϵ]]]
```

is the property that

> "there exists an n
> with the property, that n is a member of the set of natural numbers, such that
> for all m
> with the property, that m is a member of the set of natural numbers and m is greater than n,
> the absolute value of the difference between the reciprocal of m and a is less than ϵ"

(with a and ϵ free variables). If we assign to a the real number 0 and to ϵ the real number $\frac{1}{3}$, then the meaning is: "there exists a natural number n such that for all natural numbers m greater than n the absolute value of the reciprocal of m is less than $\frac{1}{3}$", which is "true". Why is this true? We substitute e.g. 10 for n. Suppose m is a natural number greater than 10, then, by laws for inequalities, we know that $\frac{1}{m}$ is less than $\frac{1}{10}$, and from this, since $\frac{1}{m}$ is positive, we know that also the absolute value of $\frac{1}{m}$ is less than $\frac{1}{10}$, thus, by transitivity of "less", $\frac{1}{m}$ is less than $\frac{1}{3}$.

If we assign to a the real number 1, and to ϵ the real number $\frac{1}{3}$, then the meaning is: "there exists a natural number n such that for all natural numbers m greater than n the distance between the reciprocal of m and 1 is less than $\frac{1}{3}$", which is "false". Why is this false? For all natural numbers greater than 3 the reciprocal is less than $\frac{1}{3}$, thus, the distance from 1 is greater than $\frac{2}{3}$. Therefore, no such number n can exist.

Exercise 10

Convince yourself that the de Morgan laws for negating quantifiers hold even for quantifiers with conditions, i.e.

$$(a)\quad \neg \underset{\substack{x\\C}}{\forall} A \Leftrightarrow \underset{\substack{x\\C}}{\exists} \neg A$$

$$(b)\quad \neg \underset{\substack{x\\C}}{\exists} A \Leftrightarrow \underset{\substack{x\\C}}{\forall} \neg A$$

Solution →

Exercise 11

We often need to express "there exists a *at most one* x such that A". Think about the logical structure of this formula and express it using only standard quantifiers.

Solution →

Exercise 12

The special form $\exists_x! A$ stands for "there exists a *unique* x such that A". Think about the logical structure of this formula and express it using only standard quantifiers.

Solution →

1.8 How to Define New Symbols in Terms of Known Symbols

The use of definitions can be nicely compared with the use of subprograms in programming. Subprograms are not necessary but convenient for giving structure to some program. The same is true for definitions. Definitions are not necessary for mathematics but convenient for structuring mathematical knowledge.

1.8.1 Explicit Definitions

1.8.1.1 Explicit Definition of Functions

An *explicit definition* of an n-ary function f is a statement of the form

$$f[x_1, \ldots, x_n] := \tau_{x_1,\ldots,x_n}$$

where f is a *new function symbol* and $\tau_{x_1,\ldots,x_n}$ is a term containing only known symbols and no free variables other than $x_1, \ldots, x_n$.

The intention is that $f[x_1, \ldots, x_n]$ is introduced as an abbreviation for $\tau_{x_1,\ldots,x_n}$ and the definition introduces knowledge about the new function symbol f, namely

$$\forall_{x_1,\ldots,x_n} \quad f[x_1, \ldots, x_n] = \tau_{x_1,\ldots,x_n}$$

called the *defining axiom* for f. It is save to introduce new symbols in this way as long as one observes the rule concerning the free variables. The defining axiom allows from then on to replace each occurence of $f[x_1, \ldots, x_n]$ by $\tau_{x_1,\ldots,x_n}$

for arbitrary $x_1, \ldots, x_n$. The term on the right hand side may not contain some of the variables on the left hand side, which is, however, an indication that the function does actually not depend on these variables and they can therefore be omitted also on the left hand side. If the defining term contains some additional free variable, say y, then the definition is *wrong* in the sense that replacing $f[x_1, \ldots, x_n]$ by $\tau_{x_1,\ldots,x_n,y}$ may change the meaning of the expression.

1.8.1.2 Explicit Definition of Predicates

An *explicit definition* of an n-ary predicate p is a statement of the form

$$p[x_1, \ldots, x_n] :\Leftrightarrow \varphi_{x_1,\ldots,x_n}$$

where p is a *new predicate symbol* and $\varphi_{x_1,\ldots,x_n}$ is a formula containing only known symbols and no free variables other than $x_1, \ldots, x_n$.

The intention is that $p[x_1, \ldots, x_n]$ is introduced as an abbreviation for $\varphi_{x_1,\ldots,x_n}$ and the definition introduces knowledge about the new predicate symbol p, namely

$$\underset{x_1,\ldots,x_n}{\forall} \quad p[x_1, \ldots, x_n] \Leftrightarrow \varphi_{x_1,\ldots,x_n}$$

called the *defining axiom* for p. It is save to introduce new symbols in this way as long as one observes the rule concerning the free variables. The defining axiom allows from then on to replace each occurence of $p[x_1, \ldots, x_n]$ by $\varphi_{x_1,\ldots,x_n}$ for arbitrary $x_1, \ldots, x_n$. The formula on the right hand side may not contain some of the variables on the left hand side, which is, however, an indication that the predicate does actually not depend on these variables and they can therefore be omitted also on the left hand side. If the defining formula contains some additional free variable, say y, then the definition is *wrong* in the sense that replacing $p[x_1, \ldots, x_n]$ by $\varphi_{x_1,\ldots,x_n,y}$ may change the meaning of the expression.

1.8.2 Implicit Function Definitions

Consider the square root function on the non-negative real numbers, which is usually defined for every $x \in \mathbb{R}_0^+$ as

$$\sqrt{x} := \text{the } y \in \mathbb{R}_0^+ \text{ such that } y^2 = x \ .$$

The characteristic feature of this type of definition is that a new function f is defined via a formula. More precisely, an n-ary function $f[x_1, \ldots, x_n]$ is defined to be *the* y such that $\varphi_{x_1,\ldots,x_n,y}$. In contrast to an explicit definition, the object $f[x_1, \ldots, x_n]$ is not defined by explicitely describing its structure, thus we call a definition of this type an *implicit function definition*.

Let's study the expression "the $y \in \mathbb{R}_0^+$ such that $y^2 = x$" in a little bit more detail. We can substitute different values for x, typically resulting in different values for y, e.g. "the $y \in \mathbb{R}_0^+$ such that $y^2 = 9$" is 3, whereas "the $y \in \mathbb{R}_0^+$ such that $y^2 = \pi^2$" is π. Substituting a value for y would turn the expression into some meaningless expression, e.g. "the $\pi \in \mathbb{R}_0^+$ such that $\pi^2 = 3$". Changing y into some other variable leaves the meaning of the expression unchanged, e.g. both "the $y \in \mathbb{R}_0^+$ such that $y^2 = 9$" and "the $z \in \mathbb{R}_0^+$ such that $z^2 = 9$" denote the value 3. This indicates, as discussed in Section 1.6.2, to view the language construct "*the* y such that $\varphi_{x_1,\ldots,x_n,y}$"—written as $\underset{y}{\exists!}\ \varphi_{x_1,\ldots,x_n,y}$—as a *quantifier* that binds y. The typical use of the $\exists!$-quantifier is in implicit definitions

$$f[x_1, \ldots, x_n] := \underset{y}{\exists!}\ \varphi_{x_1,\ldots,x_n,y} \quad . \tag{2}$$

Note that, when viewing the $\exists!$-quantifier as a quantifier that results in a term, there is no syntactic difference between an implicit definition and an explicit definition! Again, the right hand side of the definition is a term containing only known symbols and no free variables other than $x_1, \ldots, x_n$. However, the big difference between explicit and implicit definitions lies in how they are used. The defining axiom for f as defined in (2) is

$$\underset{x_1,\ldots,x_n,y}{\forall} f[x_1, \ldots, x_n] = y \iff \varphi_{x_1,\ldots,x_n,y} \tag{3}$$

and it can be shown that this is equivalent to

$$\underset{x_1,\ldots,x_n}{\forall}\ \underset{y}{\exists!}\ \varphi_{x_1,\ldots,x_n,y} \quad . \tag{4}$$

In contrast to the defining axiom of an explicit definition, which is always save to introduce as long as the restrictions on the free variables are not violated, an implicit definition needs a *proof* before it can be given. Typically, this is accomplished by proving the unique existence condition (4). More often even, proving a formula of the form (4) gives rise to introducing a new function f to be defined implicitely like in (2). The use of an implicit definition (in proving), however, relies in most of the examples on the defining axiom (3), i.e. each occurrence of an equality $f[x_1, \ldots, x_n] = y$ may be replaced by $\varphi_{x_1,\ldots,x_n,y}$ and vice versa.

Remark:

An explicit function definition $f[x_1, \ldots, x_n] = \tau_{x_1,\ldots,x_n}$ as described in Section 1.8.1 can be viewed just as a special case of an implicit definition with $\varphi_{x_1,\ldots,x_n,y}$ the formula $y = \tau_{x_1,\ldots,x_n}$. In this case, the unique existence condition (4) need not be proven because it holds for every term $\tau_{x_1,\ldots,x_n}$. The defining axiom specializes now to

$$\forall_{x_1,\ldots,x_n,y} f[x_1, \ldots, x_n] = y \iff y = \tau_{x_1,\ldots,x_n} \quad ,$$

which simplifies to

$$\forall_{x_1,\ldots,x_n} f[x_1, \ldots, x_n] = \tau_{x_1,\ldots,x_n} \quad ,$$

which is just what we chose as the defining axiom for the explicit definition.

1.8.2.1 Weaker Form of Implicit Definitions

There is also a weaker form of an implicit definition, which occurs e.g. in

$$\text{higherPrime}[x] := \text{such a } y \text{ with } y \text{ is prime and } y > x \ .$$

Similar to "the y such that" we can quickly convince ourselves that an expression "such a y with" ought to be viewed as a quantifier that binds y. We write $\exists_y \varphi_{x_1,\ldots,x_n,y}$ for "such a y with $\varphi_{x_1,\ldots,x_n,y}$". The idea is that

$$f[x_1, \ldots, x_n] := \exists_y \varphi_{x_1,\ldots,x_n,y} \tag{5}$$

introduces a new axiom

$$\forall_{x_1,\ldots,x_n,y} f[x_1, \ldots, x_n] = y \Longrightarrow \varphi_{x_1,\ldots,x_n,y} \tag{6}$$

called again the *defining axiom* for f, which is equivalent to

$$\forall_{x_1,\ldots,x_n} \exists_y \varphi_{x_1,\ldots,x_n,y} \ . \tag{7}$$

Again, the existence condition (7) is typically used to justify a definition of the form (5) or the proof of an existence formula of the form (7) gives the motivation to introduce a new function f defined implicitely in the form (5). One must be cautious, however, because it is *only the implication* in the defining axiom, which is available as knowledge on f. As we will see later when studying proof rules, this will not allow anymore unrestricted replacement of expressions of the form $f[x_1, \ldots, x_n] = y$. In particular, a proof of $f[x_1, \ldots, x_n] = y$ cannot be reduced anymore to just proving $\varphi_{x_1,\ldots,x_n,y}$, as it can be done in the case, when

f is defined using the $\exists!$-quantifier, because this is exactly where *unique* existence comes into play. However, it can be convenient to define functions in this weaker form, since in some cases this is all what is really needed.

Exercise 13

Give precise definitions of "the quotient of the integers x and y" and "the remainder of the integers x and y" in the language of predicate logic. Which additional information is needed for this definition to be "correct"?

Solution →

Exercise 14

Give a precise definition of "the inverse element of x w.r.t. multiplication" in the language of predicate logic. Which additional information is needed for this definition to be "correct"? When is this information available?

Solution →

Exercise 15

Give a precise definition of "the inverse function of f" in the language of predicate logic, see lecture notes Linear Algebra. Which additional information is needed for this definition to be "correct"? When is this information available?

Solution →

Exercise 16

Give a precise definition of $\sup A$ ("the supremum of a set A)" in the language of predicate logic according to Definition 2.9 in the lecture notes Analysis. Which additional information is needed for this definition to be "correct"? When is this information available? Think about how "sup" is used e.g. in Definition 2.11.

Solution →

Exercise 17

Give a precise definition of "the limit of the sequence (x_n)" in the language of predicate logic according to Definition 3.2 in the lecture notes Analysis. Which additional information is needed for this definition to be "correct"? When is this information available? Think about how "limit of" is used e.g. in Theorem 3.5.

Solution →

Exercise 18

Give a precise formulation of Definition 3.15 from the lecture notes Analysis in the language of predicate logic. If necessary, give precise definitions for underlying concepts. Give a formulation of Theorem 3.16 from the lecture notes Analysis in predicate logic using the concepts defined before.

Solution →

Exercise 19

Give a precise definition of "the derivative of a function f" in the language of predicate logic according to Definition 6.1 from the lecture notes Analysis. Give precise definitions for all underlying concepts. Which role plays Theorem 6.2?

Solution →

Exercise 20

Give a precise definition of "s is orthogonal to A" (for $s \in \mathbb{R}^n$ and A a straight line) in the language of predicate logic, see lecture notes Linear Algebra. Give precise definitions for all underlying concepts.

Solution →

Exercise 21

Give a precise definition of "p is the interpolating polynomial for the tuples x and y" in the language of predicate logic, see lecture notes Algorithmic Methods. Give precise definitions for all underlying concepts.

Solution →

Example (Danger of weak implicit definitions)

Consider the concepts of "bounded sets" and "upper bound of a set". A set $A \subseteq M$ is called *bounded from above* iff there is a $c \in M$, which is greater equal all elements of A. Every such c is then called an *upper bound of* A. Written in predicate logic:

$$\text{is-bounded}[A, M] :\Leftrightarrow \underset{c\in M}{\exists} \underset{x\in A}{\forall} x \le c$$

is an explicit definition of a new predicate "is-bounded". For "upper-bound" one would be tempted to introduce a *function* implicitely defined as

$$\text{upper-bound}[A, M] := \underset{c\in M}{\ni} \underset{x\in A}{\forall} x \le c$$

so that the fact that "10 is an upper bound of A in M" can be stated as

$$\text{upper-bound}[A, M] = 10\,. \tag{8}$$

Intuitively, this seems to be fine. Now consider the well-known theorem that every number greater than an upper bound must again be an upper bound. From this theorem together with (8) we can then easily infer

$$\text{upper-bound}[A, M] = 12\,, \tag{9}$$

from which, by symmetry and transitivity of equality, we get the contradiction

$$10 = 12\,.$$

Example (Predicate instead of implicitely defined function)

Weak implicit definitions are often better expressed as predicates. In the example above, instead of the function "upper-bound" one would rather define a predicate

$$\text{upper-bound}[A, M, c] :\Leftrightarrow c \in M \bigwedge_{x \in A} \forall\, x \leq c\,.$$

(It can, in general, often help clarify the structure if one introduces new symbols with each quantifier in a sequence of quantifiers.

$$\text{is-bounded}[A, M] :\Leftrightarrow \exists_{c}\, \text{upper-bound}[A, M, c]$$

In a next step one could now introduce a quantifier for M and introduce a new unary function/predicate depending only on A. Think about what the meaning of this would then be. See also next example.)

The fact that "10 is an upper bound of A in M" would then be written as

$$\text{upper-bound}[A, M, 10]\,.$$

The theorem would now allow to infer

$$\text{upper-bound}[A, M, 12]\,,$$

which does not cause any problems. (As soon as we have studied rules for proving, we will see that in the formulation using a *function* "upper-bound" it would not even be possible to *prove* the theorem, which—on the other hand—saves us from the contradiction $10 = 12$ arising in the previous example.)

Example (Structured definitions by introducing only one quantifier at the time)

Consider the term

$$\frac{f[x] - f[a]}{x - a}$$

containing the free variables f, a, x. It may, thus, be reasonable to define a 3-ary function

$$\text{gradient-secant-between}[f, a, x] := \frac{f[x] - f[a]}{x - a}$$

giving the "gradient of the secant between the points $(x, f[x])$ and $(a, f[a])$". Now, binding x through the lim-quantifier, we can define

$$\text{gradient-at}[f, a] := \lim_{x \to a} \text{gradient-secant-between}[f, a, x] \quad \left(= \lim_{x \to a} \frac{f[x] - f[a]}{x - a} \right)$$

being the "gradient of f at a". Now, binding a through the λ-quantifier, we can define the derivative of f as

$$\text{derivative}[f] := \lambda_a \text{ gradient-at}[f, a]$$

$$\left(= \lambda_a \lim_{x \to a} \text{gradient-secant-between}[f, a, x] = \lambda_a \lim_{x \to a} \frac{f[x] - f[a]}{x - a} \right).$$

(Note, that we did not bother with existence of limits etc. in this example. The complete and appropriate formulation of differentiability and related concepts can be found in the next example.)

Example (Structured definitions by introducing only one quantifier at the time)

We have a notion of *limit* for functions (see Definition 5.24 lecture notes Analysis):

$$\text{is-limit-of-at}[f, a, y] :\Leftrightarrow \forall_{\epsilon > 0} \exists_{\delta > 0} \forall_{x \in U_\delta[a] \setminus \{a\} \cap \mathcal{D}[f]} f[x] \in U_\epsilon[y] \quad \left(i.e. \lim_{x \to a} f[x] = y \right)$$

$$\text{gradient-secant-between}[f, a, x] := \frac{f[x] - f[a]}{x - a}$$

In fact, we want to view gradient-secant-between as a function of x:

$$\text{gradient-between}[f, a] := \lambda_x \text{ gradient-secant-between}[f, a, x] \quad \left(= \lambda_x \frac{f[x] - f[a]}{x - a} \right)$$

$$\text{converges-gradient-between-to}[f, a, y] :\Leftrightarrow \text{is-limit-of-at}[\text{gradient-between}[f, a], a, y]$$

$$\text{differentiable}[f, a] :\Leftrightarrow \exists!_y \text{ converges-gradient-between-to}[f, a, y]$$

$$\text{derivative}[f, a] := \underset{y}{\ni!}\ \text{converges-gradient-between-to}[f, a, y]$$

For $\text{derivative}[f, a]$ we introduce the notion $f'[a]$, where in this case " ' " is a binary function symbol written in infix notation.

$$\text{differentiable}[f] :\Leftrightarrow \underset{a}{\forall}\ \text{differentiable}[f, a]$$

$$\text{derivative}[f] := \underset{a}{\lambda}\ \text{derivative}[f, a]$$

For $\text{derivative}[f]$ we introduce the notion f', where in this case " ' " is a unary function symbol written in postfix notation. It must be noticed that in the definition (lecture notes Analysis p.87)

$$\begin{aligned} f' &: I \to \mathbb{R} \\ & x \mapsto f'[x] \end{aligned}$$

the " ' "-symbol in the first line is the unary postfix function symbol, whereas in the second line it is the binary infix function symbol, i.e.

$$\begin{aligned} \text{derivative}[f] &: I \to \mathbb{R} \\ & x \mapsto \text{derivative}[f, x] \end{aligned}$$

telling that (for all x)

$$\text{derivative}[f][x] = \text{derivative}[f, x] \quad .$$

The process of representing a binary function φ as a unary function, whose value when applied to the first argument is itself a function, which is then applied to the second argument, is called *currying*. In general, currying reduces n-ary functions to nested applications of unary functions. By currying we can for instance think of $x + y + z$ ($^{TM}\text{Plus}[x, y, z]$) as

$$^{TM}\text{Plus}[^{TM}\text{Plus}[x][y]][z]$$

where

$$^{TM}\text{Plus}[x] := \underset{u}{\lambda}\ x + u \quad .$$