Strong Symbols

Henning Thielemann

2004-02-08

Strong Symbols

Contents

1. Why Symbols?
2. Meaning of symbols
3. Ambiguity - One notation, different meanings
4. Redundancy - Different notations, one meaning
5. Common abuse of notation

Natural languages vs. formal languages

Natural language:

- Developed by many people with few central control.
- Quite ambiguous (allowing for puns), complicated, inconsistent.
- Just the often used phrases tend to inconsistencies.
- Hard to analyse for machines.

Formal and programming language:

- Developed and maintained by few people.
- Various levels of matching theoretical goals.
- Analysable for machines.
- Usually we learn a natural language first and later some formal languages explained in terms of a natural one.

Why symbols?

- Chemists use structural formulas and equations for chemical reactions.
- Musicians use notes on staffs and symbols for interpretation.
- Mathematicians use mathematical formulas.
- Computer scientists use programming languages.

- Symbols abbreviate natural language.
- Symbols are easier to realise.
- Formalisms may prevent from ambiguity.
- Formalisms may be understood by machines.

The reality looks different

The last two points are goals that are usually not matched:

- Chemical formulas contain not enough information for synthesising substances or simulating reactions.
- Musical scores allow for much interpretation.
- Theorems in mathematical articles cannot be proven with a machine, calculations can't be executed.
- Computer programs can be executed by a machine, they can be processed by other programs, certain properties can even be proven! But programming language differ very much in robustness, simplicity, orthogonality, consistency, expressiveness.

Even worse: Different notations have evolved for some mathematical applications:

- Physics: $s = s(t)$ meaning "The way s is a quantity that depends on the time t"
- Stochastics: A variate X represented by a probability density function p
- Differential equations: $u_t = \Delta u$

Problems on the way to more formal expressions:

- Theory evolves and when getting new insights some of the formalism turns out to be inappropriate.
- When creating new formalisms habit is often stronger than the will for matching goals like consistency.

Example

Consider famous $3n + 1$ problem:

Given the function f with

$$f(n) = \begin{cases} \frac{n}{2} & : n \equiv 0 \mod 2 \\ 3n + 1 & : n \not\equiv 0 \mod 2, \end{cases}$$

apply f iteratively to a start value, e.g.

$$7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \cdots$$

That's what some pupil write down for calculation:

$$7 \cdot 3 + 1 = 22 : 2 = 11 \cdot 3 + 1 = 34 : 2 = 17 \cdot 3 + 1 = 52$$

This is obviously a wrong expression. The problem is certainly a misunderstanding of the '=' sign. It does not mean

Continuation of the calculation

but

Left hand side and right hand side have equal value.

You think such problems are an issue for pupils only?

You are definitely wrong!

Mathematical notation is full of inconsistencies, fuzzy definitions, and even worse intentional abuse.

But: Many problems can be avoided if one wants to.

From mathematical notation to programming languages

Because mathematical formulas are commonly considered as state of the art of exactness many programming language designers adapted common mathematical notation – and ran into serious troubles.

Example: Differentiate an expression for a variable x and evaluate it for, say $x = 2$:

$$\left(\frac{\mathrm{d}}{\mathrm{d}\,x}\log x\right)\Bigg|_{x=2}$$

Example: Computer algebra system `Mathematica`:

Replace all occurences of `x` by `2`

```
In>  ReplaceAll[Log [x],x->2]
```

$$\text{Out>}\quad \texttt{Log [2]}$$

Derive `Log [x]` with respect to `x`

```
In>  D[Log [x],x]
```

$$\text{Out>}\quad \frac{1}{x}$$

Derive and evaluate for `x->2`

```
In>  ReplaceAll[D[Log [x],x],x->2]
```

$$\text{Out>}\quad \frac{1}{2}$$

Inconsistency! If ***ReplaceAll*** *would actually replace* each *occurence of `x` by `2` the expression would reduce to:*

```
In>  D[Log [2],2]
```

General*::"ivar": "2 is not a valid variable."*

$$\text{Out>}\quad \partial_2 \texttt{Log [2]}$$

Problem:

While expressions like $x+2$ denote operations on *values* (take the sum of x and 2 and use the sum for further calculations) the functions `ReplaceAll` (term substitution) and `D` (derivation) are operations that transform *terms*. Mixing them with normal operations makes troubles.

Better alternative:

Use pure functions whereever possible, avoid term transformators.

Compare with functional approach as in `Haskell`:

- Consider derivation as function, that is
  ```
  derive :: (a -> a) -> (a -> a)
  ```
- Use the function `\x -> log x` or simply `log` as argument for `derive`.
  ```
  derive (\x -> log x) 2
  derive log 2
  ```

Equivalent formulations in mathematical notation:

$$(x \mapsto \log x)'(2)$$

$$\log'(2)$$

More problems with variables

Using variable quantities rather than functions is quite common in physics. But it leads to trouble even without differential calculus.

Say you have an accelerated motion. Then at every time t the way can be computed by

$$s = \frac{1}{2}at^2$$

and the velocity is

$$v = at$$

thus one can substitute at by v in the first equation, yielding

$$s = \frac{1}{2}vt$$

. But this look like an unaccelerated motion. What went wrong?

The values of s and v aren't static, they depend on t. Even more they depend uniquely on t so it is a good idea to declare s and v as functions rather than variables.

$$\begin{aligned} s(t) &= \frac{1}{2}at^2 \\ v(t) &= at \end{aligned}$$

Now the substitution looks no longer contradictory:

$$s(t) = \frac{1}{2}v(t)t$$

. Nevertheless we should note that we derived this property for uniformly accelerated motion and it might be wrong for other kinds of motion.

From programming languages to mathematical notation

Thus we should ask the other way round:

What can mathematics learn from computer science about formalisms?

We will observe that several conceptions of programming languages like types, scopes, modularization (interfaces, identifier separation, hiding of implementation details) also apply to mathematical notation.

Basics of mathematics and computer science

Similarities:

Mathematics	Computer science
Axiomatics	Data type
Model	Data structure
Sets	Bits

Ambiguity - One symbol, different meanings

Axioms tell you what properties certain objects should have. They don't tell you how these objects may look like. There may be several models for the same axiomatics. Sometimes we use the same symbols for very different objects if they only satisfy some common axioms.

A popular representation of the natural number '2' in mathematics based on sets is $\{0, 1\}$ or fully expanded:

$$\{\{\}, \{\{\}\}\}$$

A popular representation of the natural number '2' in computer science:

$$10$$

But the fraction of value '2' is completely different.

A fraction can be considered as an equivalence class of pairs, each pair consisting of an integer and a natural number, where two pairs (a, b) and (c, d) are considered as equivalent if they fulfill $a \cdot d = c \cdot b$. With this explanation the fraction '2' is represented by

$$\{(2, 1), (4, 2), (6, 3), (8, 4), (10, 5), \cdots\}$$

where the figures denote natural numbers. Both natural numbers and pairs can be modeled with sets, too.

Realising the difference between natural numbers and fractions, we become aware that the widely accepted statement

$$\mathbb{N} \subset \mathbb{Q}$$

is wrong without further justification.

How to solve this problem?

There are many types of numbers: natural, integer, fractional, algebraic, real, complex. All of these types can be represented as complex numbers. Is it a solution to compute with complex numbers always also if only natural numbers are needed?

This is certainly not a good idea since it means that you will always need a model that covers all extensions that were made to complex numbers.

There are some operations that alter or become useless when we go from one number type to another.

- Division of integers with remainder becomes division of fractions. For fractions big parts of the number theory become pointless.
- Comparison of real numbers cannot be extended to complex numbers.
- When going to more complex types: Although distributions are an extension of functions in some sense they can't replace functions since in opposition to functions distributions can't be evaluated for some arguments.

If functions are explained as relations that are sets of pairs then it is obvious that functions are really different for different number types although we use the same symbols. E.g. the '+' for integers is very different from '+' for rationals.

Expressions using different types of numbers require even more explanation e.g.

implicit conversions:

$$2 + \frac{1}{2}$$

Programming languages and mixed number types

Example: `MatLab` stores any plain number as floating point with double precision. This shall imitate the universal meaning of mathematical symbols like '2'.

But this doesn't work! Numerical problems disallow this strategy. `MatLab`'s `1/10` is different from $\frac{1}{10}$!

The expression `length(0:1/10:0.9999999999999999)` is evaluated to 11, whereas `length((0:1:9.999999999999999)/10)` results correctly in 10!

The statement `zeros(1,10/77*77)` raises the warning

```
Warning:  Size vector should be a row vector with integer elements.
```

Suggestion: `MatLab` should distinguish strictly between floating point numbers and integers. It should provide and promote routines like

- Make an arithmetic progression consisting of n numbers starting at a with increment b.
- `linspace(a,b,n)`: Subdivide the range $[a, b]$ into n parts, where a and b are floating point numbers and n is a natural number.

Superscripts

A superscript can have many meanings: A multiplication power, a composition power, a derivative or just an index.

Notation	most often	explanation
$\sin^2$	multiplication power	$\sin^2 x = \sin x \cdot \sin x$
$\sin^{-1}$	composition power	$\sin^{-1} = \arcsin$, $\sin \circ \sin^{-1} = \mathrm{id}$
$f^{(n)}$	nth derivative	$f^{(n)} = f^{(n-1)'}$

Subscripts

Subscripts are used for different purposes as well: Indices, distinction between similar identifiers, partial derivatives.

a_{min}	element of a sequence
$a_{\min}$	more specific identifier
f_x	partial derivative

Parentheses

Parentheses have even more meanings:

$(a + b) \cdot c$	overriding precedences
(a, b), $(a, b, c, \cdots)$	a pair, tuple or sequence
(a, b)	left and right open interval between a and b
(a, b)	kind of scalar product of a and b
$f(x)$	function f evaluated for argument x
$\binom{a}{b}$	binomial coefficient

Redundancy - Different symbols, one meaning

Different names for functions

- Function
- Functional
- Operator

Different notations for inline functions

- $n \mapsto n^2$
- $(n^2)_n$
- $(n^2 : n)$
- lambda-calculus $\lambda_n n^2$

Very different notations for standard functions:

- prefix notation: $\sin$, $\cos$, $\ln$
- postfix notation: !, $'$
- infix notation: $+$, $-$, $\cdot$, $:$, $*$, $\in$, $<$, $\vee$
- overfix: $\bar{z}$, $\widehat{f}$, $\dot{x}$
- "surround"fix: $\lfloor\cdot\rfloor$, $\|\cdot\|$, $\langle\cdot,\cdot\rangle$, $\sqrt{}$
- special: ab (multiplication without dot), a^b (power with superscript), $\frac{a}{b}$ (fraction)

Advantage of common notation:

- Space saving.
- Easier to read for human.
  - Maybe this is only a result of usage?

Common mathematical notation: $a + bx + cx^2$

Pure prefix notation: $++a \cdot b\, x \cdot c\, \hat{}\, x\, 2$

Function notation: `Add (Add (a, Mul (b, x)), Mul (c, Pow (x, 2)))`

Disadvantage of common notation:

- A bunch of rules for precedence and associativity for reducing the number of parentheses, e.g. '$-$' is left associative ($a - b - c = (a - b) - c$) whereas the power is right associative ($a^{b^c} = a^{(b^c)}$).
- The order of application is not clear, e.g. $\sin x!$, $\mathcal{F} f'$.

Different notations for function evaluation:

- $f(x)$
- a_i

Different notations for asymptotic bounds:

- computer science O, Ω, Θ
- $\lesssim$, $\gtrsim$, $\sim$
- LANDAU symbols O, o

where for functions f, g the connection

$$f \in O(g) \iff f \lesssim g$$
$$f \in \Omega(g) \iff f \gtrsim g$$
$$f \in \Theta(g) \iff f \sim g$$

holds.

LANDAU **symbols**

Popular definition of the LANDAU scheme:

- "$g(x) = O(f(x))$ for $x \to x_0$"
  Holds if and only if
  $\exists C > 0 : \exists \delta > 0 : \forall x \in D \setminus \{x_0\} : |x - x_0| < \delta \Rightarrow |g(x)| \leq C \cdot |f(x)|$.
- "$g(x) = o(f(x))$ for $x \to x_0$"
  Holds if and only if
  $\forall C > 0 : \exists \delta > 0 : \forall x \in D \setminus \{x_0\} : |x - x_0| < \delta \Rightarrow |g(x)| \leq C \cdot |f(x)|$.

It is a scheme, it is not combineable, e.g. it can't be deduced immediately what $f(x) = h(x) + O(g(x))$ may mean.

Suggestion: Let's follow the definition of Θ which is a function that maps a scalar function to a set of functions. Further we eliminate the dependency on a point x_0 by requiring that functions are moved to the origin.

$$O(f) = \{g : \exists C > 0 : \exists \delta > 0 : \forall x \in D \setminus \{0\} : \|x\| < \delta \Rightarrow |g(x)| \leq C \cdot |f(x)|\}$$

$$o(f) = \{g : \forall C > 0 : \exists \delta > 0 : \forall x \in D \setminus \{0\} : \|x\| < \delta \Rightarrow |g(x)| \leq C \cdot |f(x)|\}$$

Example: Total derivative

With help of the freshly defined LANDAU symbols we can express the condition that a function g of $\mathbb{R}^n \to \mathbb{R}$ has a total derivative A at x_0. Instead of

$$g(x) = g(x_0) + A \cdot (x - x_0) + o(\|x - x_0\|) \text{ for } x \to x_0$$

or

$$g(x_0 + h) = g(x_0) + A \cdot h + o(\|h\|) \text{ for } h \to 0$$

we write

$$(h \mapsto g(x_0 + h) - g(x_0) - A \cdot h) \in o(h \mapsto \|h\|)$$

.

Example: Remainder in TAYLOR series

When a function g of $\mathbb{R} \longrightarrow \mathbb{R}$ can be successfully expanded into a TAYLOR series we used to write

$$g(x_0 + h) = \sum_{j=0}^{k} \frac{g^{(j)}(x_0)}{j!} \cdot h^j + O\left(h^{j+1}\right)$$

but with the new definition of O we obtain

$$\left(h \mapsto g(x_0 + h) - \sum_{j=0}^{k} \frac{g^{(j)}(x_0)}{j!} \cdot h^j\right) \in O\left(h \mapsto h^{j+1}\right)$$

.

Defining functions

A fixed scheme for defining a function is

$$\begin{aligned} f : A &\to B \\ x &\mapsto x^2, \end{aligned}$$

where A is the domain, B the range, x a general argument and x^2 its image under the mapping f.

This scheme does not allow for combination. Can it be compared with the notation $f \in L_2(\mathbb{R})$?

Suggestion: Consider $A \to B$ as the set of all functions mapping from A to B. Consider $x \mapsto x^2$ as a general description that has to be interpreted in the context of the given domain A and range B. This is as ambiguous as the symbols 2 or $+$ are.

$$f \in A \to B$$
$$f = x \mapsto x^2$$

Now you can state things like

$$A \to B \subset \mathcal{P}(A \times B)$$
$$A \to (B \to C) \sim (A \times B) \to C$$
$$A \to B \sim B^A$$
$$L_2(\mathbb{R}) \subset \mathbb{R} \to \mathbb{R}$$

Common mathematical operations through functional glasses

Determinant	$\det$	$\in (\mathbb{R}^n)^n \to \mathbb{R}$
Expected value	E	$\in (\mathbb{R} \to [0,1]) \to \mathbb{R}$
Variance	D	$\in (\mathbb{R} \to [0,1]) \to \mathbb{R}$
Minimum	$\min$	$\in \mathcal{P}(A) \to A$
Minimizing argument	argmin	$\in (A \to B) \to A$
Limit of a sequence	$\lim$	$\in (\mathbb{N} \to A) \to A$

Derivation	$'$	$\in (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$
Integration	$\int$	$\in \mathbb{R} \to (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$
Gradient	∇	$\in (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^n)$
LAPLACE	Δ	$\in (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$
Convolution	$*$	$\in (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$
Norm	$\|\cdot\|$	$\in (\mathbb{R}^n \to \mathbb{R}) \to \mathbb{R}$
Support	supp	$\in (\mathbb{R}^n \to \mathbb{R}) \to \mathcal{P}(\mathbb{R}^n)$
Scalar product	$\langle \cdot, \cdot \rangle$	$\in (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R}) \to \mathbb{R}$
Fourier transform	$\mathcal{F}$	$\in (\mathbb{R}^n \to \mathbb{C}) \to (\mathbb{R}^n \to \mathbb{C})$
Asymptotic bound	O	$\in (\mathbb{R} \to \mathbb{R}) \to \mathcal{P}(\mathbb{R} \to \mathbb{R})$
Asymptotic comparison	$\lesssim$	$\in (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R}) \to \mathrm{Bool}$

Common abuse of notation

The following slides will point to some notations that are common but wrong.

Functions versus function values

The expressions in the right column are most oftenly used although something different is meant.

right	most oftenly wrong
$f \in L(\mathbb{R})$	$f(x) \in L(\mathbb{R})$
$f'(x)$	$f(x)'$
$O\left(n \mapsto n^2\right)$	$O(n)$
$x \mapsto x^2 \lesssim x \mapsto x^3$	$x^2 \lesssim x^3$

These mistakes occur for almost every function on functions, like scalar product, norm, convolution, support, fourier transform.

Sometimes a dot is used to mark a variable of an expression as *active* to turn this expression into a function, e.g. $f(x, \cdot)$ for $y \mapsto f(x, y)$.

This is a bad solution because this does not allow for multiple variables and for functions with one argument but constant value, e.g.

- $(x, y) \mapsto x^2 + y^2$
- $x \mapsto 2$

Even worse, there is the ambiguity that it is not known what levels of the expression are affected by the function construction. So, what means $f(g(\cdot))$? Does it mean

- $f(x \mapsto g(x))$ which is equivalent with $f(g)$ or
- $x \mapsto f(g(x))$?

Some people even prefer the cumbersome $f(\cdot)$ over f!

$f(x) \equiv 0$ should be better $f = x \mapsto 0$.

Functions varying with their argument

Some people even use different argument variables to denote different functions, i.e. the f in $f(a)$ is different from the f in $f(b)$. Thus it may be possible to have $a = b$ but $f(a) \neq f(b)$.

See [**?**], Section 2.4

Algebraic extensions

If $(K, +, \cdot)$ is a ring, then a ring extension of K by an element x is denoted by $K[x]$ and defined as:

$$K[x] = \left\{ \sum_{i=0}^{n} p_i \cdot x^i : n \in \mathbb{N} \wedge p \in K^n \right\}.$$

Thus $K[x]$ is the ring closure of $K \cup \{x\}$.

The common abuse is to treat $K[x]$ as the ring of polynomials.

Strictly spoken, a polynomial p of degree n is an $n+1$-tuple $(p_0, \ldots, p_n)$. The $+$ is defined as element-wise sum and the $\cdot$ as convolution. There is an application homomorphism φ which maps a polynomial to a function. Thus $\varphi(p)$ is a function and $\varphi(p)(x)$ is a scalar value. The latter one is commonly abbreviated to $p(x)$. So in contrast to the elements of $K[x]$ no polynomial of the polynomial ring with respect to K is connected to any x.

Compromise: "p is a polynomial for which $p(x) \in K[x]$ holds."

Minimum

$$\min_{x \in \mathbb{R}} \{f(x)\}$$

cannot work, because $\{f(x)\}$ is a (parametrized) set with a single element and there is no commonly accepted ordering for sets. Commonly accepted is

$$\min_{x \in \mathbb{R}} f(x)$$

and purely functional is

$$\min \{f(x) : x \in \mathbb{R}\}$$

and even point-free is

$$\min(\mathrm{map}(f, \mathbb{R}))$$

.

Minimizing argument

$$\operatorname{argmin} \{f(x) : x \in \mathbb{R}\}$$

cannot work, because the set $\{f(x) : x \in \mathbb{R}\}$ contains only function values of f but no information about the corresponding function arguments. Fine is

$$\operatorname*{argmin}_{x \in \mathbb{R}} f(x)$$

or purely functional

$$\operatorname*{argmin}_{\mathbb{R}} f$$

.

Binary operations

$$A \dot\vee B \dot\vee C$$

$A \dot\vee B$ means ”either A or B”. But $A \dot\vee B \dot\vee C$ does not mean ”either A or B or C”.

Counterexample: Let A, B, C be true. Then $A \dot\vee B$ is false and thus $(A \dot\vee B) \dot\vee C$ is true. Whereas ”Either A or B or C” is false.

$$a < b < c$$

The expression $a < b < c$ is nonsense.

The expression $a < b$ denotes a predicate, thus its value is a logical value. How to compare the logical value $(a < b)$ with the number c ?

Alternative: $b \in (a, c)$

Lift scalar operations to functions and sets

If f and g are functions then the expression $f + g$ is interpreted as a function with

$$\forall t : (f + g)(t) = f(t) + g(t),$$

if f is a function and y a scalar then $f + y$ is interpreted as

$$\forall t : (f + y)(t) = f(t) + y,$$

similarly if A and B are sets then $A + B$ is interpreted as

$$A + B = \{a + b : (a, b) \in A \times B\},$$

if A is a set and b is a scalar then $A + b$ is interpreted as

$$A + b = \{a + b : a \in A\}.$$

Lift functions on scalars to functions on sets

If f is a function that maps elements of A to elements of B

$$f \in A \to B$$

it is common to interpret f also as function that maps subsets of A to subsets of B

$$f \in \mathcal{P}(A) \to \mathcal{P}(B)$$

with

$$f(A') = \{f(t) : t \in A'\}$$

.

Example:

t	$f(t)$
$\emptyset$	$\emptyset$
$\{\emptyset\}$	$\emptyset$

Then because of the first interpretation it is

$$f(\{\emptyset\}) = \emptyset$$

and because of the second interpretation it is

$$\begin{aligned} f(\{\emptyset\}) &= \{f(t) : t \in \{\emptyset\}\} \\ &= \{\emptyset\} \end{aligned}$$

Better: Lift with a function like `Haskell`'s map.

$$\begin{aligned} \mathrm{map} &\in (A \to B) \to (\mathcal{P}(A) \to \mathcal{P}(B)) \\ \mathrm{map}(f, A') &= \{f(t) : t \in A'\} \end{aligned}$$

Scopes

The following expression for partial sums

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

is certainly wrong without knowing details.

Reason: k is not *visible* outside the sum.

And so the commonly used notation

$$F(x) = \int f(x) \, \mathrm{d}\, x$$

is wrong, too.

You find it ok, though?

Then you will also like:

$$F(2) = \int f(2)\,\mathrm{d}\,2$$

Suggestions: To get rid of undetermined constants and prevent a set valued integration operator fix the lower integration limit.

$$F(x_1) = \int_{x_0}^{x_1} f(x)\,\mathrm{d}\,x$$

$$F = \int_{x_0} f(x)\,\mathrm{d}\,x$$

$$F = \int_{x_0} f$$

Commas

What means $0 \leq x, y \leq 1$?

- $0 \leq x \wedge y \leq 1$ or
- $\{x, y\} \subset [0, 1]$

Does $0 < x, y \in \mathbb{R}$ mean

- $0 \leq x \wedge y \in \mathbb{R}$ or
- $\{x, y\} \subset \mathbb{R}_+$

?

Whatever you mean, better write it unambiguous!

$$x, y \in \mathbb{R}$$

seems to be unambiguous - no reason to write it correct?

What about

- $x \in \mathbb{R} \wedge y \in \mathbb{R}$
- $\{x, y\} \subset \mathbb{R}$
- $(x, y) \in \mathbb{R}^2$

?

Meaning of quantifiers

Quantifiers can be considered as shortcuts for multiple logical operations:

$$A(0) \wedge A(1) \wedge A(2) \wedge \ldots \quad - \quad \bigwedge_{j \in \mathbb{N}} A(j) \quad - \quad \forall j \in \mathbb{N} : A(j)$$

$$A(0) \vee A(1) \vee A(2) \vee \ldots \quad - \quad \bigvee_{j \in \mathbb{N}} A(j) \quad - \quad \exists j \in \mathbb{N} : A(j)$$

Analogy: $\sum$

Thus

$$A(n) \quad \forall n$$

is as wrong as

$$f(n) \prod_n$$

is!

Surprisingly I have never seen "$A(n) \quad \exists n$" and I have seldom seen a "$\bigwedge$" but never behind the quantified expression!

Calculating with quantifiers

The key point is: ”$\forall$” is **not** a shortcut for ”for all” but it is a mathematical sign!

Thus you can calculate with it as DEMORGAN's law may show:

$$\neg \bigwedge_{i \in I} A(i) \quad \Longleftrightarrow \quad \bigvee_{i \in I} \neg A(i)$$

$$\neg \bigvee_{i \in I} A(i) \quad \Longleftrightarrow \quad \bigwedge_{i \in I} \neg A(i)$$

Quantifiers: continuity vs. homogenous continuity

An example from a basic lecture about calculus: A function f is called *continuous* if and only if holds

$$\forall \varepsilon > 0 : \exists \delta : \forall y : |x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon \qquad \forall x.$$

Knowing that this notation is nonsense we wonder what it may mean.

Does it mean normal continuity, that is

$$\forall x : \forall \varepsilon > 0 : \exists \delta : \forall y : |x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon$$

or does it mean

$$\forall \varepsilon > 0 : \exists \delta : \forall x, y : |x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon$$

which is uniform continuity?

(Evil example is the C programming language where `int *d[10];` is used for ”array of 10 pointers, each pointing to an integer” instead of `int * [10] d;` or `d [10] * int;`.)

Omitted quantifiers

The informal statement

$$A \cdot \|f\| < \|Tf\| < B \cdot \|f\| \quad A, B \in \mathbb{R}, f \in L_2(\mathbb{R})$$

can easily be turned into strict logic:

$$\exists \{A, B\} \subset \mathbb{R} : \forall f \in L_2(\mathbb{R}) : A \cdot \|f\| < \|Tf\| \wedge \|Tf\| < B \cdot \|f\| .$$

Negation required?

No problem:

$$\forall \{A, B\} \subset \mathbb{R} : \exists f \in L_2(\mathbb{R}) : \|Tf\| \leq A \cdot \|f\| \vee B \cdot \|f\| \leq \|Tf\|$$

Hidden quantifiers

What's the difference between

$$0 = x^2 + 2x + 1$$

and

$$y'(x) = x + \sin x$$

?

$0 = x^2 + 2x + 1$ means $0 = x^2 + 2x + 1$

$y'(x) = x + \sin x$ means $\forall x : y'(x) = x + \sin x$

This explains why the following trick doesn't work:

$$\ln x = x + 2 \quad \Big| \quad \frac{\mathrm{d}}{\mathrm{d}\,x}$$

$$\frac{1}{x} = 1$$

$$x = 1$$

Sets, multi-sets, sequences

Parentheses $(,)$ construct a sequence.

Curly braces $\{, \}$ denote a set.

Thus $(1, 1, 1, \ldots)$ is a sequence with the accumulation point 1.

In contrast $\{1, 1, 1, \ldots\}$ is a set containing only one element and has no accumulation point at all.

Multi-sets are unordered collections of objects like sets but in contrast to sets they can contain each object multiple times. Sometimes multi-sets are denoted by brackets, e.g. $[1, 1, 1, \ldots]$.

Sets, multi-sets, sequences as functions

set A	$f \in A \to \mathrm{Bool}$	$f(x)$ is true if x is an element of A
multi-set of A	$f \in A \to \mathbb{N}_0$	$f(x)$ evaluates the multiplicity of x in the multi-set
sequence over A	$f \in \mathbb{N}_0 \to A$	$f(n)$ returns the element at the nth position in the sequence

Sets, multi-sets, sequences and vector bases

B is called a *basis* of the vector space V if it is a *set* of vectors that are linearly independent and V is the linear closure of B.

Thus $\{v, w, 2v - w\}$ can never be a basis because the vectors are always linearly dependent.

Wrong!

If $v = w$ then $\{v, w, 2v - w\}$ is a set that consists of one element only. The set "collapses" to $\{v\}$. This is a linear independent set and is a basis of the vector space $\{\lambda \cdot v : \lambda \in \mathbb{R}\}$.

Problem: Sets can contain every object at most once!

Solution: Define a basis as multi-set or even better as function that maps an index to a basis vector.

Various flaws

- $\{x_2 > 0\}$ instead of $\{(x_1, x_2) : x_2 > 0\}$
- $\{u = r\}$ instead of $\{x : u(x) = r\}$

Typographic flaws

Denote a variable with n instead of n.

$$\frac{\mathrm{d}}{\mathrm{d}\,t} tan\ x = an\ x$$

$$\gcd(a,b) = \prod_{p\in\mathbb{P}} p^{min(\mathrm{pexp}(p,a),\mathrm{pexp}(p,b))}$$

$$\stackrel{?}{=} a^{min}, b^{min}$$

$$\frac{dadb}{a^2} = \frac{d^2b}{a}$$

[LMR97]

References

[LMR97] Alfred Karl Louis, Peter Maass, and Andreas Rieder. *Wavelets: Theory and Application*. Wiley, Chichester, 1997.

Problems when using mathematical notation in computer algebra: `http://www.stephenwolfram.com/publ`

History of some symbols: `http://page.mi.fu-berlin.de/ froetsch/manosem/`

More history: `http://www.roma.unisa.edu.au/07305/symbols.htm`

Mathematical notation and the programming language J: `http://www.jsoftware.com/books/help/camn/c`

Florian Cajori: A History of Mathematical Notations

Horst Hischer: Zur Geschichte des Funktionsbegriffs `http://hischer.de/uds/forsch/preprints/hische`