Mathematics of Proof Assistant

Dedicated to the memory of Professor Shigeru Igarashi

Masahiko Sato
Graduate School of Informatics, Kyoto University

IFIP WG 2.2 Meeting
RWTH Aachen University
September 26, 2025

Concepts

The title of IFIP WG 2.2 is

Formal description of Programming Concepts

The main aim of a proof assistant is

Concepts

The title of IFIP WG 2.2 is

Formal description of Programming Concepts

The main aim of a proof assistant is

Formal description of Mathematical Concepts

Concepts

The title of IFIP WG 2.2 is

Formal description of Programming Concepts

The main aim of a proof assistant is

Formal description of Mathematical Concepts

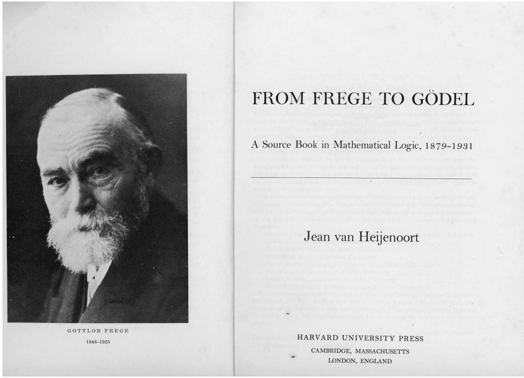
If the above claim is correct, why not take the notion of concept itself as the key notion of a proof assistant.

We are designing such a proof assitant called HyperMath. In this talk, we give a brief account of HyperMath.

We will see that neither the notion of type nor set can replace the notion of concept.

Gottlob Frege (1848 – 1925)

Frege published his first book *Begriffsscript (conceptual notation)* in 1879. In the book, he introduced the symbol $\vdash$.



{1967, Harvard University Press}

Mathematics is an Activity

Mathematics is an Activity

Mathematics is a human, social, and linguistic activity.

The core of the activity consists of defining mathematical concepts and mathematical objects, and proving theorems.

Mathematics is a language, and it is inheriently open-ended and practiced through communiactions among people interested in mathematics.

However, basic concepts are often introduced not by definitions but by axioms.

Essence of Mathematics

1. Mathematics is linguistic activity
2. Mathematical objects are expressed by (symbolic) expressions
3. Any expression can be identified with a finite sequence of bits (0 and 1)
4. Mathematics is an open system

I would like to replace above items by:

Essence of Mathematics

1. Mathematics is linguistic activity
2. Mathematical objects are expressed by (symbolic) expressions
3. Any expression can be identified with a finite sequence of bits (0 and 1)
4. Mathematics is an open system

I would like to replace above items by:

1. Mathematics is linguistic activity
2. Mathematical objects are (symbolic) expressions
3. Any expression can be identified with a finite sequence of bits (0 and 1)
4. Mathematics is an open system

Proof Assistant

A proof assistant is a software which supports development of formal proofs by human-computer interaction.

I am currently implementing a new proof assistant which I named ***HyperMath***.

In this talk, I will show examples of ***HM*** code, which is the kernel language supporting ***HyperMath***.

Structure of the *HyperMath* System

The ***HyperMath*** system consists of the six layers below.

Proof
Logic
Computation
Expression
HyperMath kernel
The Internet

The Internet

Proof
Logic
Computation
Expression
HyperMath kernel
The Internet

The bottom layer is the Internet.

The most important feature of the Internet is that any two computers connected by the Internet can exchange any sequence of bits (0 or 1) using the TCP/IP protocol provided by the Internet.

HyperMath kernel

Proof
Logic
Computation
Expression
HyperMath kernel
The Internet

On top of the Internet layer, we will implement the ***HyperMath*** kernel as a Turing-complete and open-ended language which will always be extended and distributed over the Internet.

Symbolic expression

We begin with defining the concept symbolic expression which shows the syntactic and gramatical structure of the laguage ***HM***.

```
SExp :=: symbolic_expression

DefCon SExp
    (u) :=: u
    (s t u) :=: (s (t u))

    ()     : SExp
    (s t) : SExp :- s : SExp, t : SExp
end
```

Expression

```
DefCon Exp
    Exp ::= (::=) | (|) | (...) | :=: | : | :- | , | <: |
         |  SExp | symbolic_expression | Exp | expression
         |  DefCon | end |
...
```

The keyword ... above means that the definition of expression is not yet completed, but all the symbolic expressions listed are expressions.

So, we see that expression is an open concept.

Here, we omit the definition of $\boldsymbol{x} = \boldsymbol{y}\ (\mathtt{Exp})$, but is defined as syntactic identity modulo $\mathbf{:=:}$ Hence, the equality is decidable in ***HM***.

Object

We define object as follows.

```
DefCon Obj :=: object
    (a A) : Obj :- a : Exp, A : Con, a : A
    (a A) = (b B) :- A <: C, B <: C, a = b (C)
end
```

Equality of two objects $(a\ A)$ and $(b\ B)$ is defined by the equality $a = b\ (C)$, where C is a common super-concept of A and B.

The symbol `<:` means the sub-concept relation.

Pair

```
DefCon Pair :=: pair
     (a b) : Pair :- a : Obj, b : Obj
     (a b) = (c d) :- a = c (Obj) , b = d (Obj)
end
```

Empty

The context empty has no members, and we define it as follows.

```
DefCon () :=: {} :=: 0 :=: zero :=: empty
end
```

Since empty has no rules in its defintion, empty has no members. We use (), {}, 0, and zero as aliases of empty.

One, Two, Three, ...

The context one has exactly one member, so we define one as follows.

```
DefCon 1 :=: one
  0 : 1 :-
  0 = 0 :-
end
```

We define two, three, etc similarly as needed.

```
DefCon 2 :=: two
  0 : 2 :-
  1 : 2 :-
  0 = 0 :-
  1 = 1 :-
end
```

List

We define list as follows.

```
DefCon List
       () : List   :-
      a L : List   :- a : Obj, L : List
       () = ()     :-
    (a L) = (b M)  :- a = b (Obj), L = M
end
```

For example:

```
a b c () : List :- a : Obj,
  b c () : List :- b : Obj,
    c () : List :- c : obj,
      () : List :-
```

List of a concept

Given a concept $\boldsymbol{C}$, we can define the concept list of $\boldsymbol{C}$ as follows.

```
DefCon List(C) :=: list_of_C <: List
     () : List(C) :- C : Con
    a L : List(C) :- a : C, L : List(C)
end
```

Since List($\boldsymbol{C}$) is a sub-concept of List, equality on list($\boldsymbol{C}$) is inherited from List. Hence, definion of equality is missing in the above definition.

It is important to remark that the concept List is prior to the concept list of ($\boldsymbol{C}$) for any concept $\boldsymbol{C}$ other than Obj.

This is the reason why systems of dependent type theory have become so complex.

Natural number

Here, we define natural number as list of One.

```
Nat :=: natural_number :=: List(One)
```

For example, we have:

```
0 0 0 0 () : Nat :- 0 : One,
  0 0 0 () : Nat :- 0 : One,
    0 0 () : Nat :- 0 : One,
      0 () : Nat :- 0 : One,
        () : Nat :-
```