Truth by Evidence

Masahiko Sato
Graduate School of Informatics
Kyoto University

Quiz

Mathematicians study mathematics.

Logicians study what?

Quiz (cont.)

Mathematicians study mathematics.

Logicians study metamathematics.

Computer scientists study what?

Quiz (cont.)

Mathematicians study mathematics.

Logicians study metamathematics.

Computer scientists also study metamathematics.

Observations and a Question

Mathematicians study mathematical objects such as numbers, rings etc.

Logicians study metamathematical objects such as terms, formulas, proofs etc. from logical point of view.

Computer scientists also study metamathematical objects such as terms, formulas, proofs etc. but mainly from computational point of view.

Since metamathematics is also mathematics, metamathematical objects are mathematical objects.

But, what is a mathematical object?

My Answer

My answer is that a mathematical object is a symbolic expression sitting in the universe of expressions equipped with equality relations among expressions.

- The universe is open ended.
- There are two equality relations: intensional equality and extensional equality.

Intensional equality is computational equality and can be mechanically checked by computation. Extensional equality is logical equality and can be checked by reasoning.

Motivation

We wish to create a computer environment for doing mathematics in it.

Doing mathematics means computing and proving.

Ideas

Computation and Logic are closely related, so they should be studied together rather than separately.

At Kyoto University we offer an under graduate course on Computation and Logic, where students can try all the materials covered by the course on computers.

We provide a computer environment called CAL for the above purpose. CAL is implemented in Emacs Lisp and can be run within an Emacs buffer.

What are common between Computation and Logic?

- The notion of variable plays the crucial role.
- The notions of judgment and derivation plays the crucial role.
- They both manipulate symbols and they are both formalisable.
- There are structural isomorphisms called the Curry-Howard isomorphisms between certain computational structures and logical structures.

It is therefore more efficient to study them at the same time rather than separately!

Formal vs. Informal

A mathematical *entity* is formal if it can be implemented on a computer. A mathemaical *notion* is formal if it can be defined within a system implemented on a computer.

For example, natural numbers 0, 1, 2 etc. are formal entities and the notions of natural number, even natural numbers and prime natural numbers are formal notions.

Hilbert's formalism claims that all the mathematics are formalizable.

Formal vs. Informal (cont.)

Informal: If $x = y$, then $y = x$.

Semi-formal: $x = y \Rightarrow y = x$.

Formal: $\mathtt{x} = \mathtt{y} \supset \mathtt{y} = \mathtt{x}$.

In the beginning, we must go from informal to formal, but once a certain amount of formal objects and notions are established, we can go from formal to informal.

Natural Framework

It is therefore important to have a computer environment which supports formalisation of mathematics, including computation and logic.

We have designed and implemented such a framework which we call Natural Framework (NF).

Concepts and Objects (Derivation Games)
Judgments and Derivations (NF)
Expressions (CAL)
Symbolic Expressions (Emacs Lisp)

(Structure of NF)

Judgments and Derivations

We think that the two most fundamental notions in mathematics are judgment and derivation, since a mathematical *statement* is expressed as a judgment and its truth is established by means of a *proof* which may also be called a derivation.

For example, a theorem is a judgment which has a derivation.

Now, in a derivation of a mathematical judgment, both logic and computation play essential roles.

So, we may assert the following:

$$\begin{aligned} & \text{Computation and Logic} \\ = \; & \text{Science of Judgment and Derivation} \end{aligned}$$

Developing meaningful mathematics in NF

In order to develop mathematics formally and at the same time intuitionistically meaningful way, we introduce a meaning theory of judgments as a refinement of Brouwer-Heyting-Kolmogorov interpretation.

According to BHK interpretation, the meaning of a judgment (proposition) is given by defining what is a construction (which is an abstract counterpart of proof) of it. For example, a construtiom of a judgment $A \supset B$ is a method (or a function) f, which for any construction a of A yields a construction of B by applying the method f to a, that is, $f(a)$ gives a construction of B.

Developing meaningful mathematics in NF (cont.)

We modify BHK interpretaion by using our basic principle that a mathematical object is an expression. Thus, for example, a function is just a program which describes methods in a concrete way.

We design our theory of expressions so that we can represent different syntactic categories of judgments and objects naturally.

Meaning of a judgement expression will be given in terms of an evidence which is also an expression, and meaning of an object expression is given by its denotation which is also an expression.

Theory of Expressions

Requirements for the theory.

- Can be used as a common notation system for various formal languages.
- Supports higher order abstract syntax.
- Must be simple enough.

Theory of Expressions (cont.)

We define an *arity* as an expression of the form:

$$\kappa[\kappa_1, \ldots, \kappa_n]$$

where κ and κ_i are either o (objects) or j (judgments). When $n = 0$, we will simply write κ for $\kappa[]$

An arity is *saturated* (*unstaturated*) if $n = 0$ ($n > 0$).

Unsaturated variables (constants) are also called *higher order* variables (constants) since they have n argument places to be filled in by expressions.

Saturated variables (constants) are also called *first order* variable (constants).

Theory of Expressions (cont.)

Expressions will be built up by combining variables and constants appropriately. With each variable or a constant we associate a unique arity.

We write

$$x \,@\, \alpha \;\; (c \,@\, \alpha)$$

if a variable x or a constant c has arity α.

Theory of Expressions (cont.)

Expressions are defined as follows. We say that an expression is an *object expression* (*judgment expression*) if we have $e : o$ ($e : j$) by the following rules.

$$\frac{x \mathrel{@} \kappa[\kappa_1, \ldots, \kappa_n] \quad a_1 : \kappa_1 \quad \cdots \quad a_n : \kappa_n}{x\,[a_1, \ldots, a_n] : \kappa}\;\mathsf{var}$$

$$\frac{c \mathrel{@} \kappa[\kappa_1, \ldots, \kappa_n] \quad a_1 : \kappa_1 \quad \cdots \quad a_n : \kappa_n}{c\,[a_1, \ldots, a_n] : \kappa}\;\mathsf{const}$$

$$\frac{x \mathrel{@} \alpha \quad a : \kappa}{(x)\,[a] : \kappa}\;\mathsf{abs} \qquad \frac{x \mathrel{@} o \quad a : j \quad b : \kappa}{(x :: a)\,[b] : \kappa}\;\mathsf{cabs}$$

Environments

We will evaluate expressions in environments.

If x is a variable of arity $\kappa[\kappa_1, \ldots, \kappa_n]$ and e is an expression of the form $(y_1, \ldots, y_n)\,[a]$ where $y_i \,@\, \kappa_i$ and $a : \kappa$, then we say that e is *assignable* to x and call the form:

$$x = e$$

a *definition*.

$$\rho = \{x_1 = e_1, \ldots, x_k = e_k\}$$

is an *environment* if $x_1, \ldots, x_k$ are distinct variables, and its *domain* $|\rho|$ is $\{x_1, \ldots, x_k\}$.

Instantiation

Given an expression e and an environment ρ, we define an expression $[e]_\rho$ as follows. We choose fresh local variables as necessary.

1. $[x]_\rho := e$ if x is *first order* and $x = e \in \rho$.
2. $[x[a_1, \ldots, a_n]]_\rho := [e]_{\{x_1 = [a_1]_\rho, \ldots, x_n = [a_n]_\rho\}}$
   if x is *higher order* and $x = (x_1, \ldots, x_n)\,[e] \in \rho$.
3. $[x[a_1, \ldots, a_n]]_\rho := x[[a_1]_\rho, \ldots, [a_n]_\rho]$ if $x \notin |\rho|$.
4. $[c[a_1, \ldots, a_n]]_\rho := c[[a_1]_\rho, \ldots, [a_n]_\rho]$.
5. $[(x)\,[a]]_\rho := (x)\,[[a]_\rho]$.
6. $[(x :: a)\,[b]]_\rho := (x :: [a]_\rho)\,[[b]_\rho]$.

Instantiation (cont.)

Well-definedness

An environment ρ is *first order* if all the variables in $|\rho|$ are first order, and it is *higher order* if $|\rho|$ contains at least one higher order variable.

1. First, carry out the above inductive definition for first order environments.

2. Then, carry out the above inductive definition for higher order environments.

Instantiation (cont.)

Remark.

It is essential to distinguish first order variables and higher order variables.

Without the distinction, evaluation of expressions may fail to terminate.

$$
\begin{aligned}
& [x[x]]_{\{x=(y)\,[y[y]]\,\}} \\
\equiv\; & [y[y]]_{\{y=[x]_{\{x=(y)\,[y[y]]\,\}}\}} \\
\equiv\; & [y[y]]_{\{y=(y)\,[y[y]]\,\}} \\
\equiv\; & \cdots
\end{aligned}
$$

Instantiation (cont.)

Remark.

It is essential to distinguish first order variables and higher order variables.

Without the distinction, evaluation of expressions may fail to terminate.

$$
\begin{aligned}
& [x[x]]_{\{x=(y)\,[y[y]]\}} \\
\equiv\;& [y[y]]_{\{y=[x]_{\{x=(y)\,[y[y]]\}}\}} \\
\equiv\;& [y[y]]_{\{y=(y)\,[y[y]]\}} \\
\equiv\;& \cdots
\end{aligned}
$$

x is saturated, but x is unsaturated.

Developing mathematics in NF

Mathematics in NF is open ended in the sense that one can always extend it by introducing new notions and objects. Introduction of new notions and objects is done in two steps.

The first step is syntactical and we add new constants for new notions/objects. Expressions are then automatically extended.

The second step is semantical and we assign meaning to newly created expressions. We will assign meaning to object expressions and judgment expressions.

In the third step we add inference rules for deriving newly created judgments.

Assigning meaning to object expressions

We assign meaning to object expressions by the evaluation relation

$$e \Downarrow v$$

where e and v are both object expressions.

We say that e has *value* v if the above relation holds.

An object expression v is said to be a *value expression* (or, simply *value*) if some expression e has value v.

The evaluation relation should be defined in such a wary that the following holds.

- If $e \Downarrow v_1$ and $e \Downarrow v_2$, then v_1 and v_2 are the same expression.
- If v is a value expression, then $v \Downarrow v$.

An expression may not have a value.

Assigning meaning to Judgments

We can classify judgments into the following three forms.

1. *Universal Judgment*: $(x)[J]$.
2. *Conditional Judgment*: $(x :: H)[J]$.
3. *Basic Judgment*: $c[a_1, \ldots, a_n]$ where $c \,@\, j[\kappa_1, \ldots, \kappa_n]$ and $a_i : \kappa_i$.

Meaning of a judgement is determined by the relation:

$$e :: J$$

which we read "e is an evidence of J", where e is an object expression and J is a judgment.

Assigning meaning to Judgments (cont.)

The following rule defines the evidence relation for universal judgments.

> If $e \Downarrow (x)\,[d]$ and $[d]_{\{x=a\}} :: [J]_{\{x=a\}}$ for all a which is assignable to x, then $e :: (x)\,[J]$.

The meaning of conditional judgments is given by the following rule.

> If $e \Downarrow (x)\,[d]$ and $[d]_{\{x=a\}} :: [J]_{\{x=a\}}$ for all a such that $a :: H$ then $e :: (x :: H)\,[J]$.

We must also add rules which define meaning of new judgment constants.

We will call these rules *meta rules*. Meta rules are all introduction rules if we borrow the terminology of natural deduction style type/logical systems.

Getting started

We have described general schema of assigning meaning to expressions, but we did it concretely only for universal and conditional judgments.

Next step is to add concrete object constants and judgment constants, and give meaning to them. We add the following four object constants and two judgment contstants.

- Object constants: $\mathtt{ok} \mathbin{@} o$, $\mathtt{nil} \mathbin{@} o$, $\mathtt{cons} \mathbin{@} o[o, o]$, $\mathtt{apply} \mathbin{@} o[o, o]$.
- Judgment constants: $\Downarrow \mathbin{@} j[o, o]$, $:: \mathbin{@} j[o, j]$.

Getting started (cont.)

We will informally write $f\,(e)$, $e \Downarrow v$ and $e :: J$ as follows.

$$f\,(e) \equiv \texttt{apply}[f, e],$$

$$e \Downarrow v \equiv \Downarrow[e, v],$$

$$e :: J \equiv ::[e, J].$$

`nil` and **`cons`** are used to construct list of objects. So, for example, (a, b) stands for $\texttt{cons}[a, \texttt{cons}[b, \texttt{nil}]]$.

Rules for evaluating object expressions

The meta rules for evaluating object expressions we now have are as follows.

$$\frac{}{\texttt{ok} \Downarrow \texttt{ok}}$$

$$\frac{}{\texttt{nil} \Downarrow \texttt{nil}} \qquad \frac{a \Downarrow u \quad b \Downarrow v}{\texttt{cons}\,\texttt{[}a, b\texttt{]} \Downarrow \texttt{cons}\,\texttt{[}u, v\texttt{]}}$$

$$\frac{}{\texttt{(}x\texttt{)}\,\texttt{[}F\texttt{]} \Downarrow \texttt{(}x\texttt{)}\,\texttt{[}F\texttt{]}} \qquad \frac{}{\texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]} \Downarrow \texttt{(}x\texttt{)}\,\texttt{[}F\texttt{]}}$$

$$\frac{f \Downarrow \texttt{(}x\texttt{)}\,\texttt{[}F\texttt{]} \qquad [F]_{\{x=e\}} \Downarrow v}{f\,\texttt{(}e\texttt{)} \Downarrow v}$$

Rules for basic judgments

Meta rules for newly added basic judgments are as follows.

$$\frac{d \Downarrow \texttt{ok} \quad e \Downarrow v}{d :: e \Downarrow v}$$

$$\frac{d \Downarrow v \quad e \Downarrow v \quad e :: J}{d :: e :: J}$$

Note. Since $\Downarrow$ has arity $j[o, o]$ and $::$ has arity $j[o, j]$, $d :: e \Downarrow v$ cannot be parsed as $(d :: e) \Downarrow v$. Therefore we have $d :: e \Downarrow v \equiv d :: (e \Downarrow v)$. Similarly, we have $d :: e :: J \equiv d :: (e :: J)$.

Declaration and Context

Any mathematical argument is done in a context, namely, we assume certain amount of assumptions (hypotheses) whenever we make a mathematical argument.

In NF, a *context* is a sequence of declarations, where a *declaration* is either a expression variable declaration or a derivation variable declaration.

- An *expression variable declaration* is simply a variable x and it declares that x stands for an arbitrary expression.

- A *derivation variable declaration* is of the form $X :: J$ and it declares that the variable X stands for an arbitrary derivation of the judgment J.

Formal Rules for Derivations

We now introduce formal rules for constructing *derivations*.

Any derivation d constructed by these rules will become an evidence for a judgment J and this J can be uniquely determined from d.

Any derivation d has another important property that from d one can uniquely recover how d is derived by the rules for construction derivations.

This property is crucial for commucating derivations among humans and also for mechanical checkability of the corectness of derivations.

Formal Rules for Derivations (cont.)

All the premises and conclusions of the following rules have the form:

$$\Gamma \vdash d :: J$$

where Γ is a context such that $(\Gamma)[d :: J]$ becomes a closed judgment. Actually, the above form is an abbreviation of $(\Gamma)[d :: J]$.

For example, if A is a variable of arity j and x is a variable of arity o, then

$$A, x :: A \vdash x :: A \equiv (x, x :: A)[x :: A] \equiv (x)[(x :: A)[x :: A]].$$

Formal Rules for Derivations (cont.)

Assumption

$$\frac{}{\Gamma, x :: H, \Delta \vdash x :: H}$$

Universal judgment

$$\frac{\Gamma, x \vdash d :: J}{\Gamma \vdash (x)\,[d] :: (x)\,[J]} \qquad \frac{\Gamma \vdash f :: (x)\,[J]}{\Gamma \vdash f\,(e) :: [J]_{\{x=e\}}}$$

Conditional judgment

$$\frac{\Gamma, x :: H \vdash d :: J}{\Gamma \vdash (x :: H)\,[d] :: (x :: H)\,[J]} \qquad \frac{\Gamma \vdash f :: (x :: H)\,[J] \quad \Gamma \vdash e :: H}{\Gamma \vdash f\,(e) :: [J]_{\{x=e\}}}$$

Formal Rules for Derivations (cont.)

Evaluation judgment

$$\frac{}{\Gamma \vdash \texttt{mkeval}[\texttt{ok}, \texttt{ok}, ()] :: \texttt{ok} \Downarrow \texttt{ok}}$$

$$\frac{}{\Gamma \vdash \texttt{mkeval}[\texttt{nil}, \texttt{nil}, ()] :: \texttt{nil} \Downarrow \texttt{nil}}$$

$$\frac{\Gamma \vdash d_1 :: a \Downarrow u \quad \Gamma \vdash d_2 :: b \Downarrow v}{\Gamma \vdash \texttt{mkeval}[\texttt{cons}[a, b], \texttt{cons}[u, v], (d_1, d_2)] :: \texttt{cons}[a, b] \Downarrow \texttt{cons}[u, v]}$$

$$\frac{\Gamma \vdash d :: e \Downarrow v}{\Gamma \vdash \texttt{mkeval}[\texttt{evi}[e], v, (d)] :: \texttt{evi}[e] \Downarrow v}$$

$$\frac{}{\Gamma \vdash \texttt{mkeval}[(x)[F], (x)[F], ()] :: (x)[F] \Downarrow (x)[F]}$$

$$\frac{\Gamma \vdash d_1 :: f \Downarrow \texttt{(}x\texttt{)}\,\texttt{[}F\texttt{]} \qquad \Gamma \vdash d_2 :: [F]_{\{x=e\}} \Downarrow v}{\Gamma \vdash \texttt{mkeval}\,\texttt{[}f\,\texttt{(}e\texttt{)}, v, \texttt{(}d_1, d_2\texttt{)}\texttt{]} :: f\,\texttt{(}e\texttt{)} \Downarrow v}$$

$$\frac{}{\Gamma \vdash \texttt{mkeval}\,\texttt{[}\texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]}, \texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]}, \texttt{()}\texttt{]} :: \texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]} \Downarrow \texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]}}$$

$$\frac{\Gamma \vdash d_1 :: f \Downarrow \texttt{(}x :: H\texttt{)}\,\texttt{[}F\texttt{]} \qquad \Gamma \vdash d_2 :: [F]_{\{x=e\}} \Downarrow v}{\Gamma \vdash \texttt{mkeval}\,\texttt{[}f\,\texttt{(}e\texttt{)}, v, \texttt{(}d_1, d_2\texttt{)}\texttt{]} :: f\,\texttt{(}e\texttt{)} \Downarrow v}$$

$$\frac{\Gamma \vdash d :: \texttt{eval}\,\texttt{[}e, v\texttt{]} \Downarrow w}{\Gamma \vdash \texttt{mkeval}\,\texttt{[}\texttt{mkeval}\,\texttt{[}e, v, D\texttt{]}, w, \texttt{(}d\texttt{)}\texttt{]} :: \texttt{mkeval}\,\texttt{[}e, v, D\texttt{]} \Downarrow w}$$

$$\frac{\Gamma \vdash d :: e \Downarrow v}{\Gamma \vdash \texttt{mkeval}\,\texttt{[}\texttt{mkevi}\,\texttt{[}e\texttt{]}, v, \texttt{(}d\texttt{)}\texttt{]} :: \texttt{mkevi}\,\texttt{[}e\texttt{]} \Downarrow v}$$

Evidence judgment

$$\frac{\Gamma \vdash d :: J}{\Gamma \vdash \texttt{evi}[d] :: d :: J} \qquad \frac{\Gamma \vdash e :: d :: J}{\Gamma \vdash \texttt{mkevi}[e] :: J}$$

Evaluation rules for additional object contstants

$$\frac{e \Downarrow v}{\texttt{evi}\,[e] \Downarrow v} \qquad \frac{e \Downarrow v}{\texttt{mkevi}\,[e] \Downarrow v}$$

$$\frac{}{\texttt{mkeval}\,[e, v, D] \Downarrow \texttt{ok}}$$

What we teach

- Expressions
- Derivation Games
- Formal Syntax and Definitional Equality
- Propositional Logic in Natural Deduction style (NJ)
- Propositional Logic in Hilbert's style
- Simply Typed λ-calculus
- Reduction of NJ derivations and λ-terms
- Curry-Howard Isomorphism
- Heyting Arithmetic