

Algebra and Geometry of the Lambda-Calculus

Masahiko Sato

Graduate School of Informatics, Kyoto University

MLG 61

Kobe University

October 24, 2026

We define traditional Curry-Hindley style untyped λ term by the grammar:

$$M, N ::= x \mid (M \ N) \mid \lambda x. M$$

Question: Can we define only **closed λ -terms** without using variables?

- **Algebra** = Solving equations:

$$f(\mathbf{x}) = M \Leftrightarrow f \cdot \mathbf{x} = M \Leftrightarrow f = M/\mathbf{x}.$$

- **Geometry** = Invariance of the shape under group $(\mathbb{Z})$ action +
Equivariance of the shape under group (S_n) action.

Overview

System	Name	Metavariable	Equality
Symbolic Expressions	Σ	S, T	$\equiv$
Curry's λ -calculus	λ	M, N, P	$\equiv, \equiv_{\alpha}, =$
Canonical λ -calculus	Λ	M, N, P	$\equiv, =_{\Lambda}$
Objects of ∇	Δ	M, N, P	$\equiv, =_{\Delta}$
Morphisms of ∇	∇	d, e, f, g	$\equiv, =_{\nabla}$

Curry's λ -calculus is a well-known **untyped calculus**. We add the other systems, in order to analyze λ by distinguishing **syntactic** arguments and **semantic** arguments using two distinct equalities $\equiv$ (syntactic equality) and $=$ (semantic equality), respectively.

∇ is a **category** with objects Δ and morphisms ∇ . We will call ∇ the **Contextual Lambda Calculus**,

To sum up, we study λ from the viewpoint of **Contextual Lambda Calculus**.

Syntactic Equality ($\equiv$)

System	Name	Metavariable	Equality
Symbolic Expressions	Σ	S, T	$\equiv$
Curry's λ -calculus	λ	M, N, P	$\equiv, \equiv_\alpha$
Canonical λ -calculus	Λ	M, N, P	$\equiv$
Objects of ∇	Δ	M, N, P	$\equiv$
Morphisms of ∇	∇	d, e, f, g	$\equiv$

Every mathematical objects we talk about will be defined as [symbolic expressions](#). The class of these expressions, Σ , forms a free algebra, and it has a unique equality relation on it determined by its initiality. We will write $S \equiv T$ if S and T are equal in this sense.

As shown in the above table, all the other systems will inherit this syntactic equality. They all have their own equality ($=$) as well.

Formalization of Mathematics (Computation)

Just a part of the history of [formalization of computation](#).

- Moses Schönfinkel (1888 – 1942) invented [combinators](#) and gave a talk on it in 1920. Theory of combinators is called [Combinatory Logic \(CL\)](#).
- Haskell Curry (1900 – 1982) reinvented CL (around 1927). Curry also studied the λ -calculus in his book [Combinatory Logic](#) (with Robert Feys).
- Alonzo Church (1902 – 1995) introduced the [\$\lambda\$ -calculus](#) in a paper published in 1936.
- Alan Turing (1912 – 1954) introduced the [Turing machine](#) in a paper published in 1937.
- John McCarthy (1927 – 2011) invented [Lisp programming language](#) (in late 1950's) whose domain of computation is a [free magma](#) generated from the set of atoms.

Formalization of Mathematics (Logic)

Just a part of the history of [formalization of logic](#).

- Gottlob Frege (1848 – 1925) introduced his logical system in his book *Begriffsschrift* (published in 1879).
- Gerhard Gentzen (1909 – 1945) introduced [natural deduction](#) and [sequent calculus](#) in 1935.
- Per Martin-Löf (1942 –) introduced [dependent type theory](#) in his book *Intuitionistic Type Theory* (published in 1984).

Categorical Judgment vs. Hypothetical Judgment

Frege dealt with **categorical judgments**, while both Gentzen and Martin-Löf worked on **hypothetical judgments**.

A categorical judgment $\vdash P$ is a special case of the hypothetical judgment $\Gamma \vdash P$ where the **context** (hypothesis) Γ is empty.

For example, let us consider the following inference in Martin-Löf's ITT.

$$\frac{x : A \quad \vdash b : B}{\vdash \lambda x. b : A \rightarrow B}$$

Note that the premise of this inference is a hypothetical judgment having $x : A$ as its hypothesis, while its conclusion is a canonical judgment.

In this talk, I will argue that these two judgments have the same **meaning** according to the semantics we will give later.

Confluence and variable capture

We define traditional Curry-Hindley style untyped λ term by the grammar:

$$M, N ::= x \mid (M \ N) \mid \lambda x. M$$

Let $M := (\lambda y. x \ y)$ and $N := (\lambda x. M \ y)$. Then, we have

$$N \equiv (\lambda x. (\lambda y. x \ y) \ y)$$

where $\equiv$ means syntactic identity. N has two β -redexes N and M .

Here, we have the following two β -reduction paths which are confluent.

$$N \longrightarrow_{\alpha} (\lambda x. (\lambda z. x \ y) \ y) \longrightarrow_{\beta} (\lambda z. y \ y) \longrightarrow_{\beta} y \quad (1)$$

$$N \longrightarrow_{\beta} (\lambda y. x \ y) \longrightarrow_{\beta} y \quad (2)$$

The reduction path (1) shows that α -conversion is indispensable to have the confluence.

λ -calculus vs. Combinatory Logic

The terms of λ -calculus and Combinatory Logic (CL) are defined by the following grammars:

$$M, N ::= x \mid \lambda x. M \mid (M N)$$

$$A, B ::= x \mid \mathbf{S} \mid \mathbf{K} \mid \mathbf{I} \mid (A B)$$

Some observations about λ terms and λ -calculus as equational theory.

- λ terms have both free and bound variables.
- Without free variables, we have no λ terms.
- To perform β -convesion, we need substitution operation.
- To show $\lambda x. M = \lambda x. N$, we need to show $M = N$ and use ξ -rule.
- To prove Church-Rosser theorem, we need α -conversion rule.

By contrast, none of these observations apply to CL.

Plan of the talk

The purpose of this talk is to analyze Curry's untyped λ -calculus **syntactically** and **semantically**.

To this end, we split the talk into three parts.

- Part 1 : Symbolic Expressions
- Part 2 : Syntax of the lambda calculus
- Part 3 : Semantics of the lambda calculus

We will use the type **B** consisting of two boolean values **true** and **false**.

Part 1 : Symbolic Expressions

System	Name	Metavariable	Equality
Symbolic Expressions	Σ	S, T	$\equiv$
Expressions	$\langle \text{Exp} \rangle$	E, F, G	$\equiv$

The type Σ of Symbolic Expressions

McCarthy introduced **symbolic expressions** which he used to write codes for the Lisp programming language. McCarthy's symbolic expressions can be characterized as a **free magma** on a countable set of atoms, where a **magma** is an algebra with a single binary operator. (Serre and Bourbaki used the word **magma**.)

Here, we simplify McCarthy's symbolic expressions by allowing only one atom and write Σ for such a set of symbolic expressions. In order to provide a concrete notation for each element of Σ , we define Σ by the following BNF notation.

$$S, T : \Sigma ::= () \mid (ST)$$

We can also define Σ by the following inductive definition.

$$\frac{}{() : \Sigma} \text{ nil} \qquad \frac{S : \Sigma \quad T : \Sigma}{(ST) : \Sigma} \text{ cons}$$

Equality relation $\equiv$ on Σ

$$\frac{}{() : \Sigma} \text{ nil} \qquad \frac{S : \Sigma \quad T : \Sigma}{(ST) : \Sigma} \text{ cons}$$

The data type Σ defined here provides concrete expressions for representing **abstract syntax** in the sense of McCarthy.

It is a **free magma** generated from its only atom $()$ by the binary magma operator **cons**.

We define the **equality relation** $\equiv$ on Σ as follows.

$$\frac{}{() \equiv ()} \qquad \frac{S \equiv S' \quad T \equiv T'}{(ST) \equiv (S'T')}$$

All the systems we introduce below will be defined as subclasses of Σ and we will use $\equiv$ defined here as their syntactic equality relation.

The type of natural numbers $\langle \text{Nat} \rangle$

We define the type $\langle \text{Nat} \rangle$ of **natural numbers** as a sub-type of Σ . We use n, i, l, u, x, y etc. as metavariables ranging over natural numbers. We write n' for $n + 1$.

$$n : \langle \text{Nat} \rangle ::= \langle \text{NatZ} \rangle \mid \langle \text{NatS} \rangle$$

$$0 : \langle \text{NatZ} \rangle ::= ()()$$

$$n' : \langle \text{NatS} \rangle ::= (n0)$$

We will write $1, 2, 3, \dots$ for $0', 0'', 0''', \dots$. Namely,

$$1 : \equiv 0' \equiv (((()))())()$$

$$2 : \equiv 1' \equiv (((()))((()))())()$$

$$3 : \equiv 2' \equiv (((()))((()))((()))())()$$

In the next slide, we define the type $\langle \text{Exp} \rangle$ as an extension of $\langle \text{Nat} \rangle$, so that we will have the sub-type relation:

$$\langle \text{Nat} \rangle \subseteq \langle \text{Exp} \rangle \subseteq \Sigma.$$

The type of expressions $\langle \text{Exp} \rangle$

Here, we define a sub-type $\langle \text{Exp} \rangle$ of Σ as follows. Note that S is a symbolic expression.

$$E, F, G : \langle \text{Exp} \rangle ::= \langle \text{Exp} \rangle_1 \mid \langle \text{Exp} \rangle_2 \mid \langle \text{Exp} \rangle_3 \mid \dots$$

$$E_1 : \langle \text{Exp} \rangle_1 ::= (())S$$

$$E_2 : \langle \text{Exp} \rangle_2 ::= (E F)$$

$$E_3 : \langle \text{Exp} \rangle_3 ::= (E F G)$$

...

In the above definition, we wrote $(E F)$ for (EF) and $(E F G)$ for $(E(FG))$ etc. Elements of $\langle \text{Exp} \rangle_1$ are called **atoms**. We use atoms to encode words like: 'var', 'lam', 'pair',

In Part 2, every expression will be written using expressions in $\langle \text{Exp} \rangle$.

Part 2 : Syntax of the lambda calculus

System	Name	Metavariable	Equality
Curry's λ -calculus	λ	M, N, P	$\equiv, \equiv_\alpha$
Canonical λ -calculus	Λ	M, N, P	$\equiv$
Objects of ∇	Δ	M, N, P	$\equiv$

In Part 2, we deal only with **syntax** of the above systems, using the equality relation $\equiv$ we defined in Part 1.

Curry's λ -calculus λ as a syntactic type

We define λ as a type whose members are symbolic expressions inductively defined as follows. Recall that $x : \langle \text{Nat} \rangle$.

$$\frac{}{(\text{var } x) : \lambda} \quad \frac{M : \lambda}{(\text{lam } x M) : \lambda} \quad \frac{M : \lambda \quad N : \lambda}{(\text{pair } M N) : \lambda}$$

Equivalently, we can define λ as a syntactic class by means of the following context-free grammar.

$$\lambda : M, N ::= (\text{var } x) \mid (\text{lam } x M) \mid (\text{pair } M N)$$

Thus defined λ gives essentially the same lambda terms as defined by Curry.

The second rule is lambda-abstraction rule and the third is pair rule. Note that, usually, the **pair rule** is called **application rule**.

System	Name	Metavariable	Equality
Curry's λ -calculus	λ	M, N, P	$\equiv, \equiv_\alpha$
Canonical λ -calculus	Λ	M, N, P	$\equiv$
Objects of ∇	Δ	M, N, P	$\equiv$

Now, we define Λ as a subclass of λ consisting of canonical λ -terms.

The type Λ of canonical λ -terms

We define a family of types Λ^n indexed by n as follows.

$$\frac{x < n}{(\text{var } x) : \Lambda^n} \qquad \frac{M : \Lambda^{n'}}{(\text{lam } n \ M) : \Lambda^n} \qquad \frac{M : \Lambda^n \quad N : \Lambda^n}{(\text{pair } M \ N) : \Lambda^n}$$

Given x and n , any Λ^n -term M enjoys the following properties.

- $(\text{var } x)$ occurs free in $M \Leftrightarrow x < n$.
- $(\text{var } x)$ occurs bound in $M \Leftrightarrow n \leq x$.

This means that a variable in M may not occur both as free variable and bound variable. Frege and Gentzen imposed this condition for their logical calculus, but Curry did not.

We put $\Lambda := \Lambda^0$. Λ -terms correspond to closed λ -terms in de Bruijn level notation. A Λ^n -term contains at most n free variables.

System	Name	Metavariable	Equality
Canonical λ -calculus	Λ	M, N, P	$\equiv$
Objects of ∇	Δ	M, N, P	$\equiv$

In Λ , the notion of a **context** did not appear explicitly. However, the n in $(\text{lam } n \ M)$ represented the context of length n in reality.

Now, we introduce Δ which **internalizes** the notion of **context**.

The change from Λ to Δ corresponds to the change from the notion of **Hypothetical Judgment** to the notion of **Categorical Judgment**.

The type $\langle \Delta \rangle$

We define the type $\langle \Delta \rangle$ as follows. Recall that $u, x, i : \langle \text{Nat} \rangle$.

$$\frac{}{(\text{var } u \ x) : \langle \Delta \rangle} \quad \frac{}{(\text{prj } u \ i) : \langle \Delta \rangle} \quad \frac{M : \langle \Delta \rangle \quad N : \langle \Delta \rangle}{(\text{pair } u \ M \ N) : \langle \Delta \rangle}$$

u stands for the **context** of length u .

prj stands for **projection function** and will play the key role in the development of our theory.

We will use this type as the frame work in which we define the type Δ .

Anatomy of $\langle \Delta \rangle$

We redefine $\langle \Delta \rangle$ in terms of BNF notation.

$$\begin{aligned} M, N : \langle \Delta \rangle &::= \langle \Delta_{\text{var}} \rangle \mid \langle \Delta_{\text{prj}} \rangle \mid \langle \Delta_{\text{pair}} \rangle \\ \langle \Delta_{\text{var}} \rangle &::= (\text{var } \textcolor{red}{u} \ x) \\ \langle \Delta_{\text{prj}} \rangle &::= (\text{prj } \textcolor{red}{u} \ i) \\ \langle \Delta_{\text{pair}} \rangle &::= (\text{pair } \textcolor{red}{u} \ M \ N) \end{aligned}$$

Note that $\langle \Delta \rangle$ does not have a syntactic class for the λ -abstraction. In fact, it is distributed (as $\textcolor{red}{u}$) over the three syntactic classes: $\langle \Delta_{\text{var}} \rangle$, $\langle \Delta_{\text{prj}} \rangle$, and $\langle \Delta_{\text{pair}} \rangle$.

This has the effect of making the anatomy of the type $\langle \Delta \rangle$ $\textcolor{red}{algebraic}$.

Anatomy of $\langle \Delta \rangle$ (cont.)

We introduce the following notation.

$$V_x^u \equiv (\text{var } u \ x)$$

$$P_i^u \equiv (\text{prj } u \ i)$$

$$M \oplus^u N \equiv (\text{pair } u \ M \ N)$$

Note that $\oplus^u : \langle \Delta \rangle \rightarrow \langle \Delta \rangle \rightarrow \langle \Delta_{\text{pair}} \rangle$.

We define two functions $\text{car}, \text{cdr} : \langle \Delta_{\text{pair}} \rangle \rightarrow \langle \Delta \rangle$ as follows.

$$\text{car}(M \oplus^u N) \equiv M$$

$$\text{cdr}(M \oplus^u N) \equiv N$$

We have the following equality.

$$M \oplus^u N \equiv \text{car}(M \oplus^u N) \oplus^u \text{cdr}(M \oplus^u N).$$

Anatomy of $\langle \Delta \rangle$ (cont.)

Definition (Length of $\langle \Delta \rangle$ -expressions)

Given $M : \langle \Delta \rangle$, we write $|M|$ for the length of M .

$$|V_x^u| : \equiv u$$

$$|P_i^u| : \equiv (u + i)'$$

$$|M \oplus^u N| : \equiv u$$

Definition ($\langle \Delta_u \rangle$)

$$\langle \Delta_u \rangle := \{M : \langle \Delta \rangle \mid u \leq |M|\}.$$

$\langle \Delta_u \rangle$ is a subtype of $\langle \Delta \rangle$ consisting of $\langle \Delta \rangle$ expressions with length $\geq u$. So, we have $\langle \Delta_0 \rangle = \langle \Delta \rangle$ and the following strictly decreasing sequence of types.

$$\langle \Delta_0 \rangle \supset \langle \Delta_1 \rangle \supset \langle \Delta_2 \rangle \supset \cdots .$$

The type Δ

The type Δ is defined as follows. Δ is a subtype of $\langle \Delta \rangle$.

$$\frac{}{P_i^u : \Delta} \quad \frac{M : \Delta \quad N : \Delta}{M \oplus^u N : \Delta} (\dagger),$$

where $(\dagger)$ is the side condition: $u \leq |M|$ and $u \leq |N|$.

Δ does not have the **var-rule**, and it implies that all Δ -terms are **closed**.

Definition (Δ^u)

$\Delta^u := \{M : \Delta \mid u \leq |M|\}.$

$$\Delta = \Delta^0 \supset \Delta^1 \supset \Delta^2 \supset \dots .$$

Examples of Δ -terms

- ① $\mathbf{I} : \equiv P_0^0$. $\mathbf{I}$ takes 1 argument M_0 and returns M_0 .
- ② $\mathbf{K} : \equiv P_1^0$. $\mathbf{K}$ takes 2 arguments M_0, M_1 and returns M_0 .
- ③ In general, P_i^u takes $(u + i)'$ arguments $M_0, \dots, M_{u+i}$ and returns M_u .
Hence, u corresponds to [the de Bruijn level](#) and i corresponds to [the de Bruijn index](#).
 P_i^u corresponds to the following λ -term:

$$\lambda x_0 \dots x_u \dots x_{l+i}. x_u.$$

- ④ $\mathbf{S}_\Lambda : \equiv (\text{lam } 0 (\text{lam } 1 (\text{lam } 2 (\text{pair } (\text{pair } (\text{var } 0) (\text{var } 2)) (\text{pair } (\text{var } 1) (\text{var } 2))))))$.
In Δ , we have

$$\begin{aligned} \mathbf{S} &: \equiv (\text{pair } 3 (\text{pair } 3 (\text{prj } 0 \ 2) (\text{prj } 2 \ 0)) (\text{pair } 3 (\text{prj } 1 \ 1) (\text{prj } 2 \ 0))) \\ &\equiv (P_2^0 \oplus^3 P_0^2) \oplus^3 (P_1^1 \oplus^3 P_0^2). \end{aligned}$$

Context insert

Definition (Context insert)

Given $M : \Delta^u$ and k , we define context insert function $\uparrow_k^u : \Delta^u \rightarrow \Delta^{u+k}$ as follows. The result N satisfies $|N| = |M| + k$.

$$\uparrow_k^u P_i^v \equiv \begin{cases} P_i^{v+k} & \text{if } u \leq v, \\ P_{i+k}^v & \text{if } v < u. \end{cases}$$
$$\uparrow_k^u (M \oplus^v N) \equiv (\uparrow_k^u M) \oplus^{v+k} (\uparrow_k^u N)$$

Note that $\uparrow$ distributes over $\oplus$, but involves action of k on contexts.

This function corresponds to the k times iteration of Gentzen's [weakening](#)-rule in [LJ](#).

Application operation

Definition ($\text{app}^u : \Delta^u \rightarrow \Delta^u \rightarrow \Delta^u$)

We write $M \otimes^u P$ for $\text{app}^u(M, P)$.

① $|M| = u$: $M \otimes^u P \equiv M \oplus^u P$.

② $|M| > u$ and $M = P_i^v$:

$$P_i^v \otimes^u P \equiv \begin{cases} P_i^{v-1} & \text{if } u < v, \\ \uparrow_i^u P & \text{if } u = v, \\ P_{i-1}^v & \text{if } u > v. \end{cases}$$

③ $|M| > u$ and $M = M_1 \oplus^{v'} M_2$:

$$(M_1 \oplus^{v'} M_2) \otimes^u P \equiv (M_1 \otimes^u P) \oplus^{v'} (M_2 \otimes^u P)$$

The application operation corresponds to the **substitution** operation in the λ -calculus.

While the substitution operation takes three arguments, $\otimes^u$ is an algebraic **binary operator**. Observe that, in the third item, $\otimes^u$ distributes P over $\oplus$.

Application Lemma

We have the following lemma.

Lemma (Application Lemma)

If $u < v'$, $M : \Delta^{v''}$, $N : \Delta^{v'}$ and $P : \Delta^u$. Then

$$(M \otimes^{v'} N) \otimes^u P \equiv (M \otimes^u P) \otimes^{v'} (N \otimes^u P).$$

Compare this with the following equality, which follows from the definition of application.

$$(M \oplus^{v'} N) \otimes^u P \equiv (M \otimes^u P) \oplus^{v'} (N \otimes^u P)$$

Application Lemma is a generalization of [substitution lemma](#) in the λ -calculus.

Part 3 : Semantics of the lambda calculus

System	Name	Metavariable	Equality
Canonical λ -calculus	Λ	M, N, P	$\equiv, =_{\Lambda}$
Objects of ∇	Δ	M, N, P	$\equiv, =_{\Delta}$
Morphisms of ∇	∇	d, e, f, g	$\equiv, =_{\nabla}$

In this final part, we introduce the final system ∇ .

∇ is defined as a syntactic extension of Δ by adding meta-level operators for **multiplying** and **composing** ∇ -terms.

Finally, the semantics of Λ -terms is given by defining the denotation map $\llbracket - \rrbracket : \Lambda \rightarrow \Delta$. Here, the **semantic** equality relation $=_{\Delta}$ on Δ plays the key role.

The type $\langle \nabla \rangle$

We introduce the following notation.

$$d \star^u e \equiv (\text{mult } u \ d \ e)$$

$$d ; e \equiv (\text{comp } d \ e)$$

We define the type $\langle \nabla \rangle$ as follows.

$$\begin{array}{c} \overline{V_i^u : \langle \nabla \rangle} \quad \overline{P_i^u : \langle \nabla \rangle} \\[10pt] \frac{d : \langle \nabla \rangle \quad e : \langle \nabla \rangle}{d \oplus^u e : \langle \nabla \rangle} \quad \frac{d : \langle \nabla \rangle \quad e : \langle \nabla \rangle}{d \star^u e : \langle \nabla \rangle} \quad \frac{d : \langle \nabla \rangle \quad e : \langle \nabla \rangle}{d ; e : \langle \nabla \rangle} \end{array}$$

The variable V_i^u is used only when we extend ∇ to $\nabla[n]$, and it is not used when we construct ∇ .

The type ∇

The type ∇ is defined as a subtype of $\langle \nabla \rangle$ by removing the **var-rule** and restricting the remaining formation rules of $\langle \nabla \rangle$. Namely, the second and the third rules must satisfy the condition $(\dagger) \ u \leq |d| \text{ and } u \leq |e|$, and the fourth rule must satisfy the condition $(\dagger) \ \text{cod}(d) = \text{dom}(e)$.

Because of $(\dagger)$, dom and cod are defined simultaneously with ∇ .

$$\frac{}{P_i^u : \nabla} \quad \frac{d : \nabla \quad e : \nabla}{d \oplus^u e : \nabla} (\dagger) \quad \frac{d : \nabla \quad e : \nabla}{d \star^u e : \nabla} (\dagger) \quad \frac{d : \nabla \quad e : \nabla}{e ; u : \nabla} (\dagger)$$

' $\star$ ' is used to **apply** d to the argument e , and ';' is used to apply d (to its argument), and then apply e .

The **semantic equality relation** $=_{\nabla}$ on ∇ will be given as a part of the definition of ∇ .

The category ∇

We define ∇ as a category whose **objects** are elements of Δ and **morphisms** are elements of ∇ .

Since Δ becomes a subtype of ∇ , we define the **identity morphism** on $M : \Delta$ by putting $\text{Id}_M := M : \nabla$.

To define **composition** of morphisms, we define $\text{dom} : \nabla \rightarrow \Delta$ and $\text{cod} : \nabla \rightarrow \Delta$ in the next slide. Assuming these two functions, we define composition as follows. We will write $f : M \rightarrow N$ if $\text{dom}(f) \equiv M$ and $\text{cod}(f) \equiv N$.

Definition (Composition)

Given, $f : M \rightarrow N$ and $g : N \rightarrow P$, the composition of g and f , written $g \circ f$, is defined by putting $g \circ f := f ; g$, where $u \equiv |\text{dom}(d)|$.

Transitivity of composition can be checked easily.

Domain and codomain

Definition (dom and $\text{cod} : \nabla \rightarrow \Delta$)

$$\text{dom}(P_i^u) : \equiv P_i^u$$

$$\text{dom}(d \oplus^u e) : \equiv \text{dom}(d) \oplus^u \text{dom}(e)$$

$$\text{dom}(d \star^u e) : \equiv \text{dom}(d) \oplus^u \text{dom}(e)$$

$$\text{dom}(d ; e) : \equiv \text{dom}(d)$$

$$\text{cod}(P_i^u) : \equiv P_i^u$$

$$\text{cod}(d \oplus^u e) : \equiv \text{cod}(d) \oplus^u \text{cod}(e)$$

$$\text{cod}(d \star^u e) : \equiv \text{cod}(d) \otimes^u \text{cod}(e)$$

$$\text{cod}(d ; e) : \equiv \text{cod}(e)$$

Semantic equality $=_{\nabla}$

Definition ($=_{\nabla} : \nabla \rightarrow \nabla \rightarrow \mathbf{B}$)

Let d and $e : \Delta$.

$$d =_{\Delta} e \Leftrightarrow \text{dom}(d) \equiv \text{dom}(e) \text{ and } \text{cod}(d) \equiv \text{cod}(e).$$

This equality is primitive-recursively decidable.

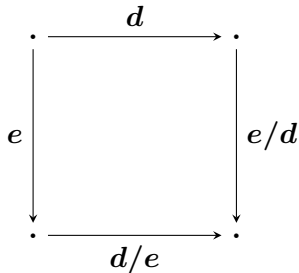
Derivation

Our intended meaning of a morphism $d : M \rightarrow P$ is that d is a **derivation** (or a proof) of the judgment $M \rightarrow P$, whose meaning, in turn, is that there is a way to **convert** M to N using the conversion rules of the ∇ -calculus.

Under this interpretation, it is natural to require that $\text{dom}(d) \equiv M$ and $\text{cod}(d) \equiv N$.

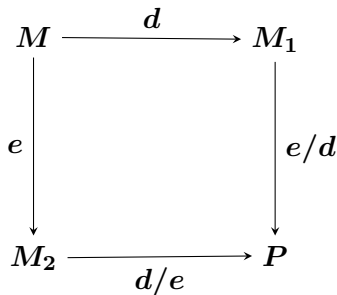
Residual and Church-Rosser Property

Given $d : \nabla$ and $e : \nabla$ such that $\text{dom}(d) \equiv \text{dom}(e)$, we can define $d/e : \nabla$ and $e/d : \nabla$ in such a way that the following diagram commutes.



- ① $\text{dom}(d) = \text{dom}(e)$,
- ② $\text{dom}(d/e) = \text{cod}(e)$,
- ③ $\text{dom}(e/d) = \text{cod}(d)$,
- ④ $\text{cod}(d/e) = \text{cod}(e/d)$, and
- ⑤ $d ; (e/d) = e ; (d/e)$.

Church-Rosser property



Let us compare the statements of the [Church-Rosser property](#) in the λ -calculus and in the ∇ -calculus.

(λ) $M \rightarrow M_1, M \rightarrow M_2 \implies \exists P_1, P_2 (M_1 \rightarrow P_1, M_2 \rightarrow P_2 \text{ and } P_1 \equiv_{\alpha} P_2)$

(∇) $d : M \rightarrow M_1, e : M \rightarrow M_2 \implies e/d : M_1 \rightarrow P \text{ and } d/e : M_2 \rightarrow P$

where $P \equiv \text{cod}(e/d) \equiv \text{cod}(d/e)$.

Semantic equality $=_{\Delta}$ on Δ

We introduce the semantic equality relation $=_{\Delta}$ on Δ as follows.

Definition (Semantic equality ($=$) on Δ)

Let M and N be Δ -terms. We say that M and N are **semantically equal** if there exist d, e and P such that $d : M \rightarrow P$ and $e : N \rightarrow P$. We write $M =_{\Delta} N$ in this case.

This equality can be written as the following inference rule.

$$\frac{d : M \rightarrow P \quad e : N \rightarrow P}{M =_{\Delta} N}$$

Semantic equality may be thought of as **computational equality**.

Semantic equality on Δ (cont.)

The **reflexivity** is shown as follows.

$$\frac{M : M \rightarrow M \quad M : M \rightarrow M}{M = M}$$

For the **symmetry**, suppose that

$$\frac{d : M \rightarrow P \quad e : N \rightarrow P}{M = N}.$$

Then we have

$$\frac{e : N \rightarrow P \quad d : M \rightarrow P}{N = M}.$$

Note that the second derivation is obtained by the action of **swapping** the derivations d and e .

Semantic equality on Δ (cont.)

For **transitivity**, suppose that

$$\frac{d : M \rightarrow Q \quad e : N \rightarrow Q}{M = N} \quad \text{and} \quad \frac{f : N \rightarrow R \quad g : P \rightarrow R}{N = P}.$$

Then, by CR, we have

$$\frac{d; (f/e) : M \rightarrow S \quad g; (e/f) : P \rightarrow S}{M = P}$$

where $S \equiv \text{cod}(f/e) \equiv \text{cod}(e/f)$.