Type Theory と Classifying Category

Shotaro Toki*

2023 年 9 月

Over view

*1 この文書は 4 つのパートからなる．まず Introduction で数学の基礎や type theory の背景や初歩を簡単に共有し，type theory の持つ 2 つの側面について話す．そして I，II でその 2 つの側面について詳しく解説し，III で現在行われている研究や文献を紹介する．

目次

Over view 1

Introduction 2

第 I 部　数学の基礎としての HoTT 4

1 Type theory の基礎 4
- 1.1 Judgements . . . 4
- 1.2 Rules . . . 5
- 1.3 2 つの equal . . . 6

2 基本的な型たち 6
- 2.1 Function types . . . 6
- 2.2 Product types . . . 8
- 2.3 Coproduct types . . . 9
- 2.4 具体的な型たち . . . 10

3 Dependent types 12
- 3.1 Dependent function types . . . 12

* © 2023 Shotaro Toki. All rights reserved.

*1 学部 2 年のときに都数の発表に向けて書いたものである．拙いがお許しあれ．rule を明確に共有した上で話を進めるというスタイルは type theory の文献であまり見ないと思うので，そのような意味で価値があると思う．

3.2 Dependent pair types 13
3.3 elmination rule,computation rule の dependent 化 14

4 Identity types 15
4.1 Identity type の導入 15
4.2 homotopy 的解釈 16

5 Higher inductive types 16
5.1 型の構成を振り返って 16
5.2 Higher inductive types 17

6 Univalence 18
6.1 Equivalences 18
6.2 The function extensionality axiom 18
6.3 The univalence axiom 19

第 II 部 Classifying category 20

7 Type theory **とは何か** 20

8 Classifying category **の作り方** 20

9 **普遍性** 21

10 **新たな数学観** 22

第 III 部 行われている研究，文献の紹介 23

Introduction

ここでは数学の基礎とは何かを振り返り，近年注目を浴びてきた type theory の歴史や特徴を説明する．

数学の基礎について

基礎論学徒でない限り，普段数学をしていて数学の基礎を気にすることはあまりないと思う．そこで，数学の基礎について少しばかり共有する．まず，一般的な大学で教わる数学で何が基礎となることが前提とされているかというと，それは素朴集合論である．素朴とは「感覚に任せる」という意味で問題ない．つまり，集合は皆の感覚に任せて，そこからの議論を厳密にやっていこうという姿勢である．大学の集合論の授業も，直積などの定義は厳密にやっていくが，集合それ自体についてはあまり言及されないことが多いと思う．そして，この感覚に任された集合を，公理系を置き形式化したのが，公理的集合論だ．その代表的な公理系として ZFC がある．つまり，どのような集合を作れるかや，集合として等しいとはどのようなことかが公理に書か

れているのだ．これをスタートにすれば集合に関して感覚に頼らないで済む．しかし，これではまだ完璧な形式化とは言えない．推論の部分を感覚に頼ってしまっているからだ．すなわち，論理記号の操作の部分だ．ZFC から始まる数学は命題がすべて論理式で書かれる．そして，とある論理式からとある論理式を導く．これが推論であるわけだが，公理集合論までの掘り下げだけだとそこは感覚で行うしかない．そこで，その論理記号の操作を形式化したものが一階論理である．つまり，普段の数学の形式化を最大まで突き詰めると，「一階論理＋ ZFC」という体系に行きつくのである．この基礎をもとに厳密に数学を展開していった一つの作品として [1] がある．この本では 1 巻で一階論理，ZFC，集合論が展開され，2 巻でやっと自然数が定義される．

Type theory とは

型理論 (Type Theory) はもともと 20 世紀初頭に Russell が集合論の Russell のパラドックスを回避すべく考案した体系であったが，当時は数学よりも computer science への応用として主に活躍していた．しかし Martin-Löf がそれを整備し直したことにより，数学の基礎として注目されることになった．その type theory は Martin-Löf type theory(MLTT) と呼ばれ，大抵，数学の基礎などの文脈の中で type theory と言ったら，それを指す．そしてその後，様々な派生が生まれていった．HoTT もそのうちの 1 つである．また Coq などの証明支援ツールへの応用も注目すべき話題である．Coq などは基本的に型理論が基礎となっていて，四色定理やケプラー予想などの問題を解決に導いた．

そして，type theory の集合論とは違う自由度から，数学全体の基礎以外の見方も出てきた．それは II で主に解説するが，category の internal language としての側面である．以降は，数学全体の基礎としての側面を第一の側面，internal language としての側面を第二の側面とする．もちろん type theory の一種である HoTT もその 2 つの側面を持つ．I では主に HoTT の第一の側面を解説する．

第 I 部
数学の基礎としての HoTT

ここでは HoTT という type theory の第一の側面について解説する．つまり「一階論理＋ ZFC」に取って代わる数学全体の基礎としての HoTT だ．II も見据えて，かなり厳密に HoTT を作っていくので，普段の数学がこの HoTT という体系の中でどのように再現されていくのかに注目しながら追ってほしい．

1 Type theory の基礎

これから HoTT という理論を作っていくわけだが，数学の基礎となる理論をつくるため，今は何も知識が仮定されていない．つまり，通常数学の文脈というのは何かしらの前提がある．例えば，群論を学ぼうとして本を開いたとき，そこでは既に一階論理 +ZFC という基礎があり，論理や集合論に関する議論は前提とされている．しかし，今回は基礎理論を作るわけなので，それらが一切ないということだ．ブルバキの数学原論 [1] のような，いわゆる「原論」的な姿勢だ．他に新しい有名なものとして，斎藤毅の数学原論 [12] があるが，これは確かに圏論を基礎に数学を構築していくという「原論」的な姿勢だが，論理の部分は構築されずに前提となっているため，完全な形式化というわけではない．

1.1 Judgements

では始めに，HoTT の judgement の形を設定する．judgement とはその Theory の"素"な主張のことである．例えば，一階論理＋ ZFC の数学では，「命題 A は証明された」という 1 種類の judgement しかない．よく考えればこれは当たり前のことである．つまり，普段数学する中での主張というのは常に命題として表現され，それが証明され始めて主張として意味を持つ．よって，judgement はこの 1 種類しかない．対して HoTT の judgement は 3 種類あり，それぞれ

$$\Gamma\,\mathrm{ctx} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a \equiv a' : A$$

という形になっている．ここで Γ は，$x_1 : A_1, x_2 : A_2, \cdots, x_n : A_n$ のような形の記号の有限列を置き換えたものである．また，3 つ目の judgement に関してはしばらく説明しない．最初の 2 つのみについて考える．さっそく意味を与えていこう．意味というのは，この単なる記号を人間がどう解釈するかであって，数学をする上で究極的には必要のないものであるため，あくまでも今後 rule などを導入していく中で理解をしやすくするためだけにあるものだということに注意してほしい．

- $a : A$ は，「a は型 A の要素である」と読まれる
- $\Gamma \vdash a : A$ とあるとき，Γ は context と呼ばれ，全体としては「Γ のとき $a : A$」と読まれる
- $\gamma\,\mathrm{ctx}$ は，「Γ は context として有効だ」と読まれる

とりあえず今は，型を集合のようなものだと思って，その要素を元だと思えばよい．これから rule が与えられ，実際に judgement を得ていく中で意味は分かってくると思うので，とりあえず読み方を受け入れてもらえれば良い．

1.2 Rules

rule というのは，既存の judgement から新しい judgement を得る方法である．従来の数学でも，HoTT でも rule はたくさんある．数学をゲームに例えるなら，judgement がプレイヤーの位置で，rule はそのプレイヤーの進め方である．

基礎の中でも基礎となる rule を 6 つ導入する．まずはすべての始まりである ctx-EMP は以下のようなものになる．

$$\frac{}{\cdot\,\mathrm{ctx}}\ \textsc{ctx-emp}$$

rule はバーの上の judgement から下の judgement を得るというものである．ctx-EXP は単なるラベルである．今上のバーには何も書かれていない．つまり問答無用に下の judgement が得られるということだ．

では次に 2，3 つ目の rule として，universe を導入する．これは型全体の集合のようなものだと思ってよい．そして universe は複数あり，階層構造をなしている．

$$\frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-INTRO} \qquad \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-CUMUL}$$

1 つ目の rule を見てみると，これは Γ が仮定されているとき，そこから「Γ のとき，$\mathcal{U}_i$ は $\mathcal{U}_{i+1}$ の要素である」という judgement が得られるということを意味している．すなわち仮定が 1 つも使われずに $\mathcal{U}_i$ が導入されているということだ．

Example 1.1. 実際に今までの 2 つを組み合わせて judgement を得てみよう．

$$\dfrac{\dfrac{}{\cdot\,\mathrm{ctx}}\ \textsc{ctx-exp}}{\cdot \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-INTRO}$$

$\mathcal{U}$-INTRO では，Γ を empty として考えたわけである．こうして，$\cdot \vdash \mathcal{U}_i : \mathcal{U}_{i+1}$ という judgement が無から得られた．

改めて 2 つの universe の rule を見てみると，$\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \cdots$ がまさに階層構造を成していることが感じられるだろう．ここから私たちはとある i を固定して，$\mathcal{U}_i$ を型全体だと思って数学をしていく（別に $i = 0$ だろうが $i = 1$ だろうが議論は変わらないので）．よって，$\mathcal{U}_i$ を $\mathcal{U}$ と略記することも多々ある．つまり，「$A : \mathcal{U}$」という主張は，「A は型である」という意味になるわけだ．ここで universe に関する rule をもう一つ追加する．

$$\frac{x_1 : A_1, \cdots, x_{n-1} : A_{n-1} \vdash A_n : \mathcal{U}}{(x_1 : A_1, \cdots, x_{n-1} : A_{n-1}, x_n : A_n)\,\mathrm{ctx}}\ \textsc{ctx-ext}$$

まずバーの上の judgement の意味は「$x_1 : A_1, \cdots, x_{n-1} : A_{n-1}$ という仮定のもとで，A_n は型である」である．そしてバーの下を見ると，仮定に $x_n : A_n$ が加わっている．つまり，A が型として現れたなら，その要素があるいう仮定は成立するということを言っているわけだ．

次に 5 つ目に rule として，

$$\frac{(x_1 : A_1, \cdots, x_n : A_n)\,\mathrm{ctx}}{x_1 : A_1, \cdots, x_n : A_n \vdash x_i : A_i}\ \textsc{Vble}$$

を加える．これは当たり前に感じられると思う．「$x : A$ のとき $x : A$ である」というのは言えていいことだ．そして最後の 6 つ目の rule は

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, \Delta \vdash b : B}{\Gamma, \Delta[a/x] \vdash b[a/x] : B[a/x]} \text{ SUBST1}$$

$b[a/x]$ の意味は b の中に出てくる文字 x をすべて a に置き換えたものである．この rule によって，代入のような操作ができるようになった．

1.3 2 つの equal

HoTT には, 2 種類の equal がある．最初の equal は, ずっと説明を後回しにしてきた 3 つ目の judgement のことである．

$$\Gamma \vdash a \equiv a' : A$$

意味は「型 A の要素 a と a' は等しい」であり，この equal のことを judgemental equal という．この equal はかなり強くもし $a \equiv a' : A$ が得られていたら，その後に出てくる a はすべて a' に置き換えて良い．この judgemental equal に関する諸 rule は，意味がわかりやすいということもあるので，ここではわざわざ言及しない．[11] の p423 を参照してほしい．

2 つ目の equal は命題としての equal であり，propositional equal という．命題は型理論において型であったため，それにあたる型があるということだ．それは以下のように導入される．

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathcal{U}} \text{ I-FORM}$$

つまり，型 A の任意の要素 a，b に対して $a =_A b$ という型があるというわけだ．propositional equal に関しては 4 で詳しく説明する．

2 基本的な型たち

ここから実際に型を導入していく．型を導入すると言っても形式的には rule を加えるだけである．しかし，この型という扱いやすい単位によって，従来の数学とは違った型理論の柔軟さが現れていると思う．

2.1 Function types

ではさっそく最初の型を導入していこう．まず型は以下の 4 つの rule によって形づけられる．

- formation rule：どのようにその型が導入されるか
- introduction rule：どのようにその型の要素を作れるか
- elimination rule：その要素を使って何が得られるか
- computation rule：その得たものはどのように計算できるか

さっそく function type の４つの rule を見てみよう[*2].

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \to B : \mathcal{U}} \to\text{-FORM} \qquad \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).b : A \to B} \to\text{-INTRO}$$

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B} \to\text{-ELIM} \qquad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x : A).b)(a) \equiv b[a/x] : B} \to\text{-COMP}$$

急に $\lambda(x : A).b$ というものが出てきて困惑するかもしれないが，これは単なる記号だと思えば良い．ここが普段の数学との違いでもあり，普段の数学では，何か新しいものが出てきたら，必ずそれはどのような対象式を置き換えたものなのかという定義が来る．しかし，この型理論では，突然見たこともない記号が rule の中に現れることが多々ある．しかし，それは別にそれで良くて，そこで私たちは何を期待すれば良いかといえば，それは，その記号がどのように使われるのか，またどのように計算されるのかということである．ただ，記号の意味的な解釈を持っておくことは大事なので，共有しておくと，$\lambda(x : A).b$ は変数 x に対して b を返すという関数である．ここでもちろん b というのは文字 x を含む可能性がある．また x が何の型の要素なのかというのは，$\lambda(x : A).b$ が何の型に属すのかによってすぐ分かるため，単に $\lambda x.b$ と略記することが多い．この λ による関数の表記を λ-abstruction という（個人的にこの表記は普段の数学でも便利なので使って良いと思う）．

また elimination rule と computation rule は少し分かりにくいので説明すると，まず elimination rule ではバーの上に出てくる $A \to B$ の任意の要素 f がどのように使われるかを表していて，今回は，もし $a : A$ があれば B の要素 $f(a)$ というものが得られるということを言っている．ただ，ここで $f(a)$ が得られたわけだが，この $f(a)$ というのが何だかわからなければ話はここで終わってしまう．そこで登場するのが computation rule だ．これは f が具体的に $\lambda(x : A).b$ という形だったら，$(\lambda(x : A).b)(a)$ は $b[a/x]$ と計算できるということを言っているのである．これはまさに普段の関数の機能をそのまま表していると言えよう．

そして，function type は型の中でも基本的な型なので，例外的に以下の uniqueness principle という rule が追加で導入される．

$$\frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f \equiv (\lambda x.f(x)) : A \to B} \to\text{-UNIQ}$$

つまり，どんな要素も λ-abstruction の形に表せるということを言っている．

Remark 2.1. 「一階論理＋ ZFC」に代わる数学全体の基礎として HoTT を導入しているのであった．では普段の数学における命題は HoTT において何になるであろうか．じつはそれは型になる．そしてなんとその型の要素がその命題の証明となる．つまり，HoTT において，「命題を証明する」ということは，「その命題にあたる型の要素を作る」ということになるのだ．この「型の命題として見る」ことは，今後の理解をとても助けるものとなる．例えばこの function type を命題として見た場合，これは「ならば」になる．この意味付けを意識して rule を見るとすべて自然に感じられることだろう．

Example 2.1. function type の例を見ていこう．以下のように変数として型 A を用意して，とある結果が

[*2] 今後，以下のような $\Gamma, x : A \vdash b : B$ や $\Gamma, x : A, y : B \vdash c : C$judgement が現れるが，$\vdash$ の右側の型（それぞれ B,C）は文字 x を含まないものとしてほしい．もちろんこれは説明の都合上の制約であり，3 で制約はなくなる．

得られる．

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\,\mathrm{ctx}}\,\textsc{ctx-emp}}{\vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\,\mathcal{U}\textsc{-intro}}{(A:\mathcal{U})\,\mathrm{ctx}}\,\textsc{ctx-ext}}{A:\mathcal{U}\vdash A:\mathcal{U}}\,\textsc{Vble}}{(A:\mathcal{U},x:A)\,\mathrm{ctx}}\,\textsc{ctx-ext}}{A:\mathcal{U},x;A\vdash x:A}\,\textsc{Vble}}{A:\mathcal{U}\vdash \lambda x.x : A\to A}\,{\to}\textsc{-intro}$$

型 A を命題として捉えたならば，これは P ならば P という自明な命題の証明と見ることができ，A を集合と捉えたならば，これはまさに恒等写像を作ったことになるわけである．恒等写像は今後も出てくるので，$\mathrm{id}_A :\equiv \lambda x.x$ としておく．

2.2 Product types

product type は集合として見ると直積であり，命題として見ると「かつ」である．

$$\dfrac{\Gamma\vdash A:\mathcal{U}\qquad \Gamma\vdash B:\mathcal{U}}{\Gamma\vdash A\times B:\mathcal{U}}\,{\times}\textsc{-form}\qquad \dfrac{\Gamma\vdash a:A\qquad \Gamma\vdash b:B}{\Gamma\vdash (a,b):A\times B}\,{\times}\textsc{-intro}$$

$$\dfrac{\Gamma\vdash C:\mathcal{U}\qquad \Gamma,x:A,y:B\vdash g:C\qquad \Gamma\vdash p:A\times B}{\Gamma\vdash \mathrm{rec}_{A\times B}(C,x.y.g,p):C}\,{\times}\textsc{-elim}$$

$$\dfrac{\Gamma\vdash C:\mathcal{U}\qquad \Gamma,x:A,y:B\vdash g:C\qquad \Gamma\vdash a:A\qquad \Gamma\vdash b:B}{\Gamma\vdash \mathrm{rec}_{A\times B}(C,x.y.g,(a,b))\equiv g[a,b/x,y]:C}\,{\times}\textsc{-comp}$$

elimination rule と computation rule が分かりにくいと思うので，そこを説明する．function type はかなり基本的な型だったので例外的だが，今回の型からは elimination rule と言ったら「考えている型の要素が他の任意の型 C にどんな要素を作るか」ということを表す．これが「型の要素をどのように使うか」と言った言明と同じだということは感じられると思う．つまり，今回でいうと，「バーの上に現れている要素 $p:A\times B$ から，どのようなときにどのような C の要素を得るのか」ということを表現しているわけである． では実際に今回の elimination rule は何を言っているのかを考えるにあたって，型を命題として考えてみよう．このとき，$A\times B$ は A かつ B を意味していた．このとき，「バーの上に現れている要素 $p:A\times B$ から，どのようなときにどのような C の要素を得るのか」とは「どのようなときに A かつ B の証明から，C の証明を得られるのか」ということになる．そして，それがまさしく「$x:A$,$y:B$ という 2 つの仮定のもとで $g:C$ が得られたとき」ということになる．つまり A と B の 2 つの情報量から C を証明できたときということになる．何を当たり前なことを，と思うかもしれないが，今私たちは A かつ B とは何なのかを定めているわけなので，その当たり前をそのまま定式化したわけなのである．

ただ実践では先に用意した function type を使ってもっと informal にこれらを扱う．function type の言葉を使えば，「バーの上に現れている要素 $p:A\times B$ から，どのようなときにどのような C の要素を得るのか」は「どのように $A\times B\to C$ の要素を得られるか」ということになるであろう．そしてそのために必要と考えていた「$x:A$,$y:B$ という 2 つの仮定のもとで $g:C$ が得られたとき」というのは「$g:A\to(B\to C)$ が得

られたとき」ということになる．そして elimination rule をまとめて実践的に表現すると，$A, B, C : \mathcal{U}$ に対して，

$$\mathrm{rec}'_{A\times B,C} : (A \to (B \to C)) \to (A \times B \to C)$$

となる．そして，ここで computation rule はどのように解釈されるかというと，

$$\mathrm{rec}'_{A\times B,C}(g)\,((a,b)) \equiv g(a)(b)$$

この informal な定式化は実際には formal に導けるので，気持ちさえ分かれば気にせずこちらを使えば良い．

Example 2.2. 実際に実践意識で projection を作ってみよう．つまり目標は集合論でいうところの $\mathrm{pr}_1 : A \times B \to A$ で $\mathrm{pr}_1\,((x,y)) = x$ である．恐らくここまでやれば，形式的体系の仕組みが大分わかってきたと思うので，これ以降は以下のような informal な説明も出てくる．

A, B という型があるとき，今から私たちが想定するような $A \times B \to C$ の要素が得たいわけだ．そこで何が必要だったかというと，$A \to (B \to C)$ の要素である．そしてそれは以下のように作ればよいのではないのだろうか．

$$\dfrac{\dfrac{\dfrac{(A, B : \mathcal{U}, x : A, y : B)\,\mathrm{ctx}}{A, B : \mathcal{U}, x : A, y : B \vdash x : A}\,\textsc{Vble}}{A, B : \mathcal{U}, x : A \vdash \lambda y.x : B \to A}\,{\to}\text{-INTRO}}{A, B : \mathcal{U} \vdash \lambda x.(\lambda y.x) : A \to (B \to C)}\,{\to}\text{-INTRO}$$

ここで $\mathrm{pr}_1 :\equiv \mathrm{rec}'_{A\times B,A}\,(\lambda x.(\lambda y.x))$ と定義すれば，$(a,b) : A \times B$ のとき，computation rule より，

$$\begin{aligned}\mathrm{pr}_1\,((a,b)) &\equiv (\lambda x.(\lambda(y.x)))\,(a)(b)\\ &\equiv (\lambda y.a)\,(b)\\ &\equiv a\end{aligned}$$

と求めていた結果が得られた．

2.3 Coproduct types

coproduct type$A + B$ は集合として見ると disjoint union で，命題として見ると「または」である．introduction rule と computation rule がそれぞれ２種類ある．

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A + B : \mathcal{U}}\,{+}\text{-FORM}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathrm{inl}(a) : A + B}\,{+}\text{-INTRO}_1 \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash \mathrm{inr}(b) : A + B}\,{+}\text{-INTRO}_2$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma, x : A \vdash c : C \qquad \Gamma, y : B \vdash d : C \qquad \Gamma \vdash e : A + B}{\Gamma \vdash \mathrm{rec}_{A+B}(C, x.c, y.d, e) : C}\,{+}\text{-ELIM}$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma, x : A \vdash c : C \qquad \Gamma, y : B \vdash d : C \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{rec}_{A+B}(C, x.c, y.d, \mathrm{inl}(a)) \equiv c[a/x] : C}\,{+}\text{-COMP}_1$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma, x : A \vdash c : C \qquad \Gamma, y : B \vdash d : C \qquad \Gamma \vdash b : B}{\Gamma \vdash \mathrm{rec}_{A+B}(C, x.c, y.d, \mathrm{inr}(b)) \equiv d[b/y] : C}\,{+}\text{-COMP}_2$$

今回も elimination rule と computation rule をより実践的に表現すると，

$$\mathrm{rec}'_{A+B,C} : (A \to C) \to (B \to C) \to (A + B \to C)$$

$$\mathrm{rec}'_{A+B,C}(g_1)(g_2)\,(\mathrm{inl}(a)) \equiv g_1(a)$$

$$\mathrm{rec}'_{A+B,C}(g_1)(g_2)\,(\mathrm{inr}(b)) \equiv g_2(b)$$

となる．またお気持ちとしては，型を命題を見るのが分かりやすい．このとき $A + B \to C$ は「A または B，ならば C」ということになるがこれを示すために普段何をするだろうか．それは「場合分け」である．つまり，「A ならば C」と「B ならば C」を示すのである．今回の elimination rule はそれをそのまま表したものだと言えよう．

2.4 具体的な型たち

3つの具体的な型を導入しながら，上で導入した型を実際に使っていく．

2.4.1 The empty type

empty type$\mathbf{0}$ は，空集合のようなものであり，rule は2つしかない．introduction rule がないというのが特徴で，それによって computation rule がないというわけだ．

$$\frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \mathbf{0} : \mathcal{U}}\ \mathbf{0}\text{-FORM} \qquad \frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash a : \mathbf{0}}{\Gamma \vdash \mathrm{rec}_{\mathbf{0}}(C, a) : C}\ \mathbf{0}\text{-ERIM}$$

つまり任意の $C : \mathcal{U}$ に対して，$\mathrm{rec}'_C : \mathbf{0} \to C$ が作れるということになる．

2.4.2 The unit type

unit type$\mathbf{1}$ は1点集合のようなものである．

$$\frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}}\ \mathbf{1}\text{-FORM} \qquad \frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \star : \mathbf{1}}\ \mathbf{1}\text{-INTRO}$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash c : C \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathrm{rec}_{\mathbf{1}}(C, c, a) : C}\ \mathbf{1}\text{-ELIM}$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash c : C}{\Gamma \vdash \mathrm{rec}_{\mathbf{1}}(C, c, \star) \equiv c : C}\ \mathbf{1}\text{-COMP}$$

informal には，

$$\mathrm{rec}'_{\mathbf{1},C} : C \to (\mathbf{1} \to C)$$

$$\mathrm{rec}'_{\mathbf{1},C}(c)(\star) \equiv c$$

となる．1点の行先さえ決めれば，function が作れるわけである．

Example 2.3. $\mathbf{2} :\equiv \mathbf{1} + \mathbf{1}$ として，この型の性質を見る．まず，

$$\mathrm{inl}(\star) : \mathbf{2} \qquad \mathrm{inr}(\star) : \mathbf{2}$$

がわかる．分かりやすいように，$\mathrm{inl}(\star) \equiv: 0_{\mathbf{2}}, \mathrm{inr}(\star) \equiv: 1_{\mathbf{2}}$ としよう．また，coproduct type の elimination rule,computation rule から，

$$\mathrm{rec}'_{\mathbf{2},C} : (\mathbf{1} \to C) \to (\mathbf{1} \to C) \to (\mathbf{2} \to C)$$

$$\mathrm{rec}'_{\mathbf{2},C}(g_1)(g_2)\,(\mathrm{inl}(a)) \equiv g_1(a) \qquad \mathrm{rec}'_{\mathbf{2},C}(g_1)(g_2)\,(\mathrm{inr}(b)) \equiv g_2(b)$$

がわかる．これと $\mathbf{1}$ の elimination rule,computation rule を組み合わせると，

$$\mathrm{rec}''_{\mathbf{2},C} : C \to C \to (\mathbf{2} \to C)$$

$$\mathrm{rec}''_{\mathbf{2},C}(c_1)(c_2)(0_{\mathbf{2}}) :\equiv c_1$$

$$\mathrm{rec}''_{\mathbf{2},C}(c_1)(c_2)(1_{\mathbf{2}}) :\equiv c_2$$

とわかり，まさに 2 点集合のような振る舞いをすることがわかる．

2.4.3 The natural number type

自然数といったらどのようなイメージを思い浮かべるだろうか．始まりがあって，そこから次へ次へというイメージであろう．よって，これをそのまま introduction rule にしてしまえば自然数の型は完成する．elimination rule は introduction rule で出てきたものの対応物を C において見つければ function ができるということをそのまま定式化するだけである．

$$\frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash \mathbb{N} : \mathcal{U}}\ \mathbb{N}\text{-FORM} \qquad \frac{\Gamma\,\mathrm{ctx}}{\Gamma \vdash 0 : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_1 \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{succ}(n) : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_2$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash c_0 : C \qquad \Gamma, x : C \vdash c_s : C \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{rec}_{\mathbb{N}}(C, c_0, x.c_s, n) : C}\ \mathbb{N}\text{-ELIM}$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash c_0 : C \qquad \Gamma, x : C \vdash c_s : C}{\Gamma \vdash \mathrm{rec}_{\mathbb{N}}(C, c_0, x.y.c_s, 0) \equiv c_0 : C}\ \mathbb{N}\text{-COMP}_1$$

$$\frac{\Gamma \vdash C : \mathcal{U} \qquad \Gamma \vdash c_0 : C \qquad \Gamma, x : C \vdash c_s : C \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{rec}_{\mathbb{N}}(C, c_0, x.y.c_s, \mathrm{succ}(n)) \equiv c_s\,[\mathrm{rec}_{\mathbb{N}}(C, c_0, c_s, n)/x] : C}\ \mathbb{N}\text{-COMP}_2$$

informal には，introduction rule は $0 : \mathbb{N}, \mathrm{succ} : \mathbb{N} \to \mathbb{N}$ と表せて，

$$\mathrm{rec}'_{\mathbb{N},C} : C \to (C \to C) \to (\mathbb{N} \to C)$$

$$\mathrm{rec}'_{\mathbb{N},C}(c_0)(c_s)(0) \equiv c_0$$

$$\mathrm{rec}'_{\mathbb{N},C}(c_0)(c_s)(\mathrm{succ}(n)) \equiv c_s\left(\mathrm{rec}'_{\mathbb{N},C}(c_0)(c_s)(n)\right)$$

となる．この elimination rule はまさに数学的帰納法を表している．

Example 2.4. 2 を足す関数 $\mathrm{add}(2) : \mathbb{N} \to \mathbb{N}$ を作ってみよう．まず 0 の行先を決めるが，それは当然 $2 :\equiv \mathrm{succ}(\mathrm{succ}(0))$ である．実は，c_s として succ を選ぶことによって

$$\mathrm{add}(2) :\equiv \mathrm{rec}'_{\mathbb{N},\mathbb{N}}(2)(\mathrm{succ})$$

と定めることができる．実際に確かめてみよう．

$$\begin{aligned}\mathrm{add}(2)(0) &\equiv \mathrm{rec}'_{\mathbb{N},\mathbb{N}}(2)(\mathrm{succ})(0)\\ &\equiv 2\end{aligned}$$

$$\begin{aligned}\mathrm{add}(2)(1) &\equiv \mathrm{rec}'_{\mathbb{N},\mathbb{N}}(2)(\mathrm{succ})(1)\\ &\equiv \mathrm{succ}\left(\mathrm{rec}'_{\mathbb{N},C}(2)(\mathrm{succ})(0)\right)\\ &\equiv \mathrm{succ}(2)\\ &\equiv 3\end{aligned}$$

このようにしっかりと「2 を足す」という関数になっていることが感じられるだろう．

3 Dependent types

型 A から $\mathcal{U}$ への function$B : A \to \mathcal{U}$ は何を表すだろうか．もし $a : A$ があるならば，$B(a) : \mathcal{U}$ を得る．つまり一つの型を得るわけだ．このように A の要素によって色々な型が現れることから，$B : A \to \mathcal{U}$ のことを type family と呼ぶ．これによる function type などの一般化が今回のテーマである．

3.1 Dependent function types

dependent function type とは，$A \to B$ の B が A の入力に依存する dependent type であることを許したものであり，$\prod_{x:A} B(x)$ とかく．これを命題としてみたときは，「任意の $x : A$ に対して，$B(x)$」という意味になる．以下，dependent function type の 5 つの rule である．

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash \prod_{x:A} B : \mathcal{U}}\ \prod\text{-FORM} \qquad \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).b : \prod_{x:A} B}\ \prod\text{-INTRO}$$

$$\frac{\Gamma \vdash f : \prod_{x:A} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}\ \prod\text{-ELIM} \qquad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x : A).b)(a) \equiv b[a/x] : B[a/x]}\ \prod\text{-COMP}$$

通常の function type とどこが変わったのかを考えてみてほしい．B が出てくるとき，常に $x : A$ が context にあることがわかるだろう．つまり B の中に x が含まれている可能性があるということだ．

では具体的にどのようなときに現れるだろうか．2.1 で $A : \mathcal{U} \vdash \lambda x.x : A \to A$ という結果を得たわけだが，dependent function type を使うと，

$$\vdash \lambda A.x.x : \prod_{A:\mathcal{U}} A \to A$$

と context が無くなり使いやすくなる（$\lambda A.(\lambda x.x)$ を $\lambda A.x.x$ と略記している）．

Example 3.1. 「任意の A, B, C に対して，$A \times B$ ならば $B \times A$」を示したい，となった場合，まずはこの命題を型の形にしよう．すると，

$$\prod_{A:\mathcal{U}} \prod_{B:\mathcal{U}} \prod_{C:\mathcal{U}} (A \times B) \to (B \times A)$$

となる．そしてこれを示すとはこれの要素を作ることに他ならない．とりあえず $A, B, C : \mathcal{U}$ を用意し，$(A \times B) \to (B \times A)$ の要素を作る．これは product type から伸びる function なので，elimination rule か

ら，$A \to (B \to (B \times A))$ の要素を作ればよい．

$$A, B, C : \mathcal{U}, x : A, y : B \vdash y : B \qquad x : A, y : B \vdash x : A$$

より，

$$A, B, C : \mathcal{U}, x : A, y : B \vdash (y, x) : B \times A$$
$$A, B, C : \mathcal{U} \vdash \lambda x.y.(y, x) : A \to (B \to (B \times A))$$

よってここから，

$$A, B, C : \mathcal{U} \vdash \mathrm{rec}'_{A\times B, B\times A}(\lambda x.y.(y, x)) : (A \times B) \to (B \times A)$$
$$\vdash \lambda A.B.C.\, \mathrm{rec}'_{A\times B, B\times A}(\lambda x.y.(y, x)) : \prod_{A:\mathcal{U}} \prod_{B:\mathcal{U}} \prod_{C:\mathcal{U}} (A \times B) \to (B \times A)$$

3.2 Dependent pair types

では function type$A \to B$ の B を dependent にしたように，product type$A \times B$ の B を dependent にしたものも考えてみよう．これを dependent pair type と言い,$B : A \to \mathcal{U}$ に対して，$\sum_{x:A} B(x)$ と書く．そして，これを命題としてみたときには，「$B(x)$ を満たす $x : A$ が存在する」という意味になる．以下４つの rule を書く．

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash \sum_{x:A} B : \mathcal{U}}\ \sum\text{-FORM}$$

$$\frac{\Gamma, x : A \vdash B : \mathcal{U} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \sum_{x:A} B}\ \sum\text{-INTRO}$$

$$\frac{\Gamma z : \sum_{x:A} B \vdash C : \mathcal{U} \qquad \Gamma, x : A, y : B \vdash g : C[(x, y)/z] \qquad \Gamma \vdash p : \sum_{x:A} B}{\Gamma \vdash \mathrm{rec}_{\sum_{x:A} B}(z.C, x.y.g, p) : C}\ \sum\text{-ELIM}$$

$$\frac{\Gamma : \sum_{x:A} B \vdash C : \mathcal{U} \qquad \Gamma, x : A, y : B \vdash g : C[(x, y)/z] \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash \mathrm{rec}_{\sum_{x:A} B}(z.C, x.y.g, (a, b)) \equiv g[a, b/x, y] : C[(a, b)/z]}\ \sum\text{-COMP}$$

product type の rule と比べてどこが変わったか見てほしい．B が context 内の $x : A$ に依存するようになったことに加えて，C が $z : \sum_{x:A} B$ に依存していることがわかるだろう．これはどういうことかというと，elimination rule とは元々「$\sum_{x:A} B \to C$ の要素をどのようにつくれるようにするか」であった．この function を dependent に一般化したのである．つまり elimination rule が「$\prod_{z:\sum_{x:A} B} C$ の要素をどのようにつくれるようにするか」に変わったわけだ．

Example 3.2. dependent pair type を使って monoid とは何か記述すると，それは後に出てくる型も使って，

$$\mathrm{Monoid} :\equiv \sum_{A:\mathrm{Set}} \sum_{\cdot:A\times A\to A} \sum_{e:A} \left(\prod_{x,y,z:A} (x \cdot y) \cdot z =_A x \cdot (y \cdot z) \right) \times \left(\prod_{x:A} x \cdot e =_A e \cdot x \right)$$

と書ける．例えば要素は $(\mathbb{N}, +, 0, p_1, p_2) : \mathrm{Monoid}$ という形になるが，これはその台集合，演算，単位元，諸公理の証明という必要な情報の塊になっているのである．

3.3 elmination rule,computation rule の dependent 化

先ほど dependent pair type にて elimination rule を dependent 用に変更した．これと同じ一般化を $A+B, \mathbf{0}, \mathbf{1}, \mathbb{N}$ でも行わなければならない．とはいっても，やることは C をその型の要素に依存させるだけなので，大した変更はない．

- $A+B$ に関して

$$\frac{\begin{array}{c}\Gamma, z : A+B \vdash C : \mathcal{U} \\ \Gamma, x : A \vdash c : C[\mathrm{inl}(x)/z] \qquad \Gamma, y : B \vdash d : C[\mathrm{inr}(y)/z] \qquad \Gamma \vdash e : A+B\end{array}}{\Gamma \vdash \mathrm{ind}_{A+B}(z.C, x.c, y.d, e) : C[e/z]}\ +\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, z : A+B \vdash C : \mathcal{U} \\ \Gamma, x : A \vdash c : C[\mathrm{inl}(x)/z] \qquad \Gamma, y : B \vdash d : C[\mathrm{inr}(y)/z] \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathrm{ind}_{A+B}(z.C, x.c, y.d, \mathrm{inl}(a)) \equiv c[a/x] : C[\mathrm{inl}(a)/z]}\ +\text{-COMP}_1$$

$$\frac{\begin{array}{c}\Gamma, z : A+B \vdash C : \mathcal{U} \\ \Gamma, x : A \vdash c : C[\mathrm{inl}(x)/z] \qquad \Gamma, y : B \vdash d : C[\mathrm{inr}(y)/z] \qquad \Gamma \vdash b : B\end{array}}{\Gamma \vdash \mathrm{ind}_{A+B}(z.C, x.c, y.d, \mathrm{inr}(b)) \equiv d[b/y] : C[\mathrm{inr}(b)/z]}\ +\text{-COMP}_2$$

- $\mathbf{0}$ に関して

$$\frac{\Gamma, x : \mathbf{0} \vdash C : \mathcal{U} \qquad \Gamma \vdash a : \mathbf{0}}{\Gamma \vdash \mathrm{ind}_{\mathbf{0}}(x.C, a) : C[a/x]}\ \mathbf{0}\text{-ERIM}$$

- $\mathbf{1}$ に関して

$$\frac{\Gamma, x : \mathbf{1} \vdash C : \mathcal{U} \qquad \Gamma, y : \mathbf{1} \vdash c : C[y/x] \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathrm{ind}_{\mathbf{1}}(x.C, y.c, a) : C[a/x]}\ \mathbf{1}\text{-ELIM}$$

$$\frac{\Gamma, x : \mathbf{1} \vdash C : \mathcal{U} \qquad \Gamma, y : \mathbf{1} \vdash c : C[y/x]}{\Gamma \vdash \mathrm{ind}_{\mathbf{1}}(x.C, y.c, \star) \equiv c[\star/y] : C[\star/x]}\ \mathbf{1}\text{-COMP}$$

- $\mathbb{N}$ に関して

$$\frac{\Gamma, x : \mathbb{N} \vdash C : \mathcal{U} \qquad \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x : \mathbb{N}, y : C \vdash c_s : C[\mathrm{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n) : C[n/x]}\ \mathbb{N}\text{-ELIM}$$

$$\frac{\Gamma, x : \mathbb{N} \vdash C : \mathcal{U} \qquad \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x : \mathbb{N}, y : C \vdash c_s : C[\mathrm{succ}(x)/x]}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, 0) \equiv c_0 : C[0/x]}\ \mathbb{N}\text{-COMP}_1$$

$$\frac{\begin{array}{c}\Gamma, x : \mathbb{N} \vdash C : \mathcal{U} \\ \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x : \mathbb{N}, y : C \vdash c_s : C[\mathrm{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, \mathrm{succ}(n)) \equiv c_s\,[\mathrm{ind}_{n,\mathbb{N}}(C, c_0, c_s, n)/x, y] : C[\mathrm{succ}(n)/x]}\ \mathbb{N}\text{-COMP}_2$$

そしてそれぞれの informal な表現は，

- $A+B$ に関して

$$\mathrm{ind}'_{A+B} : \prod_{C : A+B \to \mathcal{U}} \left(\prod_{x:A} C(\mathrm{inl}(x)) \right) \to \left(\prod_{y:B} C(\mathrm{inr}(y)) \right) \to \prod_{e:A+B} C(e)$$

$$\mathrm{ind}'_{A+B}(C)(g_1)(g_2)(\mathrm{inl}(a)) \equiv g_1(a) \qquad \mathrm{ind}'_{A+B}(C)(g_1)(g_2)(\mathrm{inr}(b)) \equiv g_2(b)$$

- $\mathbf{0}$ に関して

$$\mathrm{ind}'_{\mathbf{0}} : \prod_{C:\mathbf{0}\to\mathcal{U}} \prod_{a:\mathbf{0}} C(a)$$

- $\mathbf{1}$ に関して

$$\mathrm{ind}'_{\mathbf{1}} : \prod_{C:\mathbf{1}\to\mathcal{U}} C(\star) \to \prod_{x:\mathbf{1}} C(x)$$

$$\mathrm{ind}'_{\mathbf{1}}(C)(c)(\star) \equiv c$$

- $\mathbb{N}$ に関して

$$\mathrm{ind}'_{\mathbb{N}} : \prod_{C:\mathbb{N}\to\mathcal{U}} C(0) \to \left(\prod_{n:\mathbb{N}} C(n) \to C\left(\mathrm{succ}(n)\right)\right) \to \prod_{n:\mathbb{N}} C(n)$$

$$\mathrm{ind}'_{\mathbb{N}}(C)(c_0)(c_s)(0) \equiv c_0$$

$$\mathrm{ind}'_{\mathbb{N}}(C)(c_0)(c_s)\left(\mathrm{succ}(n)\right) \equiv c_s\left(\mathrm{ind}'_{\mathbb{N}}(C)(c_0)(c_s)\left(n\right)\right)$$

となる．最後の $\mathrm{ind}'_{\mathbb{N}}$ に注目してほしい．これは $\prod_{n:\mathbb{N}} C(n)$ を得たいがための function である．そしてこれを命題として見ると「任意の自然数 n に対して命題 $C(n)$」と読める．普段これを私たちは数学的帰納法で証明する．すなわち，$C(0)$ を示して，$C(n)$ を仮定した中で $C(\mathrm{succ}(n))$ を示すのだ．そして $C(0) \to (\prod_{n:\mathbb{N}} C(n) \to C\left(\mathrm{succ}(n)\right))$ の部分はまさにそれを表していると言えよう．

4 Identity types

ここでようやく propositional equal について扱う．前に述べた通り，propositional equal は $a =_A b$ という型のことであり，これを identity type という．なぜこれを最後に持ってきているかというと，実は今まで説明してきたことは，universe を除いてすべて通常の type theory の範囲内である．この identity type が HoTT の始まりとなっている．

4.1 Identity type の導入

さっそく rule を見ていこう．

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathcal{U}}\ \text{I-FORM} \qquad \frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{refl}_a : a =_A a}\ \text{I-INTRO}$$

$$\frac{\begin{array}{c}\Gamma, x : A, y : A, p : x =_A y \vdash C : \mathcal{U} \\ \Gamma, z : A \vdash c : C[z, z, \mathrm{refl}_z\,/x, y, p] \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B \qquad \Gamma \vdash p' : a =_A b\end{array}}{\Gamma \vdash \mathrm{ind}_{=_A}(x.y.p.C, z.c, a, b, p') : C[a, b, p'/x, y, p]}\ \text{I-ELIM}$$

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash C : \mathcal{U} \qquad \Gamma, z : A \vdash c : C[z, z, \mathrm{refl}_z\,/x, y, p] \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{ind}_{=_A}(x.y.p.C, z.c, a, a, \mathrm{refl}_a) \equiv c[a/z] : C[a, a, refl_a/x, y, p]}\ \text{I-ELIM}$$

A が自明なとき $x =_A y$ を $x = y$ と略記することが多い．また informal な形として，以下の function または式が得られるわけである．

$$\mathrm{ind}'_{=_A} : \prod_{C:\prod_{x,y:A} x=_A y\to\mathcal{U}} \left(\prod_{z:A} C(z, z, \mathrm{refl}_z)\right) \to \prod_{x,y:A} \prod_{p:x=y} C(x, y, p)$$

$$\mathrm{ind}'_{=_A}(C)(c)(a)(a)(\mathrm{refl}_a) \equiv c(a)$$

今回の elimination rule も $C(x,y,p)$ を命題として見れば意味がわかりやすい．$C(x,y,p)$ は A の 2 つの要素とその等しさに依存している．そしてこれから，$x=y$ が「x と y が等しい」というように意味付けしたい．そこで，もし $C(z,z,\mathrm{refl}_z)$ が正しいなら，$C(x,y,p)$ も正しいと言えることにしてしまおう，というのが今回の rule である．

Example 4.1. 実際に rule を使っていこう．propositional equal の対称律を示す．つまり $p : x=y$ のもとで $y=x$ の要素をつくるわけだ．このとき elimination rule を使うにあたって，$C(x,y,p)$ として $y=x$ を考えると，

$$\mathrm{ind}'_{=}(y=x) : \left(\prod_{z:A} z=z\right) \to \prod_{x,y:A}\prod_{p:x=y} y=x$$

となり，

$$\mathrm{ind}'_{=}(y=x)(\lambda z.\,\mathrm{refl}_z) : \prod_{x,y:A}\prod_{p:x=y} y=x$$

が得られる．

4.2 homotopy 的解釈

言葉の関係などで先に少し説明した方がイメージしやすくなると思うので，ここで homotopy 的な解釈の対応関係を見せる．その対応関係とは以下の通りである．これによって，左の型理論の対象を右の言葉で呼ぶ

型理論	homotopy 論
型 A	a space
$a,b : A$	points
$p,q : a =_A b$	paths
$r,s : p =_{a=b} q$	paths between the paths
$t,u : r =_{p=q} s$	paths between the paths between the paths
$\vdots$	$\vdots$

ことがある．例えば $a=b$ の要素は path と呼ばれ identity type の elimination rule と computation rule は合わせて path induction と呼ばれる．

このような対応から HoTT は生まれた．現在は数学の基礎として HoTT を見ているので，そのような analogy がある程度の認識で良いが，第二の側面を考えたときに，この対応が主役となる．

5 Higher inductive types

ここではあまり詳しいことまでは触れない．詳しくは [11] の chapter 2,5,6 を参照してほしい．

5.1 型の構成を振り返って

型を構成するためには主に 4 つの rule があった．もう一度振り返ってみよう．

- formation rule：どのようにその型が導入されるか
- introduction rule：どのようにその型の要素を作れるか
- elimination rule：その要素を使って何が得られるか
- computation rule：その得たものはどのように計算できるか

ここで改めて伝えたいことは何かというと，「elimination rule と computation rule は introduction rule が決まれば自動的に決まる」ということである．これは既に現れた型（最も基本となる $\prod_{x:A} B$ と $\sum_{x:A} B$ を除いて）にも言えることである．どういうことかというと，elimination rule は，要素をどう使うか，すなわちその型から任意の型 C への function をどう作れるようにするかということであったわけだが，その function の定め方というのは，introduction rule で与えられた要素にあたるものを C で指定すれば良いのではないかということだ．例えば，$A+B$ の introduction rule で与えられた要素は，$a:A, b:B$ に対して $inl(a)$ と $inr(b)$ であった．だからこの 2 つにあたる C の要素を指定すれば C への function が決まる．実際，elimination rule でも，その 2 つが必要な条件になっているはずだ．$\mathbb{N}$ においても同じだ．introduction rule で与えられたのは 0 と $succ$ であり，C におけるそれとは c_0 と c_s である．

このように introduction rule がその型の情報の大部分を占めていることがわかった．よって，introduction rule のことをもっと簡単に constructor と呼ぶ．またそれによって作られる elimination rule，computation rule のことをまとめて induction principle という．つまりここでの主張は「constructor さえ決めれば型は決まる」ということになる．

5.2 Higher inductive types

今まで出てきた具体的な型の constructor を見直してみると，$\mathbf{0}$ はなし，$\mathbf{1}$ は $\star$，$\mathbb{N}$ は 0 と $succ : \mathbb{N} \to \mathbb{N}$ だった．これらを単に inductive type と呼ぶ．inductive というのは，例えば $\mathbb{N}$ においては $succ$ があるので帰納的に要素が定まっていく感じがするだろう．ではどんなときに higher inductive type になるのかというと，それは「constructor に path（identity type の要素）が現れる」ときである．

Example 5.1. 例として整数の型 $\mathbb{Z}$ を constructor を決めて導入してみよう．$\mathbb{N}$ と同様に何が必要か考えれば自然とわかる．整数は前だけに進む自然数に加えて後ろにも進めるというイメージがある．よって，それをそのまま表現すると，まず $0 : \mathbb{Z}$ と $\mathrm{succ} : \mathbb{Z} \to \mathbb{Z}$．そしてこれに $\mathrm{succ}' : \mathbb{Z} \to \mathbb{Z}$ を加える．しかし，これだけではダメで，「後ろ向きに」という意味で succ' を加えたいわけなので，$p : \mathrm{succ}\,(\mathrm{succ}'(0)) = 0$ と $q : \mathrm{succ}'\,(\mathrm{succ}(0)) = 0$ が必要になる．まとめると，

- $0 : \mathbb{Z}$
- $\mathrm{succ} : \mathbb{Z} \to \mathbb{Z}$
- $\mathrm{succ}' : \mathbb{Z} \to \mathbb{Z}$
- $p : \mathrm{succ}\,(\mathrm{succ}'(0)) =_{\mathbb{Z}} 0$
- $q : \mathrm{succ}'\,(\mathrm{succ}(0)) =_{\mathbb{Z}} 0$

これらの constructor から型 $\mathbb{Z}$ は作られる．すなわち $\mathbb{Z}$ は higher inductive type である．

ただひとつ注意点として，higer inductive type の elimination rule を作るとき dependent を許すと，C における path を見つける部分が少しややこしくなるので，そう単純にはいかない（[11] の chapter 6 参照）．

6 Univalence

ここでは function が等しいとはどういうことか，型が等しいとはどういうことかということを考えていく．ここでも homotopy theory との analogy を見るのが分かりやすい．まるで型が空間のように扱われている．詳しくは [11] の chapter 2 を参照してほしい．

6.1 Equivalences

型 A, B を空間と見たならば，function$f, g : A \to B$ は連続写像にあたる．このとき homotopy にあたるものは以下になる．

$$(f \sim g) :\equiv \prod_{x:A} (f(x) =_B g(x))$$

各点で道が通っているような感じだ．これをそのまま homotopy と呼ぶ．正確にはもっと一般的に dependent function に対して定めて良い．つまり $f, g : \prod_{x:A} B(x)$ に対して，

$$(f \sim g) :\equiv \prod_{x:A} \big(f(x) =_{B(x)} g(x)\big)$$

を homotopy という．

では homotopy を使って，型の equivalence を定める．先に定義を書くと，まず，$f : A \to B$ に対して

$$\mathrm{isequiv}(f) :\equiv \left(\sum_{g:B\to A} (f \circ g \sim \mathrm{id}_B)\right) \times \left(\sum_{h:B\to A} (h \circ f \sim \mathrm{id}_A)\right)$$

として，

$$(A \simeq B) :\equiv \sum_{f:A\to B} \mathrm{isequiv}(f)$$

とする．$A \simeq B$ を equivalence という．それでは意味を見ていこう．まず isequiv(f) とは「f に（広義で）逆写像がある」という解釈ができ，そうすると，equivalence の定義が自然に思えると思う．つまり，A から B への function$f$ が存在し，それが逆をもつとき，A と B は equivalence であるとしようというわけだ．この equivalence は，型を命題だと思ったときには同値性になり，型を代数だと思ったときには同型になったりすることがなんとなくわかるだろう．

6.2 The function extensionality axiom

では function が等しいとはどういうことかを設定していく．実は今まで導入した rule で以下の要素 happly は作れる．

$$\mathrm{happly}_{f,g} : (f = g) \to (f \sim g)$$

そこで逆も言えることにしてしまおうというのが the function extensionality axiom である．その rule は，

$$\frac{\Gamma \vdash f : \prod_{x:A} B \qquad \Gamma \vdash g : \prod_{x:A} B}{\Gamma \vdash \mathrm{funext}(f, g) : \mathrm{isequiv}(\mathrm{happly}_{f,g})}\ \Pi\text{-EXT}$$

と設定できる．つまり $(f = g) \simeq (f \sim g)$ がいえる．これによって，HoTT において function が等しいとは各点で等しいことであるという設定ができたわけである．

6.3 The univalence axiom

では型として等しいとはどういうことにするのか．ここでも，今まで導入した rule から以下の要素 idtoeqv は作れる．

$$\mathrm{idtoeqv}_{A,B} : (A =_{\mathcal{U}} B) \to (A \simeq B)$$

そこでここでもこれの逆が存在するとしてしまう，それが univalence axiom なのだ．その rule は当然

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash \mathrm{univalence}(A,B) : \mathrm{isequiv}(\mathrm{idtoeqv}_{A,B})}\ \mathcal{U}\text{-UNIV}$$

となる．すなわち HoTT では equivalence が言えたら型として等しいことになるのだ． これが理にかなっていることの確認として例を挙げる．普段の数学で，2つの群が同型だったとき，それらは集合としては全く違えど，群として同じとして扱うだろう．集合も対象式としての記述が全く違えど，全単射が存在すれば同じとみなす．つまり，普段の数学では，実際には違うけど同じとみなすということが頻繁にある．そしてこれが HoTT 上の話となれば，同じとみなしていたものは実際に同じになるのだ．それが univalence axiom の言っていることなのだ．

第 II 部
Classifying category

ここでは type theory の数学全体の基礎としてではない第二の側面について解説する．その前にまずは I では type theory の中でも特殊な HoTT を扱ったわけなので，一般に type theory とは何なのか考える．

7 Type theory とは何か

まず type theory とは何であるか考える．type theory は rule の集まりでできている．すなわちどれを rule として採用するかで違いがでるわけだ．まずどんな type theory も 1.2 で紹介した基本的な rule は持っている（universe を除いて）としよう．そしたら，あとはどのような型の構成を許すかで様々な type theory ができる．これが一般的な高い自由度を持つ type theory のイメージになる． ただ「どんな type theory も 1.2 で紹介した基本的な rule は持っている」と言ったが，正確には違う．実は universe を HoTT の枠組みの中では基本的なので，最初に導入したが，もっと広い意味で type theory を考えるとき，その階層的な universe があることはかなり特殊である．すなわちまずは，階層的な universe 抜きで基礎づけすべきなのだ．

とはいってもやることは至って簡単である．まず judgement の種類を増やす．HoTT では universe があったから $\Gamma \vdash a : A$ で型の構成も要素の構成も一気に済ませていたが，universe がない前提では，これを

$$\Gamma \vdash A\,\mathrm{type} \qquad \Gamma \vdash a : A$$

の 2 つに分ける．そうして，他の rule の「$A : \mathcal{U}$」の部分を「$A\,\mathrm{type}$」に変えれば良い．

こうして universe 抜きの基礎を作れた．universe 抜きに変更した 1.2 で導入した基本的な rule たちのみを持つ type theory を $\mathbb{T}_0$ とする．またそこでは dependent type を禁ずる．つまり $\Gamma \vdash A\,\mathrm{type}$ の A は Γ の変数に依存してはならない．そして，型 X が構成できるような rule を $\mathbb{T}_0$ に追加してできた type theory を $\mathbb{T}_0(X)$ と書くことにする（product を追加した場合などは $\mathbb{T}_0(\times)$ とする）．まとめると，以降，type theory と言ったら，$\mathbb{T}_0$ にいくつかの型の構成を加えたもので dependent type を禁じられているとする．

8 Classifying category の作り方

実は，type theory$\mathbb{T}$ は，とある圏 $C_{\mathbb{T}}$ を作る．このとき，$C_{\mathbb{T}}$ を $\mathbb{T}$ の classifying category といい，$\mathbb{T}$ を $C_{\mathbb{T}}$ の internal language という．ではこれから classifying category がどのように作られるのかを説明しよう．４つの設定から $C_{\mathbb{T}}$ はできる．まず最初の２つについて説明する．

$\mathbb{T}_0$ では rule によってたくさんの judgement が生成されていく．その judgement の種類のうち以下の２つがあった．

$$\Gamma\,\mathrm{ctx} \qquad \Gamma \vdash a : A$$

ただし，左の judgement が得られたとき，私たちは object として $[\![\Gamma]\!]$ を得る（設定１）．そして右の judgement が得られたとき，私たちは morphism として $[\![\Gamma]\!] \xrightarrow{[\![a]\!]} [\![A]\!]$ を得る（設定２）．$\Gamma \vdash A$ がある時点で既に $(x : A)\,\mathrm{ctx}$ という judgement は獲得できているわけだ（$[\![x : A]\!]$ を $[\![A]\!]$ と略記した）．これらの設定から，得られる具体的な結果としては，$[\![x_1 : A_1, \cdots, x_n : A_n]\!]$ という object があったとき，rule として，

$$\frac{(x_1 : A_1, \cdots, x_n : A_n)\,\mathrm{ctx}}{x_1 : A_1, \cdots, x_n : A_n \vdash x_i : A_i}\ \textsc{Vble}$$

があるため, 任意の $i = 1, \cdots, n$ に対して morphism$[\![A_1, \cdots, A_n]\!] \xrightarrow{[\![x_i]\!]} [\![A_i]\!]$ が存在する．この morphism を π_i とする．ここで 3 つ目の設定として，$[\![A_1, \cdots, A_n]\!]$ が $[\![A_1]\!], \cdots, [\![A_n]\!]$ の product になるように morphism を追加する．このときの projection は π_i である．また空の context である $[\![\ \ ]\!]$ が終対象となるように morphism を追加する．

そして 4 つ目の設定として，$[\![b_1]\!], [\![b_2]\!] : [\![A]\!] \to [\![B]\!]$ のとき，$x : A \vdash b_1 \equiv b_2 : B$ ならば，$[\![b_1]\!]$ と $[\![b_2]\!]$ は morphism として等しいとする．

以上の 4 つの設定から，$C_{\mathbb{T}}$ は作られる．合成などの公理を満たすことは 1.2 で導入した基本となる rule から導ける．

9 普遍性

$\mathbb{C}_{\mathbb{T}_0}$ と $\mathbb{C}_{\mathbb{T}_0(X)}$ を比べると，後者は $[\![X]\!]$ という object を持つわけだが，このとき，実はこの object はとある普遍性によって特徴づけられる．そしてその普遍性は X の rule にそのまま記述されている．このことを例 $A + B$ を通して確認していこう． $[\![A]\!], [\![B]\!]$ という 2 つの object があるとして，$[\![A + B]\!]$ がどんな object なのかということを考えていく．まず introduction rule を見直してみよう．

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathrm{inl}(a) : A + B}\ +\text{-INTRO}_1 \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash \mathrm{inr}(b) : A + B}\ +\text{-INTRO}_2$$

左を圏の話に置きかえると，写像 $\mathrm{Hom}_{C_{\mathbb{T}}}([\![\Gamma]\!], [\![A]\!]) \to \mathrm{Hom}_{C_{\mathbb{T}}}([\![\Gamma]\!], [\![A + B]\!])$ の存在を表している．そして米田の補題より，これに対応した morphism$[\![A]\!] \xrightarrow{\mathrm{inl}} [\![A + B]\!]$ を得る．右からも同様に $[\![B]\!] \xrightarrow{\mathrm{inr}} [\![A + B]\!]$ を得る．次に elimination rule を見直す．

$$\frac{\Gamma \vdash C\,\mathrm{type} \qquad \Gamma, x : A \vdash c : C \qquad \Gamma, y : B \vdash d : C \qquad \Gamma \vdash e : A + B}{\Gamma \vdash \mathrm{rec}_{A+B}(C, x.c, y.d, e) : C}\ +\text{-ELIM}$$

Γ を empty にすると，

$$\frac{\vdash C\,\mathrm{type} \qquad x : A \vdash c : C \qquad y : B \vdash d : C \qquad \vdash e : A + B}{\vdash \mathrm{rec}_{A+B}(C, x.c, y.d, e) : C}\ +\text{-ELIM}$$

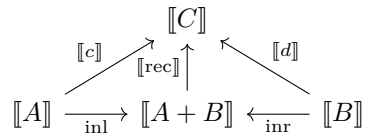
となり, $[\![A]\!] \xrightarrow{[\![c]\!]} [\![C]\!]$ と $[\![B]\!] \xrightarrow{[\![d]\!]} [\![C]\!]$ に対して, $[\![A+B]\!] \xrightarrow{\mathrm{rec}_{A+B}(C,x.c,y.d,e)} [\![C]\!]$ を得る．最後に computation rule を見ると（Γ を empty にした），

$$\frac{\vdash C\,\mathrm{type} \qquad x : A \vdash c : C \qquad y : B \vdash d : C \qquad \vdash a : A}{\vdash \mathrm{rec}_{A+B}(C, x.c, y.d, \mathrm{inl}(a)) \equiv c[a/x] : C}\ +\text{-COMP}_1$$

$$\frac{\vdash C\,\mathrm{type} \qquad x : A \vdash c : C \qquad y : B \vdash d : C \qquad \vdash b : B}{\vdash \mathrm{rec}_{A+B}(C, x.c, y.d, \mathrm{inr}(b)) \equiv d[b/y] : C}\ +\text{-COMP}_2$$

となり，圏の話に置き換えると，合成 $[\![\ \ ]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \xrightarrow{[\![c]\!]} [\![C]\!]$ と合成 $[\![\ \ ]\!] \xrightarrow{[\![a]\!]} [\![A]\!] \xrightarrow{\mathrm{inl}} [\![A + B]\!] \xrightarrow{[\![\mathrm{rec}_{A+B}(C,x.c,y.d,e)]\!]} [\![C]\!]$ が等しいことがわかり，$[\![\ \ ]\!] \xrightarrow{[\![a]\!]} [\![A]\!]$ が monomorphism であることから，$[\![A]\!] \xrightarrow{[\![c]\!]} [\![C]\!]$ と $[\![A]\!] \xrightarrow{\mathrm{inl}} [\![A + B]\!] \xrightarrow{[\![\mathrm{rec}_{A+B}(C,x.c,y.d,e)]\!]} [\![C]\!]$ が等しいことがわかる．同様に $[\![B]\!] \xrightarrow{[\![d]\!]} [\![C]\!]$ と

$[\![B]\!] \xrightarrow{\text{inr}} [\![A+B]\!] \xrightarrow{[\![\text{rec}_{A+B}(C,x.c,y.d,e)]\!]} [\![C]\!]$ が等しい．



これらの rule がそのまま $[\![A+B]\!]$ の coproduct としての普遍性になっていたことがお分かりいただけただろうか[*3]．他の型もそれぞれ対応する普遍性がある．

型	普遍性
coproduct types	binary coproducts
the empty type	initial object
product types	binary products
the unit type	terminal object
the natural number type	natural number object
identity types	diagonal/equalizer
function types	exponential object
dependent pair types	left adjoint to pullback
dependent function types	right adjoint to pullback
the proposition type	subobject classifier
the universe type	object classifier

10　新たな数学観

現代の数学では，とある数学的対象を考えるときそれが属する圏を意識する姿勢が出てきた．それはつまり，その数学的対象の住む世界がその圏となっていると言えよう．また普段の数学では基礎理論があり，そこから全数学を展開していくという「一本柱」のような展開が普通だ．これを analtic mathematics と呼ぶ．つまり集合で全数学的対象を測っていくわけだ．しかし，圏による見方が台頭することで，別の可能性が見えてきた．それが「たくさんの数学の世界」があるというイメージである．ここでいう数学の世界というのはもちろん圏のことである．そして，それらを記述する言語として，その圏の internal language が出てくるわけである．

まとめると，従来の数学は「１つの基礎理論があり，そこから１つの数学の世界が展開される」わけであったが，新しい数学観として，「圏としてのたくさんの数学の世界があり，それぞれに internal language としての type theory がある」というものが出てきた．正確には，任意の圏に対して internal language を求めるというよりかは，classifying category から来る functor が主役になっている（詳しくは [8]）．

ではこのような目線に立つメリットは何か．それは考えたい世界を直接的に扱えるということである．これが analtic mathematics に対する synthetic mathematics の姿勢である．例えば，とある空間を調べたいとなったときに，普段の数学では，集合があり，そこに位相構造を定めて，そこからやっと幾何的な操作ができ

[*3] ただ存在する射の一意性についてはここでは言及できない．

るようになる．しかし，synthetic mathematics では空間の住む圏とその internal language を用意し，直接その空間について議論をすることができる．この姿勢は，考えたい幾何の対象によって，synthetic topology や synthetic homotopy theory，synthetic differential geometry，synthetic algebraic geometry などがあり，HoTT はこの synthetic homotopy theory をやるための言語だったのである．これが HoTT の第二の側面である．

第 III 部
行われている研究，文献の紹介

- HoTTbook と呼ばれる HoTT の入門書 [11] がある．I の参考文献は主にこの chapter 1,2 である．この本は基礎論の人向けではなく，もっと一般の数学徒，数学者に対して，informal な数学の基礎としての HoTT の導入を目指して書かれたものであり，予備知識がほとんどない．synthetic homotopy theory についても書かれている．
- 私の大好きな文献として [8] がある．これは II の主な参考文献となっている．特に一番最初の「大抵の数学的対象は幾何的な構造と共に導入されているのに，集合論をベースにするなんてナンセンスだ！」という主張は，私が待ち詫びていたものであり，これで Shulman のファンになった．他にも様々な名言がこの文献の中にはあった．
- $\mathbb{T}_0$ に対する classifying category を紹介したわけだが，では HoTT はどのような圏を作るのかというのは気になるところである．そしてこれを考えることが ∞-category への扉なのである ([9],[6])．実際，morphism が等しいことを，propositional equal で定義すると，morphism の間の morphism，morphism の間の morphism の間の morphism という拡張ができて，これがまさに category から ∞-category への拡張と一致している．[5]
- homotopy 的な空間と topological な空間をつなげる目的で cohesive homotopy type theory というものが作られている（[7],[8],[4]）．
- synthetic algebraic geometry[3]．ものすごくやりたい．
- 実は型理論は日常言語との繋がりもある [2]．つまり，日常言語の一般的な文法システムを型理論で整理できるというものだ．これが数学における型理論の基礎としての自然さを裏づけるかもしれない．
- また type theory の一般的な枠組みについて整理した文献もある [10]．

参考文献

[1] N. Bourbaki. 集合論: 1. ブルバキ数学原論. 東京図書, 1968.

[2] Stergios Chatzikyriakidis and R. Cooper. Type theory for natural language semantics. 2018.

[3] Felix Cherubini, Thierry Coquand, and Matthias Hutzler. A Foundation for Synthetic Algebraic Geometry. *arXiv e-prints*, p. arXiv:2307.00073, June 2023.

[4] David Jaz Myers and Mitchell Riley. Commuting Cohesions. *arXiv e-prints*, p. arXiv:2301.13780, January 2023.

[5] E. Riehl. *Category Theory in Context*. Aurora: Dover Modern Math Originals. Dover Publications,

2017.

[6] Emily Riehl. On the ∞-topos semantics of homotopy type theory. *arXiv e-prints*, p. arXiv:2212.06937, December 2022.

[7] Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *arXiv e-prints*, p. arXiv:1509.07584, September 2015.

[8] Michael Shulman. Homotopy type theory: the logic of space. *arXiv e-prints*, p. arXiv:1703.03007, March 2017.

[9] Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes. *arXiv e-prints*, p. arXiv:1904.07004, April 2019.

[10] Taichi Uemura. A General Framework for the Semantics of Type Theory. *arXiv e-prints*, p. arXiv:1904.04097, April 2019.

[11] The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. *arXiv e-prints*, p. arXiv:1308.0729, August 2013.

[12] 斎藤毅. 数学原論. 東京大学出版会, 2020.