9　計算体系と論理体系の関係

本講義の主題は，プログラミング言語に関する研究と形式論理に関する研究が密接に関連することを学ぶことであった．本章では，いよいよ，計算体系と論理体系の関係を解き明かす．

9.1　型付きラムダ計算と直観主義命題論理の関係

これまでの演習を通して，型付きラムダ計算の体系と，直観主義命題論理の体系が極めて似ていることに気付いただろう．

たとえば，$A \to B$ という型は，$A \supset B$ という命題と非常によく似た規則を持っている．

- $\to$ の導入規則 (λ) と $\supset$ の導入規則 ($\supset I$)

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}\ \lambda \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset I$$

- $\to$ の除去規則 ($apply$) と $\supset$ の除去規則 ($\supset E$)

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ apply \quad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \supset E$$

計算体系の $M : A$ という形から $M :$ を取り除けば論理体系の対応する規則になることがわかる．

同じことは，$\times$ の規則 ($pair, left, right$) と $\wedge$ の規則 ($\wedge I, \wedge EL, \wedge ER$)，$+$ の規則 ($inl, inr, case$) と $\vee$ の規則 ($\vee IL, \vee IR, \vee E$) についても言える．また，変数の規則 (var) は仮定を導入する規則 ($assume$) に対応する．全ての場合において，計算体系における $M : A$ という形から項の部分 $M :$ を取り除けば論理体系の対応する規則になることがわかる．

これは偶然であろうか？そうではない．次章で見るようにこれは非常に本質的な現象である．ここでは，その準備として，計算体系と論理体系に若干の (非本質的な) 変更を加える．

- 型付きラムダ計算の体系に，空の型 (要素を持たない型) として $\bot$ を導入する．
  $\bot$ は以下の規則 ($\bot$ 型の除去規則) だけを持つ型である．

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash abort(M) : A}\ abort$$

  直感的には，$\bot$ 型は「空の型」であり，要素を持たないものである．したがって，もし $\bot$ 型の値を計算する項 M があったとすれば，Γ 自体が矛盾した仮定であるので，そこからどんな型 A の要素も計算できてしまえる，というのがこの規則の意味である．このような型を導入しても，計算体系は本質的には何も変わらない，ということが直感的に想像されるであろう．すなわち，$abort$ を含む項 (プログラム) を実行したとき，$abort(M)$ という部分は決して実行されない (その部分に制御が行くことはない)．要素を持たない型は存在意義がないと考えるかもしれないが，Java 言語の try-catch, ML 言語の exception などで表現される例外機構の記述では有用である．それについては，「古典論理への拡張」の節で述べる．
- 直観主義命題論理の体系で $\neg$ は省略形と考える．すなわち，$\neg A$ は $A \supset \bot$ の省略形と見なし，$\neg$ という記号はないと考える．したがって，$\neg$ に関する 2 つの規則 $\neg I$ と $\neg E$ もないものと考える．

9.2 Curry(-Howard) の同型対応

Curry-Howard の同型対応は，「命題＝型」原理 (Propositions-as-Types Principle) とも呼ばれ，命題と型が本質的に同じであることを主張するものである．しかし，それは，単に命題と型が同じ導出規則で生成されるという静的な性質の対応だけではない．命題の導出と型の導出とが対応し，さらに，命題の導出に対する計算と型の導出に対する計算が対応する，という動的な性質の対応を含むものである．

ここで，「命題の導出に対する計算」とは何であろうか？それは前章で見た「無駄を省く変形操作」のことである．では，「型の導出に対する計算」とは何であろうか？型の導出とは，何らかの項がある型を持つ，ということの導出のことであり，「型つきラムダ項の導出」と呼んでいたものであった．つまり，「型の導出に対する計算」とは，型つきラムダ項の計算に他ならない．

以上まとめると以下の表を得る．

論理体系	計算体系
命題	型
$\supset$	$\rightarrow$
$\wedge$	$\times$
$\vee$	$+$
$\bot$	$\bot$
命題の導出 ($\vdash A$ の導出)	型の導出 ($\vdash A'$ の導出)
$\supset I$ 規則, $\supset E$ 規則	$lambda$ 規則, $apply$ 規則
$\wedge I$ 規則, $\wedge EL$ 規則 $\wedge ER$ 規則	$pair$ 規則, $left$ 規則, $right$ 規則
$\vee IL$ 規則, $\vee IR$ 規則, $\vee E$ 規則	inl 規則, inr 規則, $case$ 規則
$\bot E$ 規則	$abort$ 規則
この導出に対する変形操作 (無駄を省く変形)	この導出の計算 (項の計算)
$\supset I - \supset E$ 変形	β 計算規則
$\wedge I - \wedge EL$ 変形	pair-left 計算規則
$\wedge I - \wedge ER$ 変形	pair-right 計算規則
$\vee IL - \vee E$ 変形	case-inl 計算規則
$\vee IR - \vee E$ 変形	case-inr 計算規則

この表の意味するところは以下の通りである．

- 命題 A は，それにあらわれる $\supset, \wedge, \vee, \bot$ をそれぞれ $\rightarrow, \times, +, \bot$ に読みかえることにより型 A' に対応する．
- A の導出 (正確には，$\vdash A$ という判断の導出) は，その中に現れる規則を，表の 2 番目に列挙した対応関係 ($\supset I$ を λ 規則で置きかえ，$\supset E$ を $apply$ 規則で置きかえる，等) で置きかえることにより，A' の導出 (正確には，適当な項 M に対して $\vdash M : A'$ の導出) になる．
- A の導出に対する計算 (変形操作の系列) は，各変形操作を，表の 3 番目に列挙した対応関係 ($\supset I - \supset E$ 変形を，β 計算規則で置きかえる，等) で置きかえることにより，A' の導出に対する計算となる．

すなわち，直観主義論理の命題の導出と計算は，型付きラムダ計算の項の導出と計算に完全に対応する．これらは，片方が与えられれば他方が機械的に (一意的に) 定まる，という全単射で対応付けられている．ところで，計算体系にしても論理体系にしても，我々は形式的な構文と導出のみで定まるものとしてきたので，これらが完全に対応する，ということは，直観主義論理と型付きラムダ計算の体系は同じものである，ということになる．(もちろん，最初にこれらの体系を構築した際に意図した意味論は違うものであったはずだが，結果として形式化されたものはまったく同じである．)

これが Curry-Howard の同型対応である．Curry-Howard の同型対応は，計算と論理の形式的体系の間の深い関係を示すものである．なお，歴史的には，ここで述べたような命題論理の対応は Curry が発見し，後に少々触れる述語論理まで拡張した際の対応を Howard が発見したので，本節で述べた範囲での対応は正確には「Curry の同型対応」というべきである．

9.3 Curry-Howard の同型対応の周辺

前節の結果は理論的には美しい結果であるが，ただちにいくつかの疑問がわくだろう．

- そもそも何故 Curry-Howard の同型対応がなりたったのであろうか？
- Curry-Howard の同型対応が発見されたからといって，何が嬉しいのだろうか？ 2 つの独立に作られたものが同じことがわかると，何か本質的な進歩があるのだろうか？
- 型付きラムダ計算に対応するのは，古典論理ではなく，直観主義論理である．我々の日常的推論が古典論理でおこなわれていることを考えると，直観主義論理と対応することの意味は何だろうか？
- 形式的体系がたまたま同じだからといって，それは形式化のやり方に依存することであって，「計算」と「論理」が本質的に同じという結論を導いていいのだろうか？それぞれの意味も同じといっていいのだろうか？

これらの疑問について以下では考えてみる．

9.3.1 なぜ対応するのか？

直観主義論理の形式的体系に「ぴったり一致する」意味論として，構成的解釈があることを既に学んだ．これを使って，Curry-Howard 同型対応が「必然的」なものであることが説明できる．

- 構成的解釈において $A \supset B$ という命題の証拠は，「A の証拠をもらって B の証拠を返す関数」であると定められた．「命題 A の証拠」を「命題 A に対応する型 A' に族する要素」という対応関係でみると，「A の証拠をもらって B の証拠を返す関数」は「A 型の要素をもらって B 型の要素を返す関数」であり，そのような関数を集めた型は，ちょうど $A \to B$ という型になる．
- 構成的解釈において $A \wedge B$ の証拠は「A の証拠と B の証拠の対」であった．これを「命題 A の証拠」を「命題 A に対応する型 A' に族する要素」という対応関係でみると，「A' 型の要素と B' 型の要素の対」であり，これを集めた型は，ちょうど $A \times B$ という型になる．
- 構成的解釈において $A \vee B$ の証拠は，「A であるか B であるかをあらわす 1bit の情報と，A の証拠もしくは B の証拠の対」であった．これを「命題 A の証拠」を「命題 A に対応する型 A' に族する要素」という対応関係でみると，「A' 型であるか B' 型であるかを決める 1bit の情報」と，「A' 型の要素，もしくは，B' 型の要素」の対になる．これを集めた型は，ちょうど $A + B$ という型になる．

- 命題 $\bot$ には証拠がないが，これは，要素を持たない型 $\bot$ に対応している．

言い換えると，構成的解釈を表現するのに必要なだけの表現力をもった計算体系が型付きラムダ計算であるといえる．このように考えると，直観主義論理と型付きラムダ計算が対応するのは，むしろ当然のことと言える．

9.3.2 なにが嬉しいのか？

一般に，全く異なる生い立ちをもつ 2 つの理論の関係が解き明かされると，新たな発展が期待される．Curry-Howard の同型対応でも事情は同じであり，計算体系と論理体系で別々に知られていた事実の間の関係がわかったり，片方の結果を他方に移すことで新たな展開が知られたりする．このような観点を基礎とする研究は非常に盛んであり，**プログラミング言語論の研究の中心的テーマの 1 つ**である．

この章の後の方で，そのような話題を 4 つ紹介する．

9.3.3 直観主義論理と対応したことの意味は何か？

直観主義論理は，構成的解釈に対応する形式的体系であることからわかるように「計算」によって論理を解釈しようとする体系である．したがって，このような論理が計算体系と対応したからといって，あまり驚きはないかもしれない．いずれにせよ，**「直観主義論理＝計算の論理」**である．

では，純粋な論理として見たとき，直観主義論理はどんなものであろうか？既に見てきたように，本講義のような定式化をする限り，直観主義論理の証明は古典論理の証明よりはるかに自然である．命題 A が導出可能なとき，命題 A の部分命題しかあらわれないような導出が必ず存在する．このような論理は，純粋な論理体系としても非常に興味深いものであることが想像できよう．

さらに，近年の数学の流れとして，構成的な存在証明への関心があげられる．存在証明とは，$\exists x.A(x)$ という形の定理の証明のことである．一般に，数学の定理は排中律等を多用して証明されるため，$\exists x.A(x)$ が証明されたからといって，この x を具体的に計算することができる保証はまったくない．実際，「すべての実数を順番に並べる方法がある」という定理が成立するが，具体的な並べ方を紙に書くことは (どんな数学者といえども) できない．また，「どんな円でも半径と円周の長さの比率が一定である」という定理から円周率を計算することはできないだろう．しかし，近年の数学の傾向として，$\exists x.A(x)$ の証明は，構成的であることが望ましい，とされるようになってきた．構成的な存在証明とは，その証明の中に「x を具体的に計算する手続き」がはいっているものである．構成的証明は，直観主義論理で証明を作ることに相当し，一般の (非構成的かもしれない) 証明は，古典論理で証明を作ることに相当する．直観主義論理は，古典論理における排中律 (あるいはそれと同等な他の原理) を使うことができないので，証明を作る段階では苦労が多いが，証明できてしまえば，「計算手続きを含む」という非常に良い性質を持つものである．

以上のように，直観主義論理，あるいは，構成主義は，純粋な論理や数学の世界でも注目されている重要な論理であるといえる．

9.3.4 では，計算と論理は本当に同じか？

この問に対する答えは 1 つではない．論理体系や計算体系の定式化 (形式的体系を作ること) は，一意的ではなく，多様な形式的体系があり得るからである．

論理体系の形式化は，3 つのスタイルがある．

- 自然演繹 (natural deduction), 本講義で採用したスタイル

- シーケント計算 (sequent calculus)

$$\frac{}{\Gamma, A \vdash \Delta, A}\ intial \qquad \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \supset B}\ \supset R \qquad \frac{\Gamma_1, B \vdash \Delta_1 \quad \Gamma \vdash A, \Delta_2}{\Gamma_1, \Gamma_2, A \supset B \vdash \Delta_1, \Delta_2}\ \supset L$$

  自然演繹における I(導入規則) と E(除去規則) のかわりに，このように，L($\vdash$ の左側に導入) 規則と R($\vdash$ の右側に導入) 規則とがある．

- Hilbert 流 (Hilbert が考案したスタイル)

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}\ modus\ ponens$$

$$\frac{}{\Gamma \vdash (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))}\ S \qquad \frac{}{\Gamma \vdash A \supset (B \supset A)}\ K$$

  推論規則は modus ponens だけで，あとは全て S, K のような公理である．

古典論理や直観主義論理など多くの論理体系は，これら 3 つのスタイルのいずれでも形式化でき，どのスタイルでの形式化も同等 (証明できる論理式は同じ) であることが知られている．

Curry-Howard の同型対応は，このうち，自然演繹スタイルと Hilbert スタイルに対して成立するものである．シーケント体系スタイルに対しては Curry-Howard の同型対応に相当するものはない．したがって，シーケント計算スタイルでしか与えられていない論理体系は，対応する計算体系がない，ということになる．

一方，プログラミング言語は非常に複雑なものが多い (ほとんどの実用的なプログラミング言語は，その厳密な定義を書くと 1 冊の本になる．) 本講義で述べた型付きラムダ計算であれば，それに対応する論理体系を考えることは容易であったが，たとえば，C 言語に対応する論理体系を厳密に定義するのはほとんど不可能である．

このような観点から，計算と論理が「どんな場合にも完全に一致する」ということは言えないことがわかる．しかしながら，以下のようなことは言える．

- 論理体系のみ，あるいは，計算体系のみを考察対象としているときに比べて，Curry-Howard の同型対応が成立するような論理体系と計算体系がある場合，(後に述べる 4 つの例からもわかるように) 体系に関する種々の良い性質を証明する手だてが用意され，見通しの良い議論を展開することができる．
- したがって，論理体系のみ，あるいは，計算体系のみが与えられた場合，「Curry-Howard の同型対応が成立するような相棒はなにか？」を考えることは有意義である．そのような相棒が存在しない場合，多くの場合は，もとの体系が適切に構成されていないことを意味する．
- すなわち，C-H の同型対応が成立するように論理/計算体系を設計すべき，という設計の指針となる．

9.4 対応関係の拡張

この章では，Curry-Howard の同型対応を拡張することにより，プログラミング言語および論理体系に対する新たな知見が得られる例を 4 つ紹介する．本節で述べられるのは，どれもごく表面的な introduction のみであるので詳しくは大学院での授業「プログラム理論特論」や書籍等を参照されたい．

9.4.1 その 1: 限量子と依存型

この講義における論理は，命題論理であった．しかし，我々の日常の推論は (少なくとも) 一階述語論理程度の表現力を必要とする．論理体系を命題論理から一階述語論理に拡張したとき，対応する計算体系が型付きラムダ計算からどのようなものに拡張されるかを考えてみよう．

一階述語論理の論理記号は，命題論理のそれに比べて，$\forall$ や $\exists$ という 2 つの論理記号が増えている．

そこで，$\forall x.A(x)$ という命題を型と見なそう．この命題の証拠は，構成的解釈によると，「x をもらうと $A(x)$ の証拠を返す関数」である．これを型の世界の言葉で書くと「x をもらうと $A(x)$ という型の要素を関数」となる．このように，項 x に依存して決まる型 $A(x)$ を依存型 (dependent type) という．

例: `newlist` を，自然数 n をもらうと，長さ n のリスト (要素はすべて 0) を返す関数とする．

$$\texttt{newlist}(n) = \begin{cases} \text{空リスト} & (n = 0 \text{ の時}) \\ \texttt{cons}(0, \texttt{newlist}(n-1)) & (n > 0 \text{ の時}) \end{cases}$$

ここで $\texttt{cons}(a, L)$ は，リスト L の先頭に要素 a を追加したリストを表す．たとえば，$\texttt{newlist}(3) = \langle 0, 0, 0\rangle$ となることは想像できよう (ここで，リストを $\langle a, b, c\rangle$ のように表現した．)

このような関数はしばしば現れる有用な関数である．では，その型は何であろうか？「リスト」を表す型を `List` とすると，$\texttt{newlist} : \texttt{Nat} \to \texttt{List}$ となる．これは正しい型付けではあるが，応用局面によっては，より詳しい情報が欲しいことがある．つまり，「長さが不明なリスト」ではなく「長さ n のリスト」が返されることを表現したい場合である．そのような型を $\texttt{List}(n)$ とすると，$\texttt{newlist}: \texttt{Nat} \to \texttt{List}(n)$ という型を持たせたいが，これでは正しい型とはならない．(変数 n が自由に現れてしまっている．実際にはこの n は引数のことである．) そこで，このような型を，

$$\texttt{newlist} : \Pi(n : N)\texttt{List}(n)$$

と表現して，`newlist` が「自然数 n をもらうと，$\texttt{List}(n)$ 型の要素を返す関数」の型を持つことを意味する．この Pi が依存型を構成する型構成子の 1 つである．Π は Curry-Howard の同型対応のもとで $\forall$ に対応する．

一方，$\exists x.A(x)$ という命題の証拠は，「x と，$A(x)$ の証拠の対」である．これは Σ という型構成子で構成される型に相当する．たとえば，$< 3, (0\ 0\ 0) >$ が $\Sigma(n : N)List(n)$ の要素である．

以上まとめると，以下の対応関係になる．

直観主義の一階述語論理	依存型を持つ型付きラムダ計算
$\forall$	Π 型
$\exists$	Σ 型

この対応は，命題論理と (依存型を持たない) 型付きラムダ計算との対応 (Curry の対応) の拡張となっている．この $\forall$ や $\exists$ への拡張部分を Howard が示したので，全体を合わせて Curry-Howard の対応と言う．

依存型は，随分奇妙な型と思うかもしれないが，論理の世界では昔から知られた $\forall$ や $\exists$ のことだと思うとわかりやすい．プログラム言語の方での依存型の重要性は近年認識されてきており，後の応用局面で重要な意味を持つ．

9.4.2 その 2: 帰納と帰納型

前節の拡張により，一階述語論理 (の直観主義論理版) まで拡張されたが，これではまだ数学的な表現のためには力不足である．最も基礎的な数学は自然数に関する理論であり，そこでの証明は数学的帰納法が鍵となる．

$$\frac{\Gamma \vdash A(0) \quad \Gamma, A(x) \vdash A(x+1)}{\Gamma \vdash \forall x.(Nat(x) \supset A(x))} \text{ (数学的帰納法)}$$

このルールは，$A(x)$ を x に関する何らかの命題とするとき，「$A(0)$」と「$A(x)$ ならば $A(x+1)$」の 2 つを導けたならば，「すべての自然数 x に対して $A(x)$」を導いていよい，というものである．

数学的帰納法は，一般に「帰納法」と呼ばれる推論法則の一例であり，帰納法は何らかの集合 (あるいは述語) の帰納的定義に付随して発生するものである．自然数に対する帰納法は数学的帰納法であるが，そのほかにも，リストに対する帰納法や，二分木に対応する帰納法が考えられる．

このような論理の世界における帰納法に対応する計算の世界の登場人物を考えよう．まず，自然数など，集合 (あるいは述語) の帰納的な定義に対応するものを考えると帰納型 (inductive data) という考えに到達する．帰納型は，型を帰納的に定義したものであり，たとえば，

$$\frac{}{\Gamma \vdash 0 : Nat} \qquad \frac{\Gamma \vdash x : Nat}{\Gamma \vdash s(x) : Nat}$$

という 2 つの規則で生成されるのが，自然数をあらわす Nat 型であり，

$$\frac{}{\Gamma \vdash <> : NatList} \qquad \frac{\Gamma \vdash n : Nat \quad \Gamma \vdash L : NatList}{\Gamma \vdash < n, L > : Nat}$$

という 2 つの規則で生成されるのが，自然数のリストをあらわす NatList 型である．このように，帰納的に型 (データ型) を定義することにより，型の表現力が非常に強くなることに注意されたい．

さて，帰納型が定義できたので，次に帰納法に対応するルールを考えよう．自然数の場合，

$$\frac{\Gamma \vdash n : Nat \quad \Gamma \vdash f : Nat \to T \to T \quad \Gamma \vdash a : T}{\Gamma \vdash Rec(n, f, a) : T}$$

という形になる．やや複雑な形に見えるかもしれないが，これは帰納法のルールを非常に自然に表現したものである．数学的帰納法にあらわれる $A(x)$ という命題がここでは T という型で表現され，(A が任意の述語でよかったように T も任意の型でよい)，$a : T$ は $A(0)$ の証拠に相当し，$f : Nat \to T \to T$ は $\forall n : Nat(A(n) \supset A(n+1))$ の証拠に相当する．すなわち，f は，自然数と $A(n)$ の証拠が与えられると $A(n+1)$ の証拠を返す関数である．

上記の Rec という関数はこのルールのために新たに導入された記号であり，以下の計算規則をもつ．(この計算規則は勝手に決めたものではなく，帰納型の定義から自動的に生成されるものであるが，ここでは詳細は省略する．)

$$Rec(0, f, a) \to a$$

$$Rec(s(n), f, a) \to f(n, Rec(n, f, a))$$

すなわち $R(n, f, a)$ の計算は，n により場合分けされ，n が 0 のときは a そのものを返し (a は $A(0)$ の証拠であったことに注意せよ)，n が 0 より大きいときは，f を n 回繰返し適用して $A(n)$ の証拠を得るものである．

上と同様に，リスト型に対する Rec 関数や二分木型に対する Rec 関数を定義することができる．このように，論理における帰納は，計算における帰納型 (および Rec 関数による計算) に対応するといえる．

補足: より強力な繰返し機構について

本節で紹介したのは，繰返し計算機構の中でも最も単純な原始帰納法に対応するものである．原始帰納法は，上記の Rec 関数を見てわかるように $n = 0, 1, 2, \cdots$ に対する値を順番に計算していくので，単純な for ループに対応していることがわかる．一般の再帰呼び出しはたとえば，

$$f(x) \ = \ if \ x = 1 \ then \ 1 \ else \ if \ even(x) \ then \ f(x/2) \ else \ f(3x + 1)$$

のように，x における f の値が x より前の値とは限らない y に対する値 $f(y)$ に依存している．このような再帰を一般再帰と呼ぶ．原始帰納による計算が必ず停止するのに対して，一般再帰は，計算が必ずしも停止しない．論理に対応する計算体系は，計算が停止するもののみを考えるため，一般再帰に対応する論理は存在しない．

9.4.3 その 3: 2 階論理と多相型

多相型 (polymorphic type) とは，型の世界を拡張したものである．まず例を考える．

型なしラムダ計算の例で出てきた，$double = \lambda f.\lambda x.f(f(x))$ というラムダ式は，「関数 f をもらって，その効果を 2 倍にする関数を返すような高階の関数」を表していた．これを型付きラムダ計算の世界で考えると，

$$\vdash \lambda f : A \to A.\lambda x : A.f(f(x)) \quad : \quad (A \to A) \to (A \to A)$$

という型付けを持つことがわかる．ところで，ここで A は何であろうか？明らかにどのような型を A のところに代入しても，上記のラムダ式はその型を持つ．つまり，上のラムダ式は実質的に「任意の型 A に対して $(A \to A) \to (A \to A)$ という型をもつ」はずである．

ところが，この講義で扱った範囲では，このような「任意の型」という表現はなかったので，このことは表現できない．したがって，この講義で扱った体系をプログラム言語と考えると，「上記の A を自然数の型にした場合」や「上記の A を文字列の型にした場合」などを別々に定義しなければならず，実質的に同じプログラムであるはずなのに，何回もプログラムを書かなければならない，という不具合が生じる．

そこで，「任意の型」を表現できるように，型の世界を拡張することを考える．ちなみに，ここで言う「任意の型」は，変化するものが型であるので，前々節で与えたような Π 型では表現できない．(Π 型は，変化するものが型ではなく，自然数など「項で表現されるもの」であった)．しかし，「任意」であることにはかわらないので，$\forall X$ と書いてしまおう．すなわち，$\forall X.((X \to X) \to (X \to X))$ というような型を許すのである．この型の意味は「どんな型 X に対しても $((X \to X) \to (X \to X))$ という型を持つ項の型」というものである．これを使うと上記のラムダ式は，

$$\vdash \lambda f : X \to X.\lambda x : X.f(f(x)) \quad : \quad \forall X.((X \to X) \to (X \to X))$$

と型付けできることになる．これによって，この 1 つのラムダ式を使って，「X を自然数の型にした場合」や「X を文字列の型にした場合」などが使える．

このような型を多相型 (polymorphic type) と呼び，現代的なプログラム言語では非常に重要な概念となっている．(ML 言語では，制限された多相型が導入されている．)

多相型も Curry-Howard の同型対応によって論理の世界に対応付けることができる．「型」は「命題」に対応付けられたので，「全ての型」は「全ての命題」に対応付けられる．すなわち，多相型に対応する論理体系は，「全ての命題について...」という表現を持つ強力な体系である．(一階述語論理では「全ての自然数につ

いて...」という表現はできたが，「全ての自然数上の命題について...」という表現はできなかったことに気付いてほしい．)

このように，「命題」や「述語」を表す変数を持ち，その変数を限量子によって束縛することのできる論理を二階論理という．(二階論理における命題をさらに束縛すれば 3 階論理になる等の階層があるが，ここでは立ちいらない．) 二階論理は，一階論理に比べて，本当に表現力が強くなる (圧倒的に強くなる) ことが知られている．二階論理はそれ自体非常に興味深い．なぜなら，人間は知らず知らずのうちに一階論理の範囲を越えた推論を平気でやっているからである．また，プログラム言語の世界での多相型は，型システムの重要性の高まりとともに非常に重要な概念となりつつある．

9.4.4 その 4: 古典論理とコントロールオペレータ

本講義で扱った内容は，すべて直観主義論理であった．すなわち，計算 (コンピュテーション) の論理ではあったが，排中律 ($A \vee \neg A$) や 二重否定除去の規則 ($(\neg\neg A) \supset A$) が成立しない，という意味で人間の通常の推論からは「異常」な世界であった．

では，これらの規則に (Curry-Howard 対応の意味で) 対応するような計算の論理はないのであろうか？この疑問は昔は真面目に考えられることはなかったが，1980 年頃の Griffin という研究者の研究をきっかけに急速に研究が深まり，今日では極めて豊富な結果が得られている．

その中身は，ここで述べるには高度になり過ぎるので，雰囲気だけを箇条書きする．

- 排中律や二重否定除去の規則に対応するのは，普通のラムダ計算の世界の中のものではなく，コントロールオペレータと呼ばれるものである．
- コントロールオペレータとは，ラムダ計算などの関数型プログラム言語における制御演算子である．(制御演算子とは，C 言語や FORTRAN 言語においては，GOTO や IF-THEN-ELSE や WHILE など，実行の順序を変更する構文のことである．)
- コントロールオペレータには，例外 (JAVA 言語, ML 言語など), キャッチスロー (Common Lisp 言語など), 継続 (Scheme, Standard ML 言語など) などがある．

実行の順序を変更する機構があると，なぜ，古典論理に対応するのか？簡単に説明するのは難しいが，直感的に言うと: JAVA の例外 (exception) 機構を例にとると，例外機構を使ったプログラムの実行では，「通常終了 (例外が起きずに計算が終わる)」と「例外発生」の 2 通りの終わり方がある．これを利用すると，「A が成立するときは B を行い，$\neg A$ が成立するときは C を行う」といった場合分けの論法に対応する「証拠」を計算することができる．$A \vee \neg A$ という排中律は場合分けの論法に対応しているので，例外機構を使って，排中律の証拠を書くことができれば，古典論理に対応した論理になるであろう．