8.4 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．

8.4.1 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．

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

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．

8.4.2 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．

8.4.3 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．

8.4.4 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.