7　型付きラムダ計算

プログラム言語はコンピュータ・ソフトウェアを記述する言葉であり，良いソフトウェアの構築には，良いプログラム言語の利用が欠かせない．現代のプログラム言語に欠かせない要素が、洗練された型のシステムである。本講義では、型システムを中心において、プログラム言語の理論的基礎を学ぶ。

7.1　型の概念

型付きラムダ計算は，型のないラムダ計算に型 (Type) の概念を導入した計算体系である．では，型とは何だろうか？

C, Java, Fortran, ML など、多くのプログラム言語は，型システムを持っている．型には，整数型や浮動小数点型などの基本的なものから，配列型，レコード型，構造体の型、関数の型など複合的なものまである．プログラム中の変数や式は、ただ 1 つの型をもつことになっているため，異なる型に対して使われたとき，型の整合性が崩れたとしてエラーとなる．

型は、より正確にいうと，同じ操作が可能な一群のデータを特徴付ける情報であり，個々のデータの値によって変わらない．従って，実行時に変数の型が変化することは通常はないため，実行前 (コンパイル時) に，型の整合性を判定することができる．すなわち，動的に思われる「計算」の世界における**静的な概念の代表選手**が，型である．

現代的プログラム言語の多くが型システムを持っているのは，型の概念が有用であることを示している．実際，プログラミング上の些細な (しかし，頻繁に起きる) 誤りの多くは型エラーの形で発見することができる．また，場合によっては，型情報を生かして実行速度の向上を図ることができる．

型の整合性の判定は，簡単に思えるかもしえない。たとえば、「int 型の変数に char 型の値を代入しようとした」というエラー判定は容易である。しかし、このような基本的な型だけでなく、型構成子がある場合，必ずしも簡単ではない。

例 9

$$S = \lambda x.\lambda y.\lambda z.xz(yz)$$

$$K = \lambda x.\lambda y.x$$

とするとき， S や K はどういう型をもつ関数か？

この講義では，型のないラムダ計算に型システムを導入した型付きラムダ計算の体系をとりあげる．ここで扱うのは単純型付きラムダ計算と呼ばれる最もシンプルな体系をやや拡張したものである．

7.2　構文

プログラムの構文を定義する前に，型の構文を定義する．

ここでは，`Int`, `String` など基本となる型の内部構造には立ち入らないので，それらを単に $K_1, K_2, \cdots, K_n$ という型定数で表示する．

$$\frac{}{K_i : Type} \qquad \frac{A : Type \quad B : Type}{A \times B : Type} \qquad \frac{A : Type \quad B : Type}{A + B : Type} \qquad \frac{A : Type \quad B : Type}{A \to B : Type}$$

$\times$ は直積, $+$ は直和, $\to$ は関数空間を表す型である．それらの意味は，後に述べる計算規則のところで明らかになる．

次に，項の構文を定義する．この際以下の 2 つの点に注意する．

- 型付きラムダ計算においてはすべての項は型を (整合的に) 持たなければいけないので，単に「M が項である」ということを推論する規則ではなく,「M が型 A の項である」ということを推論する規則とする．
- さらに，項の構成の過程で，M に含まれる変数がどういう型を持つかを覚えておく必要がある．そのことを，$x_1 : A_1, \cdots, x_n : A_n$ という形の列として表現することにして，一般にこういう列を Γ という文字であらわす．この列のことを「宣言」と呼ぶ．

以上の 2 点から，項の構文の構成規則は $\Gamma \vdash M : A$ という形を推論するための規則となることがわかる．ここで $\vdash$ や : という記号には深い意味はなく，(Γ, M, A) という 3 つ組の推論規則でもよかったのだが，伝統的な記法に従った．

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ \texttt{var}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B}\ \texttt{pair} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \texttt{left}(M) : A}\ \texttt{left} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \texttt{right}(M) : B}\ \texttt{right}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \texttt{inl}(M) : A + B}\ \texttt{inl} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \texttt{inr}(M) : A + B}\ \texttt{inr}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x : A \vdash N : C \quad \Gamma, y : B \vdash L : C}{\Gamma \vdash \texttt{case}(M, x : A.N, y : B.L) : C}\ \texttt{case}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}\ \lambda \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \texttt{apply}$$

それぞれの規則の横棒の右隣に規則名を書いた．(var, pair など)

型なしラムダ計算における規則に対応するのは，var, λ, apply の 3 つであり，それ以外は，ここで新しくでてきたものである．$\lambda x : A.M$ は，型なしラムダ計算における $\lambda x.M$ という項に x の型 A の情報を付加したものである．講義では，見やすさのため，しばしば $\lambda(x : A)M$ と書く．MN という形の式は型なしラムダ計算と同様に，関数 M に引数 N を適用させた結果の値を表す．

pair は直積型の項を作る規則であり，left, right は直積型の項を使う規則である．inl, inr は直和型の項を作る規則であり，case は直和型の項を使う規則である．これらの意味は必ずしも直感的に想像できないかもしれないが，後で計算規則がでてきたときに明らかになる．ここでは，項は上の規則によって，機械的に (コンピュータでもわかるように) 構成される，ということを理解してほしい．

また，宣言 Γ が必ずしも一定でないことにも注意してほしい．たとえば，$\lambda x : A.M$ という項は，x という変数が束縛されるので，M に対する宣言である $\Gamma, x : A$ より $x : A$ が減り，Γ だけになっている．(M の中で x が使われていなくてもよい．) 同様に，$\texttt{case}(M, x : A.N, y : B.L)$ という項では，N の中に x, L の中に y が自由に現れていてもよいのだが，それらは，$\texttt{case}(M, x : A.N, y : B.L)$ という項の中では束縛されるということを表している．

上記の規則を有限回適用して $\Gamma \vdash M : A$ が推論できたときに「宣言 Γ のもとで M は型 A を持つ項である」という．(Γ が空の列のとき,「M は型 A を持つ項である」ということもある．)

例 10 型つきラムダ計算の項の構成の例をあげる．

$$\frac{\dfrac{}{x : A \vdash x : A}\ \textit{var}}{\vdash \lambda x : A.x : A \to A}\ \lambda$$

$$\dfrac{\dfrac{\dfrac{}{x:A\times B\vdash x:A\times B}\ var}{x:A\times B\vdash \mathit{right}(x):B}\ right \quad \dfrac{\dfrac{}{x:A\times B\vdash x:A\times B}\ var}{x:A\times B\vdash \mathit{left}(x):A}\ left}{\dfrac{x:A\times B\vdash \langle \mathit{right}(x),\mathit{left}(x)\rangle : B\times A}{\vdash \lambda x:A\times B.\langle \mathit{right}(x),\mathit{left}(x)\rangle : (A\times B)\to(B\times A)}\ \lambda}\ pair$$

$$\dfrac{\dfrac{}{x:A+B\vdash x:A+B}\ var \quad \dfrac{\dfrac{}{x:A+B,y:A\vdash y:A}\ var}{x:A+B,y:A\vdash \mathit{inr}(y):B+A}\ inr \quad \dfrac{\dfrac{}{x:A+B,z:B\vdash z:B}\ var}{x:A+B,z:B\vdash \mathit{inl}(z):B+A}\ inl}{\dfrac{x:A+B\vdash \mathit{case}(x,y:A.\mathit{inr}(y),z:B.\mathit{inl}(z)):B+A}{\vdash \lambda x:A+B.\mathit{case}(x,y:A.\mathit{inr}(y),z:B.\mathit{inl}(z)):(A+B)\to(B+A)}\ \lambda}\ case$$

7.3 型検査と型推論

判定問題とは，YES か NO かを判定する問題である．判定問題をコンピュータによって有限時間内に解くことができるとき，決定可能 (decidable) という．

型のある計算体系に対して，以下の 3 つの判定問題を解くことが重要である．

- Γ, M, A が与えられたとき，$\Gamma \vdash M : A$ となるか？(型検査問題)
- M が与えられたとき，$\Gamma \vdash M : A$ となる Γ, A があるか？(型推論問題)
- A が与えられたとき，$\Gamma \vdash M : A$ となる Γ, M があるか？

これらの問題が決定可能であるかどうかは，型システムに依存して決まるが，一般に，型推論の方が型検査より難しいことが多い．3 番目の問題は，体系によって難しさが大きく異なる．型システムとして有用なのは，型検査問題と型推論問題の 2 つが決定可能である場合である．

型システムを持つほとんど全てのプログラム言語において，型検査/型推論問題は決定可能であり，言語処理系であるコンパイラに組み込まれている．(C 言語等では，Γ, A が完全に与えられて整合性の判定をしているので，型検査問題である．一方，ML 言語等では，型の情報はプログラム中に与えられておらず，コンパイラが型推論しなければならない．また，JAVA バイトコードの verifier なども型情報がないところから型情報を復元しているので，型推論問題を解いていることになる．)

本講義の型つきラムダ計算の体系に対しては，以下の定理が容易に示せる．

定理 6 先に与えた型付きラムダ計算の体系に対して，型検査問題，型推論問題は決定可能である．

ここではこの定理の証明 (型検査および型推論アルゴリズム) を与えるかわりに，演習によって，実際にどのようなアルゴリズムで型推論できるかを体験してもらうことにする．

7.4 Church 流と Curry 流

本講義で扱っている型つきラムダ計算は，Church-style と呼ばれるものである．これは，ラムダ式の構成において，$\lambda x : A.M$ のように必ず型 A を記述する流儀である．このため，型検査や型推論が簡単になっている．C 言語などで，ブロックごとに局所変数の型を宣言するのは，Church-style と同じである．

```
int foo (int x)
{
   int y;
   y = x * 2 + 1;
   ...
}
```

一方，ラムダ式の構成において，$\lambda x.M$ のように型 A を記述しない流儀もあり，Curry-style と呼ばれる．こちらでは，型検査や型推論を行う際に，x の型を推論しなければいけないので，問題が難しくなる．これは，ML のように変数の型を宣言しない言語に対応している．

```
fun foo x = let y = x * 2 +1 in ...
```

7.5 Lambda ゲーム

Lambda ゲーム (型付きラムダ計算の項の導出) における解答の形式は以下の通りである。

```
A[問題番号][
仮定列 ⊢ 項:型 in Lambda since
   導出
]
```

仮定列 (あるいは宣言列) は、変数 (x や A)、変数の型の宣言 (X1::x:A の形) の列である。ただし、空列でもよい。仮定列のうち、変数の部分は、これから書く導出で使う変数と型をすべて列挙したものである。不足していると CAL システムから「宣言されていない変数である」という旨のエラーが出る。仮定列のうち、変数の型の宣言の部分は、「名前::変数:型」の形をしているものの列である。ここで名前というのは、あとでこの宣言を参照するためのものである。慣習として名前としては英大文字で始まる英数字列 (たとえば “X1”) を使うことにする。型のところには、どんな型でも書いてよいので、$X1 :: x : A \to B + C$ と書いてもよい。

CAL システムの Lambda ゲームにおける項は、見やすさのため、講義における形式とは若干違う形式で扱われる。

$\lambda x : A.M$ は $\lambda\ (x : A)M$ と表記される．
$\mathtt{case}(M, x : A.N, y : B.L)$ は $\mathtt{case}(M, (x : A)N, (y : B)L)$ と表記される．
MN は $M(N)$ と表記される．

特に、関数適用では必ず $M(N)$ というように括弧を書くことに注意してほしい。この制限をつけることにより、CAL システムの parser が楽になり、より的確なエラーメッセージを出すことができるようになる。

Lambda ゲーム (型付きラムダ計算の項の導出) で使ってよいルールは前節で述べた 9 つのルールだが、これらの、CAL システムにおける表現は以下の通りである。

```
x:A by var {
  x:A という仮定に付けた名前 (X1 など)
}
```

```
<M,N>:A × B by pair {
   M:A の導出 ;
```

```
   N:B の導出
}

left(M):A by left {
   M:A × B の導出
}

right(M):B by right {
   M:A × B の導出
}

inl(M):A ＋ B by inl {
   M:A の導出
}

inr(M):A ＋ B by inr {
   M:B の導出
}

case(M,(x:A)N,(y:B)L):C by case {
   M:A ＋ B の導出 ;
   (名前１ :: x:A)[ N:C の導出 ] ;
   (名前２ :: y:B)[ L:C の導出 ] ;
}

λ (x:A)M: A → B by λ {
   (名前 :: x:A)[ M:B の導出 ] ;
}

M(N): B by apply {
   M:A → B の導出 ;
   N:A の導出
}
```

以上のルールの組み合わせによって、項の型を導出するゲームが Lambda である.