多相型について
「計算論理学」資料 (亀山)

この資料では、Curry style の単純型付きラムダ計算に、「多相型」を追加したシステムについて述べる。

この資料では、CAL の書き方ではなく、ラムダ抽象を $\lambda x.M$ と書く。(Curry style なので、x の型は明示しない。)

1 多相型

$\lambda f.\ \lambda x.\ f(f(x))$ は、$(\mathtt{nat} \to \mathtt{nat}) \to (\mathtt{nat} \to \mathtt{nat})$ という型も、$(\mathtt{bool} \to \mathtt{bool}) \to (\mathtt{bool} \to \mathtt{bool})$ という型も持つ。

型推論アルゴリズムでは、この項の型は、$(\alpha \to \alpha) \to (\alpha \to \alpha)$ という型 (α は型変数) として推論される。

このように「どの型でもよい部分」を持っている型を、多相型 (polymorphic type) と言う[1]。多相型を (型推論の中だけでなく) プログラムの型として積極的に使えるようにしたものが多相的ラムダ計算である。多相的ラムダ計算では、上記の項は、$\forall\alpha.\ ((\alpha \to \alpha) \to (\alpha \to \alpha))$ という型として表現される。$\forall$ は論理記号から来たものであり、「すべての型に対して」を表す。

これには下記のようにいくつかの種類があるが、Curry Style で型推論ができるのは、一番簡単な ML のものだけであり、その範囲でも非常に有用なものであることが知られている。

- let 多相 (制限された多相; ML の型システムに対応)
- 2 階の多相型 (System F に対応する型)
- 高階の多相型 (Haskell の型システムに対応; Coq などの定理証明系でも利用)

2 let 多相の型システム

講義で繰返し説明してきた型システムの設計の原則は、「新しい型を導入するには、その型の要素を導入する規則と、その型の要素を使う規則との 2 種類が必要である」というものであった。よって、多相型をシステムにいれるためには、多相型の要素を導入する規則と、多相型の要素を使う規則との 2 種類が必要である。ただし、ML の場合、これらは、独立した新しい規則としてではなく、それぞれ、let 規則と、変数規則と結合して 1 つの規則になっているので、それを紹介する。

まず、型の世界を拡張する。

$$A, B ::= K \mid A \times B \mid A + B \mid A \to B \mid \alpha$$
$$\sigma ::= A \mid \forall\alpha.\ \sigma$$

ML の let 多相の本質的な制限は、「$\forall$ の記号は、型の一番外側にしか来てはいけない」ということである。つまり、2 つの構文に分けて、$(\forall\alpha.\alpha \to \alpha) \times \beta$ というように、$\forall$ が内側に来るものを排除している。一方、$\forall$ は一番外にあれば、何個かさなってもよいので、$\forall\alpha.\forall\beta.(\alpha \to \beta)$ は、OK である。

A, B を型、σ を型スキームと呼ぶ。

上記の変更に対応して、型判断 $\Gamma \vdash M : A$ も拡張する。

- Γ は $x_1 : \sigma_1, \cdots, x_n : \sigma_n$ の形とする。つまり、Γ にあらわれる型 σ_i は、型スキームでよい。(普通の型も、型スキームの一種であることに注意。)
- 項の型 A の部分は、従来通りの型でなければならない。($\forall$ を含んではいけない。)

[1] 正確には、オブジェクト指向言語で用いられるサブタイピング多相や、Haskell などのプログラミング言語にあるアドホック多相と区別して、パラメトリック多相 (parametric polymorphism) と言う。

さて、規則の変更は 2 か所だけである。まず、変数を導入する規則は、多相型を「使う」規則と合体して以下のようになる。

$$\frac{\text{(下記の条件つき)}}{\Gamma \vdash x : A}$$

条件: $(x : \forall\alpha_1, \cdots, \forall\alpha_n.B) \in \Gamma$ であり、かつ、ある $C_1, \cdots, C_n$ に対して、$B\{\alpha_1 := C_1, \cdots, \alpha_n := C_n\}$ が A と一致する。

これはなかなか面倒なルールだが、まず、$n = 0$ のときは、従来の変数の規則と同じである。$n > 0$ のときは、Γ の中の型スキームを取りだし、その α_i のところに、適当な型 C_i を代入して型 A を得る。ここで、「適当な型」というのは、どんな型でもよいので、まさに多相型 $\forall$ を使っている、という感じになる。

次に、let 規則というものを導入しよう。そもそも、let とは何かというと、単相型のシステムでは、$\texttt{let}\ x = M_1\ \texttt{in}\ M_2$ は、$(\lambda x.M_2)\ M_1$ と同じである。つまり、call by value では、M_1 を計算し、その結果 V を変数 x に束縛し、M_2 を計算する、というプログラムである。ML では、その let に「多相型の導入」の機能を合体させていて、次の型規則となる。

$$\frac{\Gamma \vdash M_1 : A \quad \Gamma, (x : \sigma) \vdash M_2 : B}{\Gamma \vdash \texttt{let}\ x = M_1\ \texttt{in}\ M_2 : B}$$

条件: A に含まれる型変数たちのうち、Γ に含まれない[2]ものを $\alpha_1, \cdots, \alpha_n$ とすると、$\sigma = \forall\alpha_1, \cdots, \forall\alpha_n.\ A$ である。ここで α_i たちの順番はどうでもよい。(どの順番でも同じ意味の型スキームとなるので、順番は気にしない。)

型付けの例をあげる。

例 (単相的な let のみ使用):

$$\frac{\vdash (1+2) : \texttt{nat} \quad x : \texttt{nat} \vdash (x * 3) : \texttt{nat}}{\vdash \texttt{let}\ x = 1 + 2\ \texttt{in}\ x * 3 : \texttt{nat}}$$

例 (多相的な let を使用):

$$\frac{\vdash (\lambda x.x) : \alpha \to \alpha \qquad \dfrac{\overline{f : \forall\alpha.\ (\alpha \to \alpha) \vdash f : \texttt{nat} \to \texttt{nat}} \quad \overline{f : \forall\alpha.\ (\alpha \to \alpha) \vdash 3 : \texttt{nat}}}{f : \forall\alpha.\ (\alpha \to \alpha) \vdash f(3) : \texttt{nat}}}{\vdash \texttt{let}\ f = \lambda x.x\ \texttt{in}\ f(3) : \texttt{nat}}$$

例 (多相的な let を本当に多相的に使用):

$$\frac{\vdash (\lambda x.x) : \alpha \to \alpha \qquad \dfrac{\overline{f : \forall\alpha.\ (\alpha \to \alpha) \vdash f : (\beta \to \beta) \to (\beta \to \beta)} \quad \overline{f : \forall\alpha.\ (\alpha \to \alpha) \vdash f : \beta \to \beta}}{f : \forall\alpha.\ (\alpha \to \alpha) \vdash f(f) : \beta \to \beta}}{\vdash \texttt{let}\ f = \lambda x.x\ \texttt{in}\ f(f) : \beta \to \beta}$$

単相型の世界では、$f(f)$ というパターンには型がつかなかったが、多相型では、2 つの f に違う型をつけることにより、全体として型がつくことになる。ただし、let 多相は λ で導入された変数には適用されないので、$\lambda f.\ f(f)$ は型がつかない。

例 (多相的な let を本当に多相的に使用):

$$\frac{\vdash M : (\alpha \times \beta) \to (\beta \times \alpha) \qquad \dfrac{\dfrac{\overline{\Gamma \vdash f : \texttt{BN} \to \texttt{NB}} \quad \dfrac{\overline{\Gamma \vdash f : \texttt{NB} \to \texttt{BN}} \quad \overline{\Gamma \vdash \langle 37, true\rangle : \texttt{NB}}}{\Gamma \vdash f(\langle 37, true\rangle) : \texttt{BN}}}{\Gamma \vdash f(f(\langle 37, true\rangle)) : \texttt{NB}}}{\vdash \texttt{let}\ f = M\ \texttt{in}\ f(f(\langle 37, true\rangle)) : \texttt{NB}}$$

ただし、$M = \lambda x.\ \langle \texttt{right}(x), \texttt{left}(x)\rangle$、$\texttt{BN} = \texttt{bool} \times \texttt{nat}$, $\texttt{NB} = \texttt{nat} \times \texttt{bool}$, $\Gamma = \{f : \forall\alpha.\forall\beta.\ (\alpha \times \beta) \to (\beta \times \alpha)\}$ とおいた.

[2]正確には、「Γ において、自由な型変数として含まれないもの」とすべきである。これは、たとえば、$y : \forall\alpha\ \alpha$ という型宣言では、α は自由には含まれないとしたいからである。