How to translate combinators into CPL

酒井政裕 ⟨s01397ms@sfc.keio.ac.jp⟩

2003 年 1 月 25 日

概 要

型付き λ 計算におけるコンビネータは、Cartesian Closed Cateogry (CCC) の射としてすべて表現可能である。本稿ではこの事実を CPL を使って簡単に説明する。

1 問題の単純化

型 $A \to B$ を持つコンビネータを CCC で表現したい。

ところで、同型対応 $\mathrm{hom}(1, \exp(A, B)) \xrightarrow{\phi} \mathrm{hom}(A, B)$ を、$f \mapsto \mathrm{eval} \circ \mathrm{pair}(f \circ !, I)$ で表現できるので、型 $A \to B$ を持つ射を直接定義する代わりに型 $1 \to \exp(A, B)$ を持つ射を定義できれば十分である。以下では $1 \to \exp(A, B)$ の形に翻訳する事だけを考える。

2 翻訳の補助関数

2.1 FV

FV を λ 式が含む自由変数の集合を求める関数とする。すなわち、

$$\begin{cases} \mathrm{FV}(x) & = & \{x\} \\ \mathrm{FV}(\alpha\beta) & = & \mathrm{FV}(\alpha) \cup \mathrm{FV}(\beta) \\ \mathrm{FV}(\lambda x.\alpha) & = & \mathrm{FV}(\alpha) \setminus x \end{cases}$$

2.2 Type

Type を以下のように定義される、λ 式の型を求める関数とする。

$$\begin{cases} \mathrm{Type}(x) & = & \ldots \\ \mathrm{Type}(\alpha\beta) & = & B \quad (\text{where } \mathrm{Type}(\alpha) = \exp(A, B) \text{ and } \mathrm{Type}(\beta) = A) \\ \mathrm{Type}(\lambda x.\alpha) & = & \exp(\mathrm{Type}(x), \mathrm{Type}(\alpha)) \end{cases}$$

2.3 Types

Types を以下のように定義される関数とする。

$$\mathrm{Types}(S) = \prod_{v \in S} \mathrm{Type}(v)$$

3　翻訳関数

λ式 α を型 $\mathrm{Types}(\mathrm{FV}(\alpha)) \to \mathrm{Type}(\alpha)$ を持つ射に翻訳する関数 $[]$ を以下のように定義する。

ところで、α がコンビネータであり自由変数を含まないのならば、$\mathrm{Types}(\mathrm{FV}(\alpha)) = \mathrm{Types}(\emptyset) = 1$ より、$[\alpha]$ は望みの射 $[\alpha] : 1 \to \mathrm{Type}(\alpha)$ となる。

3.1　変数の場合

$$[x] = I$$

3.2　$\alpha\beta$ の場合

$$[\alpha\beta] = \mathrm{eval} \circ \mathrm{pair}([\alpha] \circ \Pi_\alpha, [\beta] \circ \Pi_\beta)$$

ただし、

$$\begin{cases} \Pi_\alpha : \mathrm{Types}(\mathrm{FV}(\alpha\beta)) \to \mathrm{Types}(\mathrm{FV}(\alpha)) \\ \Pi_\beta : \mathrm{Types}(\mathrm{FV}(\alpha\beta)) \to \mathrm{Types}(\mathrm{FV}(\beta)) \end{cases}$$

は projection の集まりから自然に定まる射とする。

3.3　$\lambda x.\alpha$ の場合

$$[\lambda x.\alpha] = \mathrm{curry}([\alpha] \circ \Pi)$$

ただし、

$$\Pi : \mathrm{Types}(\mathrm{FV}(\lambda x.\alpha)) \times \mathrm{Type}(x) \to \mathrm{Types}(\mathrm{FV}(\alpha))$$

は projection の集まりから自然に定まる射とする。

4　翻訳例

4.1　I コンビネータ

$$\begin{aligned} [I] &= [\lambda x.x] \\ &= \mathrm{curry}([x] \circ \pi_2) \\ &= \mathrm{curry}(I \circ \pi_2) \\ &= \mathrm{curry}(\pi_2) \end{aligned}$$

4.2 K コンビネータ

$$\begin{aligned}
[K] &= [\lambda x.\lambda y.x] \\
&= \mathrm{curry}([\lambda y.x] \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}([x] \circ \pi_1) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(I \circ \pi_1) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(\pi_1) \circ \pi_2)
\end{aligned}$$

4.3 S コンビネータ

$$\begin{aligned}
[S] &= [\lambda x.\lambda y.\lambda z.xz(yz)] \\
&= \mathrm{curry}([\lambda y.\lambda z.xz(yz)] \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}([\lambda z.xz(yz)] \circ I) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(\mathrm{curry}([xz(yz)] \circ I) \circ I) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(\mathrm{curry}([xz(yz)])) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(\mathrm{curry}(\mathrm{eval} \circ \mathrm{pair}([xz] \circ \mathrm{pair}(\pi_1 \circ \pi_1, \pi_2), [yz] \circ \mathrm{pair}(\pi_2 \circ \pi_1, \pi_2)))) \circ \pi_2) \\
&= \mathrm{curry}(\mathrm{curry}(\mathrm{curry}(\mathrm{eval} \circ \mathrm{pair}([xz] \circ \mathrm{prod}(\pi_1, I), [yz] \circ \mathrm{prod}(\pi_2, I)))) \circ \pi_2)
\end{aligned}$$

ここで、

$$\begin{aligned}
[xz] &= \mathrm{eval} \circ \mathrm{pair}([x] \circ \pi_1, [z] \circ \pi_2) \\
&= \mathrm{eval} \circ \mathrm{pair}(I \circ \pi_1, I \circ \pi_2) \\
&= \mathrm{eval} \circ \mathrm{pair}(\pi_1, \pi_2) \\
&= \mathrm{eval}
\end{aligned}$$

同様に $[yz] = \mathrm{eval}$

したがって、

$$[S] = \mathrm{curry}(\mathrm{curry}(\mathrm{curry}(\mathrm{eval} \circ \mathrm{pair}(\mathrm{eval} \circ \mathrm{prod}(\pi_1, I), \mathrm{eval} \circ \mathrm{prod}(\pi_2, I)))) \circ \pi_2)$$

5 CPL での実行例

前節の翻訳を実際に定義してみよう。[1]

```
cpl> let I' = curry(pi2)
I' : *a -> exp(*b,*b)  defined
cpl> let K' = curry(curry(pi1).pi2)
K' : *c -> exp(*a,exp(*b,*a))  defined
cpl> let S' = curry(curry(curry(eval.pair(eval.prod(pi1,I),\\
 eval.prod(pi2,I)))).pi2)
S' : *d -> exp(exp(*c,exp(*a,*b)),exp(exp(*c,*a),exp(*c,*b)))  defined
cpl>
```

[1] domain の型がここまでの定義の 1 と異なり変数になってしまっているが、これは式の最も generic な型を推論しているためである。これまでの定義に!を結合していると考えても良い。

5.1 $SKK \Rightarrow I$

λ計算では $SKK \Rightarrow I$ が成り立つことが知られている。

CPL では curry の引数を直接簡約出来ないので、$SKK \Rightarrow I$ 自体は成り立たないが、簡単な例を使って SKK と I が等しい事が示せる。

```
cpl> simp eval.pair(I', 0)
0
    : 1 -> nat
cpl> simp eval.pair(eval.pair(eval.pair(S', K'), K'), 0)
0
    : 1 -> nat
```

この例では 0 の時に両者の値が等しいことがわかるが、I', K', S' は pr, s, 0 のいずれにも依存していない (= 引数の構造に依存していない) ので、実際にはどのような $f : X \to Y$ についても等しい。