オブジェクト指向言語モデルFJのラムダ計算への埋め込み

今井 宣洋 *

平成 21 年 12 月 12 日

概 要

オブジェクト指向言語のシンプルなモデルとして Featherweight Java (FJ) [1] がある. 本文書ではこれをラムダ計算へ埋め込み、FJ がラムダ計算の部分表現であることを証明する. なおこの文書のソースコードは運が良ければ `http://bitbuckt.org/yoshihiro503/pp/`リポジトリ内の `documents/doc.tex` から最新のものが得られるかもしれない.

1 導入

[3] を読んで「オブジェクト指向言語より関数型言語の方が表現力が高い」と思ったので証明してみた。OO 言語のモデルはこのまえ id:keigoi さんと飲んだときに教えてくれた FJ(Featherweight Java) を使用した。そして、関数型言語のモデルはおなじみ λ 計算にレコードと不動点演算を入れた単純拡張 λ 計算 (λ+ と呼ぶ) を利用した。2 節では FJ をサーベイし、3 節では λ+ をサーベイする。そして 4 節で埋め込み関数を定義し、それがうまいこと行ってることを証明した。

2 形なし FJ

2.1 FJ の定義

定義 2.1.1 (term)**.** 項 T、クラス定義 L、メソッド定義 M、式 e を以下で定義する、ただし、C, D はクラス名、f はフィールド名、m はメソッド名、x, y, z は変数名をそれぞれ表す文字列とする。

$$
\begin{aligned}
T &::= L \ldots L \\
L &::= \texttt{class}\ C < D\ \ f; \cdots ; f; K; M; \cdots ; M\ \texttt{end} \\
K &::= \texttt{def}\ \texttt{init}(x, \ldots, x)\ \ \texttt{super}(y, \ldots, y);\ \texttt{this}.f = z; \cdots ; \texttt{this}.f = z\ \texttt{end} \\
M &::= \texttt{def}\ m(x, \ldots, x)\ \ e\ \texttt{end} \\
e &::= x \mid e.f \mid e.m(e, \ldots, e) \mid \texttt{new}\ C(e, , e)
\end{aligned}
$$

*(有) IT プランニング imai@itpl.co.jp, twitter id:yoshihiro503

定義 2.1.2 (式の substitution)**.** 式 e の変数 x への式 d の代入 $[x \mapsto d]e$ を次で定義する.

$$
\begin{aligned}
[x \mapsto d]x &= d \\
[x \mapsto d]y &= y \ \ (x \neq y) \\
[x \mapsto d](e.f) &= ([x \mapsto d]e).f \\
[x \mapsto d](e.m(a_1, \ldots, a_n)) &= ([x \mapsto d]e).m([x \mapsto d]a_1, \ldots [x \mapsto d]a_n) \\
[x \mapsto d](new\ C(e_1, \cdots, e_n)) &= new\ C([x \mapsto d]e_1, \ldots, [x \mapsto d]e_n)
\end{aligned}
$$

定義 2.1.3 (簡約)**.** 式の簡約を以下のルールで定義する。各 e, e_i, d_i を式とし、各 v, v_i, w_i は値とする。

$$\frac{e \to_{\beta_{FJ}} e'}{e.f \to_{\beta_{FJ}} e'.f}$$

$$\frac{e \to_{\beta_{FJ}} e'}{e.m(\overline{e_i}) \to_{\beta_{FJ}} e'.m(\overline{e_i})}$$

$$\frac{e_i \to_{\beta_{FJ}} e_i'}{v.m(\ldots, e_i, \ldots) \to_{\beta_{FJ}} v.m(\ldots, e_i', \ldots)}$$

$$\frac{e_i \to_{\beta_{FJ}} e_i'}{\texttt{new}\ C(\ldots, e_i, \ldots) \to_{\beta_{FJ}} \texttt{new}\ C(\ldots, e_i', \ldots)}$$

$$\frac{}{(\texttt{new}\ C(v_1, \ldots, v_n)).f_i \to_{\beta_{FJ}} v_i}$$

$$\frac{mbody(m, C) = \texttt{def}\ m(x_1, \ldots, x_n)\ \ e\ \texttt{end}}{\texttt{new}\ C(\overline{v_j}).m(w_1, \ldots, w_n) \to_{\beta_{FJ}} [x_1 \mapsto w_1, \ldots, x_n \mapsto w_n, \texttt{this} \mapsto \texttt{new}\ C(\overline{v_j})]e}$$

3 λ に簡単な拡張を加えた λ+

[2] の第 11 節 (Simple Extentions) を参照した.
11.8 節のレコード拡張と 11.11 の fix 演算子拡張を行ったものを λ+ と呼び、以降で使う。

4 FJ から λ+ への変換

4.1 変換の定義と諸性質

定義 4.1.1 ($[\![\bullet]\!]_{meth}$)**.** *(*TODO*)*

定義 4.1.2 ($[\![\bullet]\!]_{cons}$)**.** *(*TODO*)*

定義 4.1.3 (FJ 式の変換)**.**

$$
\begin{aligned}
[\![x]\!] &= x \\
[\![e.f]\!] &= [\![e]\!].f \\
[\![e.m(d_1, \ldots, d_n)]\!] &= [\![e]\!].m\ \ [\![d_1]\!]\ \ \ldots\ \ [\![d_n]\!] \\
[\![new\ C(e_1, \ldots, e_n)]\!] &= \left\{ f_i = [\![e_i]\!]; k = [\texttt{this} \mapsto \texttt{fix}\ (\lambda \texttt{this}.t)]\ [\![C\#init]\!]_{cons}; m_j = [\texttt{this} \mapsto \texttt{fix}\ (\lambda \texttt{this}.t)]\ [\![C\#m_j]\!]_{meth} \right\} \\
&\text{(ただし } t = \left\{ f_i = [\![e_i]\!]; k = [\![C\#init]\!]_{cons}; m_j = [\![C\#m_j]\!]_{meth} \right\})
\end{aligned}
$$

補題 4.1.1.

$$e \text{ が } value \iff [\![e]\!] \text{が } value$$

証明: 略.

補題 4.1.2.

$$[\![[x \mapsto d]e]\!] = [x \mapsto [\![d]\!]]\,[\![e]\!]$$

証明: 略.

定理 4.1.1.

$$e \to_{\beta_{FJ}} e' \text{ならば } [\![e]\!] \to^*_{\beta_{\lambda+}} [\![e']\!]$$

証明:
e の構造に関する帰納法で証明する.

- $e.f$ で $e \to_{\beta_{FJ}} e'$ のとき

$$\begin{aligned}[\![e.f]\!] &= [\![e]\!].f \\ &\to_{\beta_{\lambda+}} [\![e']\!].f \qquad \text{(帰納法の仮定より)} \\ &= [\![e'.f]\!] \qquad OK\end{aligned}$$

- $(\texttt{new}\ C(v_1, \ldots, v_n)).f$ のとき

$$\begin{aligned}[\![(\texttt{new}\ C(v_1, \ldots, v_n)).f_k]\!] &= [\![(\texttt{new}\ C(v_1, \ldots, v_n))]\!].f_k \\ &= \left\{f_i = [\![v_i]\!]; k = [\texttt{this} \mapsto ...]\,[\![C\#k]\!]_{cons}; m_j = [\texttt{this} \mapsto ...]\,[\![C\#m_j]\!]_{meth}\right\}.f_k \\ &\to_{\beta_{\lambda+}} [\![v_k]\!] \qquad OK\end{aligned}$$

- $e.m(\overline{e_i})$ で $e \to_{\beta_{FJ}} e'$ のとき

$$\begin{aligned}[\![e.m(\overline{e_i})]\!] &= [\![e]\!].m(\overline{[\![e_i]\!]}) \\ &\to_{\beta_{\lambda+}} [\![e']\!].m(\overline{[\![e_i]\!]}) \\ &= [\![e'.m(\overline{e_i})]\!] \qquad OK\end{aligned}$$

- $v.m(\ldots, e_i, \ldots)$ のとき

$$\begin{aligned}[\![v.m(\ldots, e_i, \ldots)]\!] &= [\![v]\!].m\ \cdots\ [\![e_i]\!] \cdots \\ &\to^*_{\beta_{\lambda+}} [\![v]\!].m\ \cdots\ [\![e'_i]\!] \cdots \qquad \text{(帰納法の仮定より)} \\ &= [\![v.m(\ldots, e'_i, \ldots)]\!] \qquad OK\end{aligned}$$

- $(\texttt{new}\ C(\overline{v_j})).m(w_1, \ldots, w_n)$ で $C\#m$ が $\texttt{def}\ m(\overline{x_i})\ e\ \texttt{end}$ のとき

$$(\texttt{new}\ C(\overline{v_j})).m(w_1, \ldots, w_n) \to_{\beta_{FJ}} [x_1 \mapsto w_1, \ldots, x_n \mapsto w_n, \texttt{this} \mapsto \texttt{new}\ C(\overline{v_j})]e$$

また、

$$\begin{aligned}
[\![(\texttt{new}\ C(\overline{v_j})).m(w_1,\ldots,w_n)]\!] &= \left\{\ldots;m_j = [\texttt{this}\mapsto ...]\ [\![C\#m_j]\!]_{meth}\right\}.m\ \ [\![w_1]\!]\ \cdots\ [\![w_n]\!] \\
&\to_{\beta_{\lambda+}} ([\texttt{this}\mapsto ...]\ [\![C\#m_k]\!]_{meth})\ \ [\![w_1]\!]\ \cdots\ [\![w_n]\!] \\
&= ([\texttt{this}\mapsto ...]\lambda x_1\ldots x_n.\ [\![e]\!])\ \ [\![w_1]\!]\ \cdots\ [\![w_n]\!] \\
&\to^*_{\beta_{\lambda+}} [x_1\mapsto\ [\![w_1]\!],\ldots,x_n\mapsto\ [\![w_n]\!],[\texttt{this}\mapsto ...]]\ [\![e]\!]
\end{aligned}$$

補題 4.1.2 より二つの式は等しくなる。よって　OK

- $\texttt{new}\ C(\ldots,e_i,\ldots)$ で $e_i\to_{\beta_{FJ}} e'_j$ のとき

$$\begin{aligned}
[\![\texttt{new}\ C(\ldots,e_i,\ldots)]\!] &= \left\{f_i = \ [\![e_i]\!];\texttt{init} = [\texttt{this}\mapsto ...]\ [\![C.k]\!]_{cons};m_j = [\texttt{this}\mapsto ...]\ [\![C.m_j]\!]_{meth}\right\} \\
&\to^*_{\beta_{FJ}} \left\{f_i = \ [\![e'_i]\!];\texttt{init} = [\texttt{this}\mapsto ...]\ [\![C.k]\!]_{cons};m_j = [\texttt{this}\mapsto ...]\ [\![C.m_j]\!]_{meth}\right\} \\
&= [\![\texttt{new}\ C(\ldots,e'_i,\ldots)]\!] \qquad OK
\end{aligned}$$

- それ以外のとき
  対応する簡約ルールがないので条件を満たさない。

定理 4.1.2.

e が簡約できないならば $[\![e]\!]$ も簡約できない.

(e が NF_{FJ} ならば $[\![e]\!]$ も $NF_{\lambda+}$)

証明:

- $e.f$ で e が簡約できないとき
  $[\![e.f]\!] = [\![e]\!].f$ となるが、帰納法の仮定より $[\![e]\!]$ は簡約できないので全体も簡約できない.
  OK

- $e.m(d_1,\ldots,d_n)$ で e とすべての d_i が簡約できないとき
  $[\![e.m(d_1,\ldots,d_n)]\!] = [\![e]\!].m\ [\![d_1]\!]\ \cdots\ [\![d_n]\!]$ となるが、帰納法の仮定より $[\![e]\!]$ および 各 $[\![d_i]\!]$ は簡約できないので全体も簡約できない.　　OK

- $new\ C(d_1,\ldots,d_n)$ で各 d_i が簡約できないとき

$$[\![new\ C(d_1,\ldots,d_n)]\!] = \left\{f_i = \ [\![d_i]\!];k = [\texttt{this}\mapsto ...]\ [\![C.init]\!]_{cons};m_j = [\texttt{this}\mapsto ...]\ [\![C.m_j]\!]_{meth}\right\}$$

  となるが、帰納法の仮定より各 $[\![d_i]\!]$ は簡約できないし、k や m_j は関数抽象 (つまり value) なので簡約できない。よって全体も簡約できない。　　OK

- x のとき
  $[\![x]\!] = x$ より OK

- 上記以外のときは簡約できる.

4.2 型安全性に関するうれしいこと

$\lambda+$ にいろいろな型システムを導入することが考えられる. そしてその型システムは本節の埋め込みを使って FJ にも適応できる.

定義 4.2.1. $\lambda+$ 上の任意の型システム τ に対して、*FJ* 上の型システムを次で定義する.

$$e : T \iff [] \vdash [\![e]\!] : T$$

定理 4.2.1 (Progress の伝搬)**.**

$$\lambda+ \text{で } Progress \text{ ならば } FJ \text{ でも } Progress$$

証明:
$e : T$ ならば e が value または $\exists e', e \to_{\beta_{FJ}} e'$ であることを証明する.
定義より $[\![e]\!] : T$ だが、τ の Progress 性より $[\![e]\!]$ が value または $\exists t'$, $[\![e]\!] \to_{\beta_{\lambda+}} t'$. value のときは e も value になるから O.K. $[\![e]\!] \to_{\beta_{\lambda+}} t'$ のときはもし e が簡約できないとすると定理 4.1.2 より矛盾してしまうので e は簡約できる。よって O.K.

定理 4.2.2 (Preservation の伝搬)**.**

$$\lambda+ \text{で } Preservation \text{ ならば } FJ \text{ でも } Preservation$$

証明:
$e : T, e \to_{\beta_{FJ}} e'$ ならば $e' : T$ を証明する.
定義より $[\![e]\!] : T$ だし、定理より $[\![e]\!] \to_{\beta_{\lambda+}} [\![e']\!]$ となるが、τ の Preservation 性より $[\![e']\!] : T$ が成り立ち、つまり $e' : T$ が成り立つ。

5 これから考えること

- $\lambda+$の柔軟な型付け
- λC との対応

参考文献

[1] A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight Java: A minimal core calculus for Java and GJ. ACM Transactions on Programming Languages and Systems, 23(3):396-450, 2001.

[2] Benjamin Pierce. Types and Programming Languages. The MIT Press, 2002.

[3] Satoshi Ogasawara. 知ってる？オブジェクトってね..「まぁ、そんなもんでしょう」. http://d.hatena.ne.jp/osiire/20090507 , はてなダイアリー, 2009.