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を使ったFJへの構造的部分型の導入

今井 宣洋 *

平成 21 年 12 月 15 日

概 要

先日 定義した FJ からラムダ計算の変換 [3] を使って FJ[1] に構造的部分型を伴う型システム (適当な表現?) を考えてみた. 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/doc2.tex` から最新のものが得られるかもしれない.

1 はじめに

1.1 目的

Ruby などの動的型を持つオブジェクト指向言語は便利でお手軽なため多岐に利用されているが, 静的なチェックがないために単純なミスによる不具合 (バグ) が起こることがある. でも静的なチェックによって Ruby の柔軟性が失われることは嫌だ.

本文書ではなるべく Ruby の柔軟性を生かしつつ, しかしより高信頼なプログラム開発を実現する検証器を実装するために型システムを考えた. この検証器を使えば、存在しないメソッドの呼び出しや引数の数の間違いを防ぐだけでなく, インターフェイスの記述によって大規模開発や拡張を可能とすると考えている.

1.2 例

この型システムを使えば次の様なコードが検証できる. 次の FJ のコードの例を見てみよう.

まず Duck と Car という class を定義したとする.

```
class Duck
  def sound
    'クワックワッ'
  end
  def swim
    ...
  end
end
```

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

```
class Car
  def sound
    'ブーブー'
  end
  def putOil(oil)
    ...
  end
end
```

両クラスとも sound というメソッドを持っている. ここで, 引数で与えられたオブジェクトの sound メソッドを呼ぶ関数を次の様に定義した.

```
def test(x)
  puts x.sound
end
```

このとき以下の実行は両方共有効なはずである.

```
test(new Duck)
test(new Car)
```

Java などの静的型チェックはこのようなコードをはじいてしまうが、本型システムはこれを型付け可能である. もちろんメソッドの呼び出しが定義されていない場合は型エラーである.

1.3 話の流れ

2 節では前回の簡単なおさらいをする. 3 節では型システムの定義をし, 型安全性を証明する.

2 前回までのあらすじ

2.1 FJ の定義

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

2.2 $\lambda+$ の定義

$$
t := \ldots \mid t.l \mid \left\{\overline{l_i = t_i}\right\}
$$

$$
\frac{t \to t'}{t.l \to t'.l} \qquad \frac{}{\left\{\overline{l_i = v_i}\right\}.l_k \to v_k} \qquad \frac{t \to t'}{\{.., l = t, ..\} \to \{.., l = t', ..\}}
$$

2.3 うれしい性質

定理 2.3.1. 任意の $\lambda+$ での型システム τ に対して,*FJ* での型システム τ_{FJ} が構成でき, τ で型安全性が成り立つなら τ_{FJ} でも型安全性が成り立つ.

3 $\lambda+$ への柔軟な型付定義

3.1 型付ルール

定義 3.1.1 (型表現)**.**

$$T := ... \mid \vdash l_1 : T_1, .., l_n : T_n \dashv$$

定義 3.1.2 (ルール)**.** 通常の単純型付けラムダ計算に以下の型付けルールを追加する. *TAPL[2] 11* 節とちょっと違うところに注意して欲しい.

$$\frac{\Gamma \vdash t : \vdash .., l : T, .. \dashv}{\Gamma \vdash t.l : T}$$

$$\frac{\forall i, \Gamma \vdash t_i : T_i \text{かつ } i_k \in \{1, .., n\}}{\Gamma \vdash \{l_1 = t_1, .., l_n = t_n\} : \left\vdash \overline{l_{i_k} : T_{i_k}} \right\dashv}$$

3.2 型安全性

定理 3.2.1 (Progress)**.** $\vdash t : T$ ならば t は *value* または $\exists t', t \to t'$

証明:
よーしパパ t の構造に関する帰納法で証明しちゃうぞー.

- $t.l$ のとき
  $\Gamma \vdash t.l : T$ より t も welltyped. 帰納法の仮定より t は value か $t \to t'$.
  - t が value のとき
    $t = \left\{\overline{l_i = v_i}\right\}$ (各 v_iは *value*) と表すことができる. しかも well typed 性より $l \in \{l_i\}$ となっているはずである. よって $t.l \to v_i$ となるから O.K.
  - $t \to t'$ のとき
    $t.l \to t'.f$ となるから O.K.
- $\left\{\overline{l_i = t_i}\right\}$ のとき
  仮定より $\exists \{i_k\}, \forall i, \Gamma \vdash t_i : T_i$ で $\Gamma \vdash \left\{\overline{l_i = t_i}\right\} : \left\vdash \overline{l_{i_k} : T_{i_k}} \right\dashv$. ここで帰納法の仮定よりすべての t_i は value か $t_i \to t_i'$ となるかのどっちかである. 全部 value なら、全体も value になるので O.K. どこかの k に対して $t_k \to t_k'$ となるならその k に関して $\{.., l_k = t_k, ..\} \to \left\{.., l_k = t_k', ..\right\}$ となるから O.K.

定理 3.2.2 (Preservation)**.** $\Gamma \vdash t : T$ かつ $t \to t'$ ならば $\Gamma \vdash t' : T$

証明:

- $t.l$ で $t.l \to$ なにか のとき
  $\Gamma \vdash t : \vdash .., l : T, .. \dashv$.
  - $t \to t'$ となる t' が存在して $t.l \to t'.l$ となるとき
    帰納法の仮定より $\Gamma \vdash t' : \vdash .., l : T, .. \dashv$ となるので, $\Gamma \vdash t'.l : T$ となる. O.K.
  - $t = \left\{\overline{l_i = v_i}\right\}$ で $t.l \to v_k$ のとき
    仮定から $\Gamma \vdash \{.., l : v_k, ..\} : \vdash .., l : T, .. \dashv$. よって $\Gamma \vdash v_k : T$. O.K.
- $\left\{\overline{l_i = t_i}\right\}$ で $t_k \to t'_k$ のとき型付け規則より $\exists T_k, \Gamma \vdash t_k : T_k$ となるが, 帰納法の仮定より $\Gamma \vdash t'_k : T_k$ ともなる. よって $\Gamma \vdash \left\{.., l_k = t'_k, ..\right\} : T$ となる. O.K.

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] Yoshihiro Imai. 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. http://ocaml-nagoya.g.hatena.ne.jp/yoshihiro503/20091207, はてなグループ, ocaml-nagoya グループ, 2009.