Path resolution for recursive modules

Jacques Garrigue

with Keiko Nakata

http://www.math.nagoya-u.ac.jp/~garrigue/papers/

Path Resolution for Nested Recursive Modules

中田景子と共著 [HOSC (2012)]

- 課題: 再帰モジュールのパスの正規形の存在を判定する
  (等価性に必要なため)
- 任意の再帰パスを許した一階の関数的 (strongly applicative)
  ファンクターと入れ子モジュールにおいて判定不能
  （任意のＴＭが表現できる）
- 引数のサブモジュール指定を有限な深さに制限すると判定可能

MLモジュールとは

コードを再利用可能な形で構造化するための機構

- **モジュール言語とコア言語が分かれている**
- **モジュールによる型の抽象化（隠蔽）：抽象データ型**
- **モジュールの入れ子**
- **モジュール間の関数としてのファンクター**
- **モジュールとファンクターの型：シグネーチャ**

OCamlの関数的ファンクターは弱い

関数的ファンクターを利用すると、同じファンクターを同じモジュールに二回適用する、同じ抽象型が生成される。

しかし、OCamlの関数的ファンクターは弱い。引数としてのモジュールの等価性は文字列として決まる。モジュール間の等式は考慮されない。

```
module Int = struct type t = Int of int end
module I = Int
module F(X : sig type t end) = struct type t = A of X.t end
module F1 = F(Int)
module F2 = F(Int)
module F3 = F(I)
```

`F1` と `F2` は互換性があるが、`F3` はない。直感と矛盾しており、説明しにくい現象がある。

OCamlの再帰モジュールも弱い

OCamlにおいて、再帰モジュールのサポートが断片的

- 公式な仕様がない
- 型推論が不完全: 型チェックが理由なく失敗したり、止まらなかったりする
- どのプログラムが受理されるか予測しにくい

8年前に使えた正しそうなプログラムは現在のOCamlでコンパイルできない

まず完全な型推論を提供したい

パスの正規化問題

以下の言語は一階ファンクターを含む言語の再帰的なシグネーチャを定義している

Access sig	S	$::=$	$\{m_1 : S_1 \cdots m_n : S_n\}$
Expression	e	$::=$	$\{m_1 = e_1 \cdots m_n = e_n\} \mid \lambda(x : S)e \mid p$
Path	p	$::=$	$vp \mid rp$
Variable path	vp	$::=$	$x \mid vp.m$
Rooted path	rp	$::=$	$\epsilon \mid rp(p) \mid rp.m$

モジュールと型宣言を区別しないで、どちらの等式も扱う

この言語で書かれたプログラムにおいて、任意のパスの正規化問題を考える

パス書き換え系

任意のプログラムについて、対応するパス書き換え系をルートϵから始まるパスの書き換え規則の集合として定義する

$$\begin{array}{l}\{\ \mathtt{m}_1 = \{\ \mathtt{n}_1 = \{\mathtt{n} = \{\}\}\ \ \mathtt{n}_2 = \epsilon.\mathtt{m}_1.\mathtt{n}_1\ \} \\ \quad \mathtt{m}_2 = \lambda(\mathtt{x})\{\ \mathtt{n}_1 = \{\}\ \ \mathtt{n}_2 = \mathtt{x}.\mathtt{n}_2\ \ \mathtt{n}_3 = \epsilon.\mathtt{m}_2(\mathtt{x}).\mathtt{n}_1\ \} \\ \quad \mathtt{m}_3 = \epsilon.\mathtt{m}_2(\epsilon.\mathtt{m}_1).\mathtt{n}_2\ \}\end{array}$$

のパス書き換え系は

$$\begin{array}{l}\{\ \epsilon.\mathtt{m}_1.\mathtt{n}_2 \longrightarrow \epsilon.\mathtt{m}_1.\mathtt{n}_1, \qquad \epsilon.\mathtt{m}_2(\mathtt{x}).\mathtt{n}_2 \longrightarrow \mathtt{x}.\mathtt{n}_2, \\ \quad \epsilon.\mathtt{m}_2(\mathtt{x}).\mathtt{n}_3 \longrightarrow \epsilon.\mathtt{m}_2(\mathtt{x}).\mathtt{n}_1,\ \ \epsilon.\mathtt{m}_3 \longrightarrow \epsilon.\mathtt{m}_2(\epsilon.\mathtt{m}_1).\mathtt{n}_2\ \}\end{array}$$

一般的な判定可能性

引数のサブモジュールが自由に指定できる場合

- 停止性が一般的に判定不能
- 入れ子かファンクターのどちらかがないと判定可能

証明:ファンクターと入れ子のどちらもが**１つのスタック**を表現できる。
PDAが無限な語を受理するかどうかは判定可能だが、**２つのスタック**を持ったオートマトンはチューリング機械になる

サブモジュール指定を制限するとどうなる？

サブモジュール指定の制限

引数に付けるアクセス・シグネーチャに沿って、サブモジュールが正しくアクセスされることを強制する

$$\{m_1 = \lambda(x : \{\})\{m_2 = x \quad m_3 = \epsilon.m_1(x).m_2.m_4\} \quad m_5 = \{\}\}$$

ここで x にサブモジュールがないので、以下は不正アクセスの例になる

$$\epsilon.m_1(x).m_2.m_4 \longrightarrow x.m_4 \longrightarrow \mathrm{error}$$

しかし、構文からも、基底パスの簡約列からも違反は分からない

Safe reduction : 3種類の判定を使う

$$\textbf{r-src}\frac{P \vdash p \mapsto (\theta, e) \quad P \vdash \theta \textbf{ safe} \quad e \textbf{ not a path}}{P \vdash p \downarrow p} \qquad \textbf{r-vp}\frac{ap \in \textbf{sig}_P(x)}{P \vdash x.ap \downarrow x.ap}$$

$$\textbf{r-exp}\frac{P \vdash p \mapsto (\theta, p') \quad P \vdash \theta \textbf{ safe} \quad P \vdash \theta(p') \downarrow q}{P \vdash p \downarrow q}$$

$$\textbf{r-dot}\frac{P \vdash p \downarrow p' \quad P \vdash p'.m \downarrow q}{P \vdash p.m \downarrow q} \qquad \textbf{r-app}\frac{P \vdash p_1 \downarrow p_1' \quad P \vdash p_1'(p_2) \downarrow q}{P \vdash p_1(p_2) \downarrow q}$$

$$\textbf{s-rec}\frac{P \vdash p \downarrow q \quad P \vdash q.m_i : S_i \ (1 \le i \le n)}{P \vdash p : \{m_1 : S_1 \ \ldots \ m_n : S_n\}}$$

$$\textbf{s-subst}\frac{P \vdash p_i : \textbf{sig}_P(x_i) \quad (1 \le i \le n)}{P \vdash [x_1 \mapsto p_1, \ldots, x_n \mapsto p_n] \textbf{ safe}}$$

代入補題の証明

補題 1 (substitution) $P \vdash p : S$ **かつ** $P \vdash \theta$ **safe ならば** $P \vdash \theta(p) : S$

証明は safe reduction, safety for a signature および safe substitution による相互帰納法

Coqによる形式化

- **ファンクターの引数は de Bruijn 添字**
- **代入はパスに対してのみなので、複雑な操作は要らない**
- **１週間で証明が完了**

Safe reduction の停止性判定

ある（再帰シグネーチャを表現した）プログラムが発散するパスを含むかどうかを調べる手順

- プログラムの各等式を位置で修飾する
- 同じ位置が２回簡約されないことを調べる

Scalaで使われる方法だが、これでは健全であっても完全ではない

$$\{f = \lambda(x : \{\})x^1 \quad a = \{\} \quad n = f(f(a))^2\}$$

ここで n の簡約は次のように進む

$$n \to f(f(a))^2 \to (f(a)^1)^2 \to ((a^1)^1)^2$$

等式１は２回使われたが、正規形である

アイデア 1: call-by-value

評価戦略を call-by-value に変えることで前の例が扱える

$$n \to f(f(a))^2 \to f(a^1)^2 \overset{\sharp}{\to} f(a)^2 \to (a^1)^2$$

($\overset{\sharp}{\to}$ は正規形のラベルを忘れる)

しかし、まだ完全ではない

$$\begin{array}{ll} \{ & f = \lambda(x : \{m : \{\}\})\{m = x.m^1\} \\ & a = \{m = \{\}\} \quad n = f(f(a)).m^2 \quad \} \end{array}$$

n の簡約は次のように進む

$$n \to f(f(a)).m^2 \to (f(a).m^1)^2 \to ((a.m^1)^1)^2$$

ここでも、等式１を２回簡約した

アイデア 2: η-expansion

カリー化によって、f の引数の入れ子モジュールを η-拡大する
指定できる各サブモジュールに対して異なる引数を導入することになる

$$\begin{array}{ll}\{ & f = \lambda(x_\epsilon : \{\})\lambda(x_m : \{\}).\{m = (x_m)^1\} \\ & a = \{m = \{\}\} \\ & n = f(f(a)(a.m))(f(a)(a.m).m).m^2 \quad \}\end{array}$$

これで評価がする

$$\begin{array}{l} n \to f(f(a)(a.m))(f(a)(a.m).m).m^2 \\ \quad \to f(f(a)(a.m))(a.m^1).m^2 \\ \quad \overset{\sharp}{\to} f(f(a)(a.m))(a.m).m^2 \to (a.m^1)^2 \overset{\sharp}{\to} a.m \end{array}$$

次ページの正規化アルゴリズムはこのη-拡大を全ての指定パスとその正規形からなる指定代入を呼び出し時に計算することで行っている

Semi-ground normalization

sgnlz(P, π, q) =
  match q with
  | $\mathtt{x} \Rightarrow \mathtt{x}$
  | $\epsilon \Rightarrow \epsilon$
  | $q_1.m \Rightarrow$ expand$(P, \pi,$ sgnlz$(P, \pi, q_1).m)$
  | $q_1(q_2) \Rightarrow$ expand$(P, \pi,$ sgnlz$(P, \pi, q_1)(q_2))$

expand(P, π, x.ap) =
  if $ap \in$ sig$_P(\mathtt{x})$ then $\mathtt{x}.ap$ else error
expand(P, π, rp) =
  let (θ, e) = lookup(P, rp) in
  let ρ = vp_subs(P, π, θ) in
  match e with
  | $q^i \Rightarrow$ if $i \in \pi$ then error else
        subs$(\theta, \rho,$ sgnlz$(P, \{i\} \cup \pi, q))$
  | $\{m_1 = e_1 \ldots m_n = e_n\} \Rightarrow rp$
  | $\lambda(x)e' \Rightarrow rp$

subs$(\theta, \rho, rp) = \theta(rp)$
subs(θ, ρ, vp) =
  if $vp \in$ dom(ρ) then $\rho(vp)$ else vp

vp_subs$(P, \pi, id) = id$
vp_subs$(P, \pi, \theta[\mathtt{x} \mapsto p])$ =
  sig_subs$(P, \pi, \mathtt{x}, p,$ sig$_P(\mathtt{x}))$
  $\cup$ vp_subs(P, π, θ)

sig_subs$(P, \pi, vp, rp, \{\})$ =
  $[vp \mapsto$ sgnlz$(P, \pi, rp)]$
sig_subs$(P, \pi, vp, \mathtt{x}.ap, \{\})$ =
  if $ap \in$ sig$_P(\mathtt{x})$ then $[vp \mapsto \mathtt{x}.ap]$
  else error
sig_subs$(P, \pi, vp, p, \{m_1 : S_1\} \uplus S)$ =
  sig_subs$(P, \pi, vp.m_1, p.m_1, S_1)$
  $\cup$ sig_subs(P, π, vp, p, S)

結果

定理 1 (健全性) $\mathbf{sgnlz}(P, \pi, p) = q$ **ならば** $P \vdash p \downarrow q$**.**

証明 関数帰納法で **Coqで検証済み**

定理 2 (停止性) $\mathbf{sgnlz}(P, \pi, p)$ **は必ず停止する**

証明 簡単な測度を定義する

補題 3 (postponement) p **が**ϵ**で始まるパスで、**$\mathbf{sgnlz}(P, \pi', p) = q$ **および** $\mathbf{vp\_subs}(P, \pi, \theta) = \rho$ **かつ** $\pi \subset \pi'$ **ならば**$\mathbf{sgnlz}(P, \pi, \theta(p)) = \mathbf{subs}(\theta, \rho, q)$**.**

証明 **Coqで検証済み**

定理 3 (完全性) P **は safe で** $P \vdash p : \{\}$ **ならば** $\mathbf{sgnlz}(P, \emptyset, p) \neq \mathsf{error}$**.**

証明 関数帰納法で、safe rewriting を使う

プログラミング言語への応用

ここで紹介されたアプローチを利用すれば

- **強い**関数的ファンクターを持った言語において
- **再帰**シグネーチャに対して**決定可能**な正規化が可能
- 指定用シグネーチャは既に書かれているので負担にならない
- ただし、**一階**のファンクターに制限される