定理証明系を用いた強正規化性の証明と評価器の抽出

廣田知子　(指導教員：浅井健一)

1　概要

信頼性の高いプログラムを得る一つの手法として、定理証明系を用いて実行結果の存在定理を証明し、そこからプログラムを抽出する方法がある。これによって得られたプログラムはその正当性が保証されているが、一方でプログラムの抽出を行うのは簡単ではない。

そこで本研究では、プログラム抽出を容易に扱える方法を探る一環として、評価器を抽出して知見を得ることを目指した。(評価器は、プログラムをそれ以上評価出来なくなるまで β-簡約を行うプログラム変換器を指す。) まず単純型付き λ 計算を定理証明系 Coq を用いて定式化し (定式化方法は [1] を参考にした)、次に、論理述語を用いての Tait による強正規化性の証明 [5] を Coq で定式化することにより、正当性が保証された評価器を Ocaml のプログラムとして抽出した。

2　強正規化性

強正規化性とは、プログラムを任意の順序で β-簡約していっても必ず簡約が停止する性質のことである。ただし本研究では call-by-value left-to-right の簡約規則を採用したため、簡約順序は一意に決まる。

3　Locally Nameless 手法

本研究では Locally Nameless 手法を用いて λ 抽象を定義した。この手法は、α-equality の問題を解消するための、変数の名前付けの方法であり、自由変数には $x, y, ...$ 等の名前を付け、束縛変数には名前を付けずに de Bruijn index を用いて表現する。de Bruijn index は変数を 0 以上の整数で表し、その式における一番内側の binder の変数名を 0 と考え、外にいくに従って 0, 1, 2,... と大きくしていくという表現方法である。例えば λ 計算式の $\lambda x.\lambda y.\lambda z.y\,x\,w\,z$ は Locally Nameless 手法を用いて表すと、$\lambda.\lambda.\lambda.1\,2\,w\,0$ となる。

4　単純型付き λ 計算

本研究で使用したシステムは以下の通り。

4.1　構文

$$
\begin{array}{lrcll}
\text{項}: & e & := & n & \text{(束縛変数)} \mid x \ \text{(自由変数)} \\
 & & \mid & \lambda.e & (\lambda\text{抽象}) \mid e_1\,e_2 \ \text{(関数適用)} \\
\text{値}: & v & := & x \mid \lambda.e & \\
\text{型}: & T & := & A \mid T_1 \to T_2 &
\end{array}
$$

4.2　簡約規則 (call-by-value left-to-right)

one-step reduction $\rightsquigarrow$ と big-step reduction $\rightsquigarrow^*$ の定義を以下に示す。

one-step reduction rule:

$$
\begin{array}{rcll}
(\lambda.e)\,v & \rightsquigarrow & e^v & \\
e_1\,e_2 & \rightsquigarrow & e_1'\,e_2 & \text{if } e_1 \rightsquigarrow e_1' \\
v\,e_2 & \rightsquigarrow & v\,e_2' & \text{if } e_2 \rightsquigarrow e_2'
\end{array}
$$

ここで e^v は e の中に含まれる束縛変数 0 に v を代入したものを表している。

big-step reduction rule:

$$
\begin{array}{rcll}
e & \rightsquigarrow^* & e & \\
e_1 & \rightsquigarrow^* & e_2 & \text{if } e_1 \rightsquigarrow e_2 \\
e_1 & \rightsquigarrow^* & e_3 & \text{if } e_1 \rightsquigarrow e_2\,,\ e_2 \rightsquigarrow^* e_3
\end{array}
$$

4.3　型付け規則

本稿では、型規則は $\Gamma \vdash e : T$ のように表記する。

$$\frac{ok\ \Gamma \quad (x:T) \in \Gamma}{\Gamma \vdash x : T}\ (var)$$

$$\frac{\forall x \notin L. \quad \Gamma, x : T_1 \vdash e^x : T_2}{\Gamma \vdash \lambda.e : T_1 \to T_2}\ (fun)$$

$$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\,e_2 : T_2}\ (app)$$

例えば (var) の規則は「$\Gamma \vdash x : T$ が成り立つためには環境 Γ が ok であり、かつ x が T 型を持つという情報が Γ に含まれていなくてはならない」という意味を表す。Γ が ok であるとは、Γ 中に現れる変数名が全て異なることを意味している。又、(fun) に現れる $\forall x \notin L.$ は L に含まれない x (自由変数) を表し、この L には何も制限を付けていないので、自分で自由に L を定めることが出来る。x を他のどこにも現れない変数とするのでなく、このように L に関してのみの制限にとどめることにより、証明木を作成する場合等、型付け規則に関わる証明を行う際の複雑さを軽減することが出来る。本研究では定理 3 の証明において使用した。

5　停止性 N と論理述語 R

簡約の停止性を述語 N を用いて定義し、論理述語 R は N と型に関する帰納法を使って定義した。

定義 1 (停止性)

$N(e) \iff e \rightsquigarrow^* v$ となるような値 v が存在する

定義 2 (論理述語 R)

- $R_A(e) \iff N(e)$
- $R_{T_1 \to T_2}(e) \iff N(e)$ であり、かつ任意の項 e' について $R_{T_1}(e')$ ならば $R_{T_2}(e\,e')$ が成り立つ

Tait による本来の定義では「$R_T(e)$ において e の型は T である」の条件が付加されているが、本研究ではその制約をもうけていない。強正規化性を示す際に定理の条件部に型規則の成立を入れているため、そこで e と T の関係性を示すことになり、論理述語にその定義を入れる必要がないからである。

6 Coq による定式化

λ 計算の構文や型、簡約規則、型付け規則等は全て Inductive コマンドを用い、再帰を持つデータ型として定義した。述語 N の定義には Definition コマンドを用いたが、その際、N の返す型は、プログラム抽出を行うため、関数の型を表す Set 型とした。又、論理述語 R の定式化に関しては、通常ならば Inductive コマンドを使い、データ型として定義するところだが、R の再帰が negative に現れてしまい（含意の前件部の前提に相当する部分に「任意の x に対して自身の定義が成り立つ」という条件が現れてしまう）、Coq ではデータ型として定義出来ないため、Type 型を返す関数として R を定義した。([2] においても同様の手法が取られている。）なお、簡約規則の返す型は Prop 型で定義したが、型付け規則の返す型は N と R の定義に合わせ、Set 型とした。

7 強正規化性の証明

[5] で述べられている Tait の方法を参考に、Coq を用いて証明の定式化を行った。以下はその主となる定理である。

定理 1 $R_T(e)$ ならば、$N(e)$
定理 2 $e \rightsquigarrow^* e'$のとき、$R_T(e)$ と $R_T(e')$ は同値
定理 3 $x_1 : T_1, ..., x_n : T_n \vdash e : T$ かつ $v_1, ..., v_n$について $R_{T_1}(v_1), ..., R_{T_n}(v_n)$ が成り立つとき、$R_T([x_1 \mapsto v_1]...[x_n \mapsto v_n]\, e)$

定理 1 は R の定義により成立、定理 2 は型 T に関する帰納法で、そして定理 3 は途中で定理 2 を使いつつ、型付け規則の導出における帰納法を用いて証明を行った。

Coq での証明において、ある式に関する帰納法を使うと、抽出された関数ではその部分を、その式に関する場合分けを行うよう定義される。故に定理 3 の証明方法だと、プログラムとして抽出したときに型付け規則で場合分けを行うよう定義されてしまい、型付け規則が関数の引数としてどうしても必要になってしまう。これにより、出来れば、項とその型を引数として渡せば簡約を行ってくれるような評価器を抽出したいところだが、そうはならず、型付け規則を引数として必要とする評価器が抽出されることとなる。このような理由から、本当ならば定理 3 の証明は項に関する帰納法を用いて行うのが望ましいが、locally nameless 手法では、λ 抽象の body 部分をみるときに束縛変数を置き換えているため、単純な subterm にはなっておらず、項に関する帰納法をかけにくい。故に今回はやむなく型付け規則に関する帰納法を用いた。

そして停止性を示す強正規化性の定理は以下である。

定理 4 (強正規化性) $\vdash e : T$ ならば、$N(e)$
[証明] 定理 3 により、e が $R_T(e)$ を満たすことが言え、故に、定理 1 によって $N(e)$ が導ける。□

8 プログラムの抽出

定理 4 に関して Extraction コマンドを用いることにより、Ocaml のプログラムの抽出を行った。その結果、構文・型付け規則・N・R はデータ型として、そして定理 2・3 がサブ関数として、定理 4 が評価器の関数として抽出された。簡約規則・定理 1 は定義としてはプログラムには書かれず、吸収された形になっている。

9 まとめと今後の課題

本研究では単純型付き λ 計算における強正規化性の証明を Coq で定式化することによって評価器をプログラムとして抽出した。今後は、本研究の発展として、部分評価器の抽出や、λ 計算に let 文と継続を扱う命令 shift/reset を導入した型システムにおける強正規化性の証明 [3] を用いた評価器の抽出を行う予定である。

参考文献

[1] Aydemir, B., A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich “Engineering Formal Metatheory,” *Conference Record of the 35th Annual ACM Symposium on Princeples of Programming Languages*, pp. 3–15 (January 2008).

[2] Berger, U., S. Berghofer, P. Letouzey, and H. Schwichtenberg “Program extraction from normalization proofs,” *Studia Logica 82*, pp. 25–49 (February 2006).

[3] Biernacka, M., and D. Biernacki “Context-based proofs of termination for typed delimited-control operators,” *11th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (September 2009).

[4] Biernacka, M., O. Danvy, and K. Støvring “Program extraction from proofs of weak head normalization,” *Electronic Notes in Theoretical Computer Science 155*, pp. 169–189 (May 2006).

[5] Pierce, B. C. *Types and Programming Languages*, Cambridge: MIT Press (February 2002).