Agda を使った限定継続演算子の性質の証明

理学専攻　情報科学コース　2140669　本田 華歩　(指導教員：浅井 健一)

1　はじめに

継続をプログラムで扱うには control/prompt [5] や shift/reset [3] など継続を切り取って扱う限定継続演算子が使われる。その意味論は継続渡し形式 (continuation-passing style; CPS) に変換することで与えられる。CPS 変換 [6] の正当性は、変換前の簡約と変換後の簡約が対応することにより示され、これによりコンパイル前後でプログラムの持つ意味が保たれていることが示される。さらに reflection [7] と呼ばれる関係が示されることは、CPS 変換後に行える最適化が CPS 変換前にも行えるようになることを意味する。本研究では、定理証明支援系言語である Agda において control/prompt が含まれた体系での CPS 変換の正当性の証明を目指した。また、 shift/reset が含まれた体系に対して型付きで reflection が成り立つことを示し、その証明を Agda において定式化した。

2　control/prompt 入り体系での CPS 変換の正当性

2.1　DS 項の定義

CPS 変換前の項を DS 項という。以下に DS 項の定義を示す。

$$\begin{array}{rrcl} \text{型} & \tau & := & \mathsf{Nat} \mid \mathsf{Bool} \mid \tau_2 \to \tau_1\,(\mu_\alpha)\,\alpha\,(\mu_\beta)\,\beta \\ \mathit{trail} & \mu & := & \bullet \mid \tau \to \langle\mu\rangle\ \tau' \\ \text{値} & v & := & n \mid x \mid \lambda x.\, e \\ \text{項} & e & := & v \mid e_1 \mathbin{@} e_2 \mid e_1 + e_2 \mid \mathcal{F}c.\, e \mid \langle e \rangle \end{array}$$

型は自然数型、真偽値型、関数型の三つからなる。関数型は、τ_2 型の値を受け取ったら τ_1 型の値を返す関数だが、継続 ($\tau_1 \to \mu_\alpha \to \alpha$) と trail ($\mu_\beta$) の型の情報も持っている。trail は control/prompt に特有なもので、継続が連なった形をしている。$\tau \to \langle\mu\rangle\ \tau'$ は、τ から τ' への継続で、その際に μ 型の trail を受け取ることを意味する。($\bullet$) は空の trail を表している。値は自然数、変数、λ 抽象の三つからなる。項は値、関数適用、足し算、control、prompt からなる。型と trail、値と項はそれぞれ相互再帰の形で Agda に載せることができた。

2.2　型規則

$\Gamma \vdash e : \tau\,(\mu_\alpha)\,\alpha\,(\mu_\beta)\,\beta$ という型判定は、型環境 Γ のもとで、e が $\tau \to \mu_\alpha \to \alpha$ の型の継続と μ_β 型の trail を受け取り、β 型の値を返すことを意味している。

本稿で扱うインタプリタには、Cons, Append, id_k が入っている。id_k とは初期継続を表しており、渡す trail が空かそうでないかによって挙動が変わってくる。Cons と Append はそれぞれ継続と trail、trail と trail を繋ぐものである。Cons, Append, id_k の型の辻褄を合わせるために、id-cont-type と compatible という制約関数を作成している。control/prompt の型規則 [1] を以下に示す。control のインタプリタは、Cons, Append, id_k を全て含むため、前提に三つの制約が入る。prompt は id_k のための id-cont-type が前提に入っている。control/prompt 以外の型規則は先行研究と同様である。

$$\frac{\begin{array}{c}\mathsf{id\text{-}cont\text{-}type}(\gamma, \gamma', \mu_{id}) \\ \mathsf{compatible}(t_1 \to t_2\,\mu_1, \mu_2, \mu_0)\ \ \mathsf{compatible}(\mu_\beta, \mu_0, \mu_\alpha) \\ \Gamma, k : \tau \to t_1\,(\mu_1)\,t_2\,(\mu_2)\,\alpha \vdash e : \gamma\,(\mu_{id})\,\gamma'\,(\bullet)\,\beta\end{array}}{\Gamma \vdash \mathcal{F}c.\, e : \tau\,(\mu_\alpha)\,\alpha\,(\mu_\beta)\,\beta}\ (\mathsf{TControl})$$

$$\frac{\mathsf{id\text{-}cont\text{-}type}(\beta, \beta', \mu_{id})\ \ \Gamma \vdash e : \beta\,(\mu_{id})\,\beta'\,(\bullet)\,\tau}{\Gamma \vdash \langle e \rangle : \tau\,(\mu_\alpha)\,\alpha\,(\mu_\alpha)\,\alpha}\ (\mathsf{TPrompt})$$

型のついたプログラムを実行すると、定義された通りに型がついた値を返し、型エラーになることはない。つまり、インタプリタにおける型の対応を以下のように定義した時、定理 1 がいえる。

$$\begin{aligned} [\![\bullet]\!]_\mu &= \bullet \\ [\![\tau \to \tau'\ \mu]\!]_\mu &= [\![\tau]\!]_\tau \to [\![\mu]\!]_\mu \to [\![\tau']\!]_\tau \end{aligned}$$

$$\begin{aligned} [\![\mathsf{Nat}]\!]_\tau &= \mathbb{N} \\ [\![\mathsf{Bool}]\!]_\tau &= \mathbb{B} \\ [\![\tau_2 \to \tau_1\,(\mu_\alpha)\,\alpha\,(\mu_\beta)\,\beta]\!]_\tau &= \\ &\quad [\![\tau_2]\!]_\tau \to ([\![\tau_1]\!]_\tau \to [\![\mu_\alpha]\!]_\mu \to [\![\alpha]\!]_\tau) \to [\![\mu_\beta]\!]_\mu \to [\![\beta]\!]_\tau \end{aligned}$$

定理 1 $\Gamma \vdash e : \tau\,(\mu_\alpha)\,\alpha\,(\mu_\beta)\,\beta$ ならば $\Gamma^* \vdash \varepsilon[\![e]\!] : ([\![\tau]\!]_\tau \to [\![\mu_\alpha]\!]_\mu \to [\![\alpha]\!]_\tau) \to [\![\mu_\beta]\!]_\mu \to [\![\beta]\!]_\tau$

Agda に CPS インタプリタを載せられたことにより、定理 1 が証明できた。

2.3　CPS 変換の定式化

本稿では one-pass の CPS 変換 [4] を行う。CPS インタプリタを元にしてアンダーラインのついた式とオーバーラインのついた式に出力を区別して定義し、Agda に載せた。アンダーラインのついた式は dynamic な式と呼び、自分で定義した出力時の構文を表す。オーバーラインのついた式は static な式と呼び、今回では Agda の本当の式であり CPS 変換時に実行されることを表す。

2.4　正当性の証明

CPS 変換前の項を簡約した後に CPS 変換した項と、CPS 変換後の項が β 同値であること、つまり以下の予想の証明を目指した。

予想 2 任意の項 e, e' について $e \to e'$ が成り立つならば任意の *schematic* な継続 k に対して $[\![e]\!]\,\overline{@}\,k\,\overline{@}\,t =_\beta [\![e']\!]\,\overline{@}\,k\,\overline{@}\,t$ が成り立つ。

ここで $=_\beta$ は β 同値である。予想 2 は、$e \to e'$ の簡約の形によって場合分けして取り組み、control の場合を除いて Agda 上で証明を完成させた。control の場合では、任意の trail t と schematic な継続 k に対して $[\![\langle E_p[\mathcal{F}c.\, e_1]\rangle]\!]\,\overline{@}\,k\,\overline{@}\,t =_\beta [\![\langle(\lambda c.\, e_1) \mathbin{@} (\lambda x.\, E_p[x])\rangle]\!]\,\overline{@}\,k\,\overline{@}\,t$

が成り立つ事を示すということになる。そこでコンテキストで囲まれた左辺を CPS 変換の定義によって展開できる形に変換するために次の予想を立てた。

予想 3 (context-lemma) 任意の項 e、*trail* t、*schematic* な継続 k について $[\![\langle E_p[e]\rangle]\!]\, k\, t =_\beta [\![(\lambda x.\, E_p[x])\,@\,e]\!]\, k\, t$

予想 3 を展開すると、次の重要な予想を示すことで証明が完成することが分かった。

予想 4 (k の移動) 任意の項 e、k、*trail* t に対して $\lambda v'.\, \lambda k'.\, \lambda t'.\, [\![e]\!][v'/x](\lambda v_0.\, (\lambda t_0.\, \mathit{id}_k\, \overline{@}\, v_0\, \overline{@}\, t_0))\ (k' :: t') =_\beta \lambda v'.\, \lambda k'.\, \lambda t'.\, [\![e]\!][v'/x](\lambda v_0.\, (\lambda t_0.\, \mathit{id}_k\, \overline{@}\, v_0\, \overline{@}\, (k' :: t_0)))\ t'$ が成り立つ。

予想 4 には論理関係の定義が必要になると考えており、これを証明することが今後の課題となっている。

3 shift/reset 入り体系での reflection 証明

3.1 reflection とは

Reflection [7] というのは、ソース言語とターゲット言語とその間の変換についての性質である。本稿では、ソース言語は shift/reset の入った単純型付き λ 計算、ターゲット言語は単純型付き λ 計算とする。前者は直接形式で書かれているので DS 言語、後者は継続渡し形式で書かれたプログラムになるので CPS 言語、と呼ぶことにする。DS 言語の項 M を CPS 言語に変換するのは CPS 変換と呼ばれ M^* と記述する。一方、CPS 言語の項 N を DS 言語に逆変換するのは DS 変換と呼ばれ $N^\#$ と記述する。このような設定のもとで DS 言語と CPS 言語の間に reflection の関係が成り立つとは以下のように定義される。

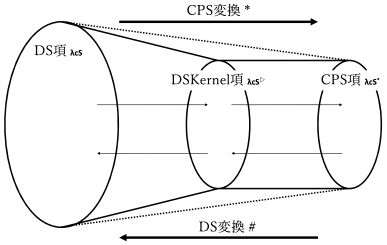
定義 5 (reflection) CPS 変換 $\cdot^*$ と DS 変換 $\cdot^\#$ が以下の 4 条件を満たすとき reflection をなすという。

1. 任意の DS 項 M について $M \longrightarrow^* M^{*\#}$
2. 任意の CPS 項 N について $N^{\#*} = N$
3. 任意の DS 項 M, M' について $M \longrightarrow M'$ ならば $M^* \longrightarrow^* M'^*$
4. 任意の CPS 項 N, N' について $N \longrightarrow N'$ ならば $N^\# \longrightarrow^* N'^\#$

ここで $\longrightarrow$ は 1 ステップの簡約、$\longrightarrow^*$ は 0 ステップ以上の簡約を示す。上記の性質のうち、3 は従来の CPS 変換の簡約保存を意味している。

3.2 DSKernel 項とは

ある DS 項を CPS 変換して更に DS 変換を行うと、もとの DS 項の部分式に名前を与えるような項、つまり A-正規形になる。これを DSKernel 項として定義している。例えば DS 項 $f\,\underline{@}\,(g\,\underline{@}\,h)$ の CPS 変換後の形は、$\lambda k.\, g\,\underline{@}\,h\,\underline{@}\,(\lambda y.\, f\,\underline{@}\,y\,\underline{@}\,k)$ になる。これに対して更に DS 変換を行うと、$\underline{\mathsf{let}}\ y = g\,\underline{@}\,h\ \underline{\mathsf{in}}\ f\,\underline{@}\,y$ という DSKernel 項になるが、これはもとの DS 項 $f\,\underline{@}\,(g\,\underline{@}\,h)$ の部分式 $g\,\underline{@}\,h$ に y という名前を与えたものになっている。それぞれの項の関係性をまとめると以下の図になる。



DSKernel 言語と CPS 言語の間の変換は自明な変換になる。つまり従来の CPS 変換とは A-正規形変換と、DSKernel 言語を CPS に変換する自明な変換の合成と表現される。

3.3 それぞれの言語の定式化

それぞれの言語の定義は、基本的には Biernacki ら [2] の定義と同じである。そこに型システムを作り Agda に載せることによって定式化を行った。

CPS 言語の型システムでは継続 k の型を引き回している。これは継続の型情報が他の項の型を決めるのに必要なためである。また、この k の型は環境には入れず特別な定数として管理しているが、これは DSKernel 項から CPS 項への変換時に型の整合性を保つための工夫である。

3.4 reflection の証明

reflection の証明は、DS 項と DSKernel 項間の reflection 証明、DSKernel 項と CPS 項間の reflection 証明に分けて、最後に二つの証明を合成する方法で行っている。証明する性質によっては代入補題等の補題が必要となるものも多くあったため、それらの補題も全て Agda 上に載せている。

4 まとめ

control/prompt を含む体系においては Agda を使って CPS インタプリタと CPS 変換を定式化し、型規則を Agda に載せたことで CPS インタプリタ前後での型の整合性を証明することができた。CPS 変換の正当性の証明は control の場合を除いて証明を定式化したが、予想 4 を証明するという課題が残った。

shift/reset を含む体系においては reflection の証明を型付きで Agda に載せることで証明を定式化した。

参考文献

[1] Kenichi Asai, Youyou Cong, and Chiaki Ishio. A functional abstraction of typed trails. In *ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM 2021)*, 2021.

[2] Dariusz Biernacki, Mateusz Pyzik, and Filip Sieczkowski. A reflection on continuation-composing style. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pp. 18:1–18:17, 2020.

[3] O. Danvy and A. Filinski. Abstracting Control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160, 1990.

[4] O. Danvy and A. Filinski. Representing Control: a Study of the CPS Transformation. *Mathematical Structures in Computer Science*, Vol. 2, No. 4, pp. 361–391, 1992.

[5] Mattias Felleisan. The theory and practice of first-class prompts. In *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 180–190, 1988.

[6] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical computer science*, Vol. 1, No. 2, pp. 125–159, 1975.

[7] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 19, No. 6, pp. 916–941, 1997.