control/prompt 入り CPS 変換の正当性の証明

本田 華歩　(指導教員：浅井 健一)

1　はじめに

プログラムにおける「継続」とは、計算中のある時点における残りの計算のことを指す。継続をプログラム中で扱うには control/prompt [4] や shift/reset [2] 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が、これまで CPS 変換の正当性は shift/reset 入りの体系において議論されてきた。 [5, 6]

また、型システムにおいては、control/prompt が含まれる体系は、shift/reset が含まれる体系と違い、継続が連なっている trail という新しい概念を取り入れなければならない。そのため CPS 変換においても扱う型が増え、正当性の証明は複雑になる。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 インタプリタ並びに CPS 変換を定式化し、CPS 変換の正当性を証明することを目指す。

2　DS 項と CPS インタプリタ

本節では CPS インタプリタを Agda で定式化する。

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。

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

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 (μ_β) 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。$\tau \to \tau'\ \mu$ 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　CPS インタプリタ

CPS 変換を定式化する前に、CPS インタプリタを Agda に載せた。CPS インタプリタの定義を以下に示す。

$$\begin{aligned} \varepsilon[\![x]\!]\,\rho\,k\,t &= k(\rho(x))\,t \\ \varepsilon[\![\lambda x.\, e]\!]\,\rho\,k\,t &= k(\lambda v.\, \lambda k'.\, \varepsilon[\![e]\!]\,\rho[v/x]\,k')\,t \\ \varepsilon[\![e_1 \mathbin{@} e_2]\!]\,\rho\,k\,t &= \varepsilon[\![e_1]\!]\,\rho\,(\lambda v_1.\, \varepsilon[\![e_2]\!]\,\rho\,(\lambda v_2.\, v_1\ v_2\ k))\,t \\ \varepsilon[\![e_1 + e_2]\!]\,\rho\,k\,t &= \varepsilon[\![e_1]\!]\,\rho\,(\lambda v_1.\, \varepsilon[\![e_2]\!]\,\rho\,(\lambda v_2.\, k(v_1 + v_2)))\,t \\ \varepsilon[\![\mathcal{F}c.\, e]\!]\,\rho\,k\,t &= \varepsilon[\![e]\!]\,\rho[\lambda v.\, \lambda k'.\, \lambda t'.\, k\,v\,(t \mathbin{@}_p k' :: t')/c]\,\mathsf{id}_k\,\mathsf{id}_t \\ \varepsilon[\![\langle e \rangle]\!]\,\rho\,k\,t &= k(\varepsilon[\![e]\!]\,\rho\,\mathsf{id}_k\,\mathsf{id}_t)t \end{aligned}$$

ここで、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。型付きの id_k において型の辻褄を合わせるために is-id-trail という名前で制約をかける関数を作っている。下に id_k の定義と is-id-trail の実際のコードを示す。is-id-trail の 3 つの引数は順に、id_k の一つ目の引数の型、id_k の返す値の型、id_k の二つ目の引数の型に対応している。同様に Cons (::) と Append $(@_p)$ はそれぞれ継続と trail、trail と trail を繋ぐものである。id_k と同様に型を合わせるために compatible という名前で制約をかける関数を作っており、compatible の各行が Cons, Append の 4 行と対応している。

$$\begin{aligned} \mathsf{id}_k\ v\ \mathsf{id}_t &= v \\ \mathsf{id}_k\ v\ k &= k\ v\ \mathsf{id}_t \end{aligned}$$

```
is-id-trail : typ → typ → trail → Set
is-id-trail τ τ' • = τ ≡ τ'
is-id-trail τ τ' (τ₁ ⇒ τ₁' , μ) = (τ ≡ τ₁) × (τ' ≡ τ₁') × (μ ≡ •)
```

$$\begin{aligned} k :: \mathsf{id}_t &= k \\ k :: k' &= \lambda v.\, \lambda t'.\, k\,v\,(k' :: t') \\ \mathsf{id}_k \mathbin{@}_p t &= t \\ k \mathbin{@}_p t &= k :: t \end{aligned}$$

```
compatible : trail → trail → trail → Set
compatible • μ₂ μ₃ = μ₂ ≡ μ₃
compatible (τ₁ ⇒ τ₁' , μ₁) • μ₃ = (τ₁ ⇒ τ₁' , μ₁) ≡ μ₃
compatible (τ₁ ⇒ τ₁' , μ₁) (τ₂ ⇒ τ₂' , μ₂) • = ⊥
compatible (τ₁ ⇒ τ₁' , μ₁) (τ₂ ⇒ τ₂' , μ₂) (τ₃ ⇒ τ₃' , μ₃) =
          (τ₁ ≡ τ₃) × (τ₁' ≡ τ₃') ×
          (compatible (τ₂ ⇒ τ₂' , μ₂) μ₃ μ₁)
```

2.3　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。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 is-id-trail 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。

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

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

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。

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

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

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。

3 CPS 変換の定式化

本稿では one-pass の CPS 変換 [3] を行う。2 節での 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。

4 CPS 変換の正当性の証明

4.1 代入規則と簡約規則

正当性を証明するために、DS 項、CPS 項それぞれにおいて代入規則と簡約規則を Agda に載せた。これまでの β 簡約と足し算の簡約規則に、下に示す control/prompt の簡約規則を加えた。

$$\begin{array}{lcll} \langle E[\mathcal{F}c.\,e_1]\rangle & \to & \langle(\lambda c.\,e_1)\,@\,(\lambda x.\,E[x])\rangle & (\mathsf{RControl}) \\ \langle v_1\rangle & \to & v_1 & (\mathsf{RPrompt}) \end{array}$$

RPrompt は、簡約する段階で prompt の中身が値になっており、prompt を外すことで簡約を進める。RControl の E は pure な評価文脈であり、Hole の中には、次に評価される項が入っている。RControl は control の c に control によって切り取られた継続を渡して e_1 を実行する。RControl を使う例を示すと以下のようになる。

$$\langle 1 + (\mathcal{F}c.\,c\,@\,1)\rangle \to \langle(\lambda c.\,c\,@\,1)\,@\,(\lambda x.\,1+x)\rangle$$

RControl は prompt の中身が、評価文脈の Hole に control が入ったコンテキスト、つまり「次に control を実行するという状態の項」である時に適用できる。例の簡約結果において、$(\lambda c.\,c\,@\,1)$ は control の中身である。$(\lambda x.\,1+x)$ はコンテキストの Hole を x に置き換えたものであり、control によって切り取られた継続になっている。$(\lambda c.\,c\,@\,1)$ に $(\lambda x.\,1+x)$ を適用することで c に切り取られた継続を渡して control の項を実行している。また、$E[x]$ の周りに prompt をつけて $\langle E[x]\rangle$ とすると shift の簡約規則になる。

4.2 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、つまり以下の定理を示す。

定理 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 κ 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。*schematic* というのは、「引数の変数の構造を変更しない」性質のことである。本稿で扱う継続 k は値と trail の二つを受け取るため、次の二式がこの性質を表す。

$$\begin{array}{rcl} (k\,\overline{@}\,x\,\overline{@}\,t)[v/x] & = & k\,\overline{@}\,v\,\overline{@}\,t \\ (k\,\overline{@}\,v\,\overline{@}\,x)[t/x] & = & k\,\overline{@}\,v\,\overline{@}\,t \end{array}$$

定理 2 の証明では、$e \to e'$ の簡約の形によって場合分けして取り組み、現段階では control 以外の場合において証明が済んでいる。証明の際は、必要な補題を都度立てる必要がある。例えば $e \to e'$ が $(\lambda x.\,e_1)\,\underline{@}\,v \to e_1[x/v]$ の場合の証明には以下の補題を使用した。

補題 3 (CPS 変換と代入の可換性) 任意の項 e_1、値 v、継続 k、*trail* t について $(\lambda x.\,e_1\,x)[v] \mapsto e_2$ が成り立つとき、$(\lambda x.\,[\![e_1\,x]\!]\,\overline{@}\,k\,\overline{@}\,t)[[\![v]\!]] \mapsto [\![e_2]\!]\,\overline{@}\,k\,\overline{@}\,t$ が成り立つ。

これは e_1 の中にある x を v に置き換えると e_2 になる時、e_1 の CPS 変化後の項にある x を v の CPS 変換後の値で置き換えると e_2 の CPS 変換結果になる、つまり代入と CPS 変換の可換性を示している。

補題 4 (継続に関する代入演算) 任意の項 e、値 v、*trail* t、*schematic* な継続 k について $(\lambda x.\,[\![e]\!]\,\overline{@}\,(k_1\,x)\,\overline{@}\,t)[v] \mapsto [\![e]\!]\,\overline{@}\,(k_1\,v)\,\overline{@}\,t$ が成り立つ。

継続 k の変数の構造が変わらない場合、k の中の x を v で置き換えられることを示している。

補題 5 (継続の簡約に関する補題) 任意の項 e_1、値 v、*trail* t、*schematic* な継続 k_1 について $(k_1\,\overline{@}\,[\![v]\!]\,\overline{@}\,t) \to (k_2\,\overline{@}\,[\![v]\!]\,\overline{@}\,t)$ が成り立つとき、$[\![e]\!]\,\overline{@}\,k_1\,\overline{@}\,t_2 \to [\![e]\!]\,\overline{@}\,k_2\,\overline{@}\,t_2$ が成り立つ。

継続の部分を同じ意味を持つものに置き換えたい時に使う。$e \to e'$ が $(\lambda x.\,e_1)\,\underline{@}\,v \to e_1[x/v]$ の場合の証明では最後の簡約で使用した。

$$\begin{array}{rl} & [\![(\lambda x.\,e_1)\,\underline{@}\,v]\!]\,k\,t \\ \vdots & \\ \to & [\![e']\!]\,\boxed{(\overline{\lambda}v.\,(\overline{\lambda}t'''.\,(\underline{\lambda}v.\,(\underline{\lambda}t''.\,k\,\overline{@}\,v\,\overline{@}\,t'')))\,\underline{@}\,v\,\underline{@}\,t''')}\,t \\ \to & [\![e']\!]\,\boxed{k}\,t \end{array}$$

5 まとめ

Agda を使って CPS インタプリタと CPS 変換を定式化することができた。CPS インタプリタと型規則を 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。また、今後は control の場合において正当性の証明に取り組む。

参考文献

[1] Kenichi Asai, Youyou Cong, and Chiaki Ishio. A functional abstraction of typed trails. In *PEPM*, 2021.

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

[3] 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.

[4] 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.

[5] Chiaki Ishio and Kenichi Asai. Verifying selective cps transformation for shift and reset. In William J. Bowman and Ronald Garcia, editors, *Trends in Functional Programming*, pp. 38–57, Cham, 2020. Springer International Publishing.

[6] Urara Yamada and Kenichi Asai. Certifying cps transformation of let-polymorphic calculus using phoas. In Sukyoung Ryu, editor, *Programming Languages and Systems*, pp. 375–393, Cham, 2018. Springer International Publishing.