shift/reset のための selective CPS 変換の正当性の証明

石尾 千晶, 浅井 健一

お茶の水女子大学

ishio.chiaki@is.ocha.ac.jp, asai@is.ocha.ac.jp

概 要　CPS 変換とは、限定継続命令 shift/reset を含まない関数型言語においても限定継続を扱えるよう、プログラムを継続渡し形式に変換するものである。CPS 変換ではすべての入力を変換の対象とするが、selective CPS 変換では変換が必要な部分に注釈をつけ、その部分のみ変換をおこなう。selective CPS 変換では、変換対象言語の項だけでなく、現在のコンテキストの型をあらわす answer type が変化するかどうかによっても場合分けが起こる。そのため、手で正当性の証明をするには複雑で、間違いの原因となりやすい。本稿では、単相の単純型付き λ 計算に shift/reset を加えた体系のための selective CPS 変換を定理証明系言語 Agda で定式化し、その正当性を証明した。定式化では、変数の束縛はメタ言語でおこなう PHOAS を採用した。また、正当性を証明する過程で、これまでの研究で示した補題のひとつに誤りを発見し、それを修正した。

1　はじめに

CPS 変換や shift/reset などの中心にある考え方は、継続と呼ばれている。継続とは、計算を実行しているある時点において、それ以降に残っている計算のことを意味する。例えば、$3+5*2$ という式の結果を知るためには、まず乗算の $5*2$ を計算する。このとき、乗算を実行中の継続は「($5*2$ を) 計算した結果を受け取って、そこに 3 を加える」というものになる。

プログラム中でこのような継続を扱うために、限定継続命令 shift/reset [5] を利用することができる。shift は $Sk.e$ と書いて「その時点の継続を切り取って変数 k に束縛し、その上で e を実行する」という意味になる。また、reset は $\langle e \rangle$ と書いて「切り取る継続の範囲を e に限定する」ことを意味する。先ほどの $3+5*2$ の例で $5*2$ を計算中の継続を利用するには、$\langle 3 + Sk.\ \ldots \rangle$ とする。これで、変数 k に $\lambda x.\langle 3+x \rangle$ という継続が入って、$\ldots$ の部分で継続を利用できるようになる。このような継続演算子を直接扱える関数型言語はそれほど多くなく、代わりに shift/reset を含んだプログラム全体を書き換える方法として CPS 変換が考えられる。

一般的な CPS 変換では入力プログラムのすべてを変換にかけるが、その変換結果が入力に比べてかなり長くなり、変換後のプログラムの実行速度にも影響を及ぼしていた。そこで、本当に変換を必要とする部分のみを CPS 変換し、残りは入力プログラムをそのまま残すことによって変換後のプログラムの効率を改善する、selective CPS 変換 [2] が考案された。この selective CPS 変換では、answer type と呼ばれる型に基づいて変換が必要かどうかの注釈を自動でつけている。

answer type とは、直近の reset に囲まれた項の型のことをあらわす。例えば、$\langle 3 + Sk.1 \rangle$ では $3 + Sk.1$ の型が answer type となる。この式の answer type は、式の $3+\ldots$ の形から int 型になる。$Sk.1$ の部分は、切り取ってきた継続 k は破棄して 1 を返すため、shift のふるまいを加味しても answer type は int 型のままである。一方で、$\langle 3 + Sk.\mathsf{true} \rangle$ を見てみると、shift 演算子は継続 $k = \lambda x.\, x + 3$ を破棄して true を返している。そのため、$3+\ldots$ から int 型とされていた answer type は、shift のふるまいを加味すると bool 型に変化することが静的に決定する。このように、shift を使うと answer type を変化させるような表現を書くことができる [1]。selective CPS 変換では、shift が含まれる項では切り取った継続を直接扱うため impure という注釈をつけ、それ以外には pure をつけて分類した上で、 impure な項だけを変換する。

selective CPS 変換の正当性を示そうとすると、注釈の付き方によって変換すべき場所が異なるために変換の定義そのものが複雑になるだけでなく、証明自体も注釈によって多くの場合分けが生じる。さらに、answer type についての証明も加えなければならず、手で正確な証明をおこなうのは次第に難しくなってくる。そのため、本研究では、定理証明系言語 Agda [12] を用いることにする。

本稿では、shift/reset を含む単相の単純型付き λ 計算のための selective CPS 変換の正当性を Agda で証明した様子を報告する。証明をおこなう前は、注釈や answer type によって場合分けが増えるので、それに対処することが目標であった。しかし、証明を進めるうちに、shift/reset や注釈を新たに導入するためには、通常の評価文脈や簡約規則の定義をより丁寧に確認する必要があることがわかったため、これ以降で詳しく述べる。さらに、この証明によって selective CPS 変換の正当性が示されただけでなく、これまでの研究 [2] で示した補題のひとつが正確ではなかったことがわかり、それを修正することができた。本研究を通して、ある程度、CPS 変換の証明の実装方法を確立できたのではないかと考えている。

これ以降の構成は、次のようになっている。2 節で変換の対象となる言語を定義し、3 節で selective CPS 変換を定義した上で、4 節で正当性の証明をおこなう。また、5 節で関連研究を、6 節に結論を述べる。

Agda による実装は、`http://pllab.is.ocha.ac.jp/~asai/jpapers/ppl/ppl19.agda` で公開している。

2 DS 項

この節では、selective CPS 変換前の言語を定義する。基本的な文法は 図 1 に示した通りである。このような言語は、shift/reset を使って継続を直接表現できることから direct style であるといい、本稿ではこれを略して DS 項と呼ぶことにする。

selective CPS 変換は、変換が必要な部分とそうでない部分を分類するために注釈を利用している。そのため、DS 項は右肩に添字の付いた、「注釈付きの項」として表現される。

2.1 DS 項の注釈

selective CPS 変換の注釈には i (impure) または p (pure) の 2 種類がある。impure の注釈が付く項は、継続を直接操作する可能性があるため CPS 変換がおこなわれる。それ以外の項や値を pure と分類する。基本的に、pure な DS 項のための CPS 変換は恒等関数の形になるが、部分式に impure な項があらわれるときのみその部分の CPS 変換をおこなう。例えば $(\lambda^{\mathsf{i}} x.\, (S^{a_2} k.\, x^{\mathsf{p}})^{\mathsf{i}})^{\mathsf{p}}$ のような項では、項全体としては関数抽象なので pure だが、body 部分が impure なので、その部分が CPS 変換される。CPS 変換について詳しくは 3.2 節 で述べる。

shift 項では、現在の継続を切り取って直接扱うので i という注釈を付けて、さらに shift 項を部分項にもつ項全体も i と定める。例えば、shift 項を含む関数適用の式 $\left((S^{a} k.\, e_1{}^{a_1})^{\mathsf{i}} \,@^{a_3}\, e_2{}^{a_2}\right)^{\mathsf{i}}$ では、この式を書いた段階で、shift 項とそれを部分項にもつ全体に i の注釈が付くことが決まる。なお、注釈 a, a_1, a_2, a_3 はすぐには決まらず、現時点では i と p のどちらも入りうることを意味する。

また、注釈のあいだには $\mathsf{p} < \mathsf{i}$ という関係が成り立ち、pure な項を impure であるとみなすことができる。つまり、必ず impure になる shift 項を pure とみなすことはできないが、本来なら CPS 変換を必要としない値 v^{p} などを、あえて impure とみなして CPS 変換をかけることはできる。

2.2 DS 項の定義と型付け規則

図 1 の型には、自然数型と矢印型が含まれる。矢印型は $\tau_2 \to \tau_1 @ \mathrm{cps}[\tau_3, \tau_4, a]$ と書いて、τ_2 型の値を受け取って τ_1 型の値を返す関数をあらわす。ただし、関数適用の前後の answer type と、それが変化するかどうかを示す注釈 a が含まれるという点において一般的な矢印型と異なっている。

$$
\begin{array}{lcll}
a & ::= & \mathsf{p} \mid \mathsf{i} & \text{注釈} \\
\tau & ::= & \mathsf{Nat} \mid \tau_2 \to \tau_1 @ \mathrm{cps}[\tau_3, \tau_4, a] & \text{型} \\
\Gamma & ::= & \cdot \mid \Gamma, x : \tau & \text{型環境} \\
v & ::= & n \mid x \mid \lambda^a x.\, A_1 & \text{値} \\
e & ::= & v \mid A_1 \mathbin{@^a} A_2 \mid S^a k.\, A_1 \mid \langle A_1 \rangle & \text{項} \\
A & ::= & e^a & \text{注釈付きの項}
\end{array}
$$

図 1. 注釈・型・DS 項の定義

関数適用前の answer type を τ_3、適用後の型を τ_4 として、これら 2 つの型が異なるとき注釈 a は i になる。逆に、a が p のときは τ_3 と τ_4 は等しい。

また、図 1 を見ると、DS 項の値の中に関数抽象 $\lambda^a x.\, A_1$ があるのがわかる。注釈付きの項 e^a を使って書き下すと、$\lambda^a x.\, {e_1}^{a_1}$ と同じことである。多くの場合、注釈 a は関数抽象の body 部分 ${e_1}^{a_1}$ の注釈 a_1 と等しい。しかし、関数の body 部分 ${e_1}^{a_1}$ が本来 ${e_1}^{\mathsf{p}}$ だが型推論のために ${e_1}^{\mathsf{i}}$ とみなしたいという場合があり、そのようなときは $a_1 = \mathsf{p}$ はそのままで $a = \mathsf{i}$ とする。こうすることで、関数抽象 $\lambda^{\mathsf{i}} x.\, {e_1}^{\mathsf{p}}$ は一見 impure に思えるが、body 部分は元の pure な項のまま保たれている。

DS 項の文法をひととおりながめたところで、図 2 の単相[1]の DS 項の型付け規則に進む。型判断は $\Gamma \vdash e^a : \tau_1 \, @\mathrm{cps}[\tau_2, \tau_3, a]$ と書いて、「型環境 Γ のもとで、DS 項 e^a は τ_1 型をもつ。また、e^a を実行すると answer type は τ_2 から τ_3 に変化する」と読む。また、型付け規則では 2 種類の制約記号を使う。1 つは注釈同士のあいだに成り立つ不等号の制約 $a_1 \le a_2$ で、$\mathsf{p} < \mathsf{i}$ を満たすように a_1 と a_2 が決まる。もう 1 つは answer type と注釈のあいだの制約 $\tau_1 \neq \tau_2 \Rightarrow a_1 = \mathsf{i}$ である。answer type の τ_1 と τ_2 が異なれば a_1 は必ず i になり、a_1 が p のとき τ_1 と τ_2 は必ず等しくなる。

図 2 の関数抽象の規則 (Fun) では、$(\lambda^{a_2} x.\, {e_1}^{a_1})^{\mathsf{p}}$ では、e_1 は項なので注釈 a_1 が impure か pure かすぐには定まらない。また、注釈 a_1 と a_2 のあいだには $a_1 \le a_2$ が成り立ち、a_2 はこの関数抽象を外側から見たときの注釈をあらわしている。

また、関数適用の規則 (App) では、$({e_1}^{a_1} \mathbin{@^{a_3}} {e_2}^{a_2})^a$ の注釈 a_3 は、実際に関数を適用することで継続が直接操作される可能性があるかどうかを示している。注釈 a_1, a_2, a_3 のあいだには制約はなく p でも i でもよいが、そのうち少なくとも 1 つが i になると、制約 $a_i \le a$ によって $a = \mathsf{i}$ になる。続けて、shift 項の規則 (Shift) を見る。$(S^{a_2} k.\, {e_1}^{a_1})^{\mathsf{i}}$ では、切り取ってきた継続を k に束縛して、その上で ${e_1}^{a_1}$ を実行している。注釈 a_2 は k の注釈をあらわしていて、継続は基本的に pure なのでほとんどの場合 $a_2 = \mathsf{p}$ になる。しかし、型推論のために、本来 pure な k を impure とみなしたいときには $a_2 = \mathsf{i}$ とする。この考え方は、上で述べた関数抽象の注釈の見方と同様のものとなっている。さらなる詳細や例については [2] を参照されたい。

ここまでに述べた DS 項の定義を実装するために、変数束縛の方法を考える。今回の実装では、Chlipala による PHOAS [4] を利用している。PHOAS を使った実装では、変数束縛の処理をメタ言語 (今回は Agda に相当する) のレベルでおこなうだけでなく、すべての DS 項がメタ言語レベルの変数をパラメータとして受け取るように定義する。これによって、煩雑な変数束縛の処理はメタ言語に任せることができる。DS 項のほとんどの定義では、問題なく PHOAS を使うことができている。それ以外のいくつかの場合については、どのように PHOAS を利用したかをそのつど説明していく。

[1]本稿では、let 多相は取り扱わない。let 多相を含む λ 計算のための一般的な CPS 変換の正当性の証明は [14] を参照されたい。

$$\boxed{\Gamma \vdash e^a : \tau_1 \,@\mathrm{cps}[\tau_2, \tau_3, a]} \quad \frac{x : \tau_1 \in \Gamma}{\Gamma \vdash x^{\mathsf{p}} : \tau_1 \,@\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]}\ (\textsc{Var}) \quad \frac{}{\Gamma \vdash n^{\mathsf{p}} : \mathsf{Nat} \,@\mathrm{cps}[\tau_1, \tau_1, \mathsf{p}]}\ (\textsc{Nat})$$

$$\frac{\Gamma, x : \tau_2 \vdash {e_1}^{a_1} : \tau_1 \,@\mathrm{cps}[\tau_3, \tau_4, a_1] \quad a_1 \leq a_2 \quad \tau_3 \neq \tau_4 \Rightarrow a_1 = \mathsf{i}}{\Gamma \vdash (\lambda^{a_2} x.\, {e_1}^{a_1})^{\mathsf{p}} : (\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_2]) \,@\mathrm{cps}[\tau_5, \tau_5, \mathsf{p}]}\ (\textsc{Fun})$$

$$\frac{\begin{array}{c} a_1 \leq a \quad a_2 \leq a \quad a_3 \leq a \quad \tau_5 \neq \tau_6 \Rightarrow a_1 = \mathsf{i} \quad \tau_4 \neq \tau_5 \Rightarrow a_2 = \mathsf{i} \quad \tau_3 \neq \tau_4 \Rightarrow a_3 = \mathsf{i} \\ \Gamma \vdash {e_1}^{a_1} : (\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_3]) \,@\mathrm{cps}[\tau_5, \tau_6, a_1] \quad \Gamma \vdash {e_2}^{a_2} : \tau_2 \,@\mathrm{cps}[\tau_4, \tau_5, a_2] \end{array}}{\Gamma \vdash ({e_1}^{a_1} \,@^{a_3}\, {e_2}^{a_2})^{a} : \tau_1 \,@\mathrm{cps}[\tau_3, \tau_6, a]}\ (\textsc{App})$$

$$\frac{\Gamma, k : (\tau_3 \to \tau_4 @\mathrm{cps}[\tau, \tau, a_2]) \vdash {e_1}^{a_1} : \tau_1 \,@\mathrm{cps}[\tau_1, \tau_2, a_1] \quad \tau_1 \neq \tau_2 \Rightarrow a_1 = \mathsf{i}}{\Gamma \vdash (S^{a_2} k.\, {e_1}^{a_1})^{\mathsf{i}} : \tau_3 \,@\mathrm{cps}[\tau_4, \tau_2, \mathsf{i}]}\ (\textsc{Shift})$$

$$\frac{\Gamma \vdash {e_1}^{a_1} : \tau_1 \,@\mathrm{cps}[\tau_1, \tau_2, a_1] \quad \tau_1 \neq \tau_2 \Rightarrow a_1 = \mathsf{i}}{\Gamma \vdash \langle {e_1}^{a_1} \rangle^{\mathsf{p}} : \tau_2 \,@\mathrm{cps}[\tau_3, \tau_3, \mathsf{p}]}\ (\textsc{Reset})$$

図 2. DS 項の型付け規則

$$\boxed{e^a[v^{\mathsf{p}}/y] = e'^a} \quad \frac{}{y^{\mathsf{p}}[v^{\mathsf{p}}/y] = v^{\mathsf{p}}}\ (\text{S-}\textsc{Var}{=}) \quad \frac{}{x^{\mathsf{p}}[v^{\mathsf{p}}/y] = x^{\mathsf{p}}}\ (\text{S-}\textsc{Var}{\neq}) \quad \frac{}{n^{\mathsf{p}}[v^{\mathsf{p}}/y] = n^{\mathsf{p}}}\ (\text{S-}\textsc{Nat})$$

$$\frac{\forall x.\, ({e_1}^{a_1}[v^{\mathsf{p}}/y] = {e'_1}^{a_1})}{(\lambda^{a_2} x.\, {e_1}^{a_1})^{\mathsf{p}}[v^{\mathsf{p}}/y] = (\lambda^{a_2} x.\, {e'_1}^{a_1})^{\mathsf{p}}}\ (\text{S-}\textsc{Fun}) \quad \frac{{e_1}^{a_1}[v^{\mathsf{p}}/y] = {e'_1}^{a_1} \quad {e_2}^{a_2}[v^{\mathsf{p}}/y] = {e'_2}^{a_2}}{({e_1}^{a_1} \,@^{a_3}\, {e_2}^{a_2})^{a}[v^{\mathsf{p}}/y] = ({e'_1}^{a_1} \,@^{a_3}\, {e'_2}^{a_2})^{a}}\ (\text{S-}\textsc{App})$$

$$\frac{\forall k.\, ({e_1}^{a_1}[v^{\mathsf{p}}/y] = {e'_1}^{a_1})}{(S^{a_2} k.\, {e_1}^{a_1})^{\mathsf{i}}[v^{\mathsf{p}}/y] = (S^{a_2} k.\, {e'_1}^{a_1})^{\mathsf{i}}}\ (\text{S-}\textsc{Shift}) \quad \frac{{e_1}^{a_1}[v^{\mathsf{p}}/y] = {e'_1}^{a_1}}{\langle {e_1}^{a_1} \rangle^{\mathsf{p}}[v^{\mathsf{p}}/y] = \langle {e'_1}^{a_1} \rangle^{\mathsf{p}}}\ (\text{S-}\textsc{Reset})$$

図 3. DS 項の代入規則

2.3 DS 項の代入規則

図 3 では DS 項の代入規則を示している。代入規則は $e^a[v^{\mathsf{p}}/y] = e'^a$ と書いて、「e^a の中に自由変数 y が含まれる可能性があり、y を 値 v^{p} で置換した結果は e'^a になる」と読む。代入規則の中で注釈 a が二度登場するが、これは代入をおこなっても項の注釈は変わらないことを示している。

変数のための代入規則 (S-Var=) では、変数 y^{p} が自由変数 y と一致するので、そのまま y^{p} を v^{p} に置き換えられる。(S-Var≠) では代入先の変数 x^{p} が変数 y と異なるため代入は起きず、結果として x^{p} がそのまま返ってくる。自然数の場合 (S-Nat) も (S-Var≠) と同様、代入は起きない。関数抽象のための代入規則 (S-Fun) では、PHOAS を利用して代入を定義する。関数抽象 $(\lambda^{a_2} x.\, {e_1}^{a_1})^{\mathsf{p}}$ の body 部分 ${e_1}^{a_1}$ には、変数 x が自由に出現する可能性がある。変数はメタ言語のレベルで束縛されているので、(S-Fun) の前提部では任意の変数 x について ${e_1}^{a}[v^{\mathsf{p}}/y] = {e'_1}^{a}$ が成り立つ必要がある。そのため、メタ言語の量化子を使って $\forall x.\ \ldots$ と表記されている。shift 項においても関数抽象と同様の方法で定義されている。

この代入規則を Agda で書こうとすると、リレーションまたは関数という 2 通りの実装の方法がある。代入規則だけを定義するなら、リレーション (Agda のデータ型) でも関数でも実装できる [4]。しかし、関数での実装をおこなうと、これ以降の証明を進めていく途中で追加の性質を示さなければならないが、今のところその性質の証明には失敗している。そのため、今回の実装ではリレーションを利用している。これまでの研究 [14] でも同様の理由で代入規則をリレーションで実装している。

$$
\begin{array}{lcll}
\sigma_{\mathsf{f}} & ::= & [\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]]_{\mathsf{f}}\ \tau_4@\mathrm{cps}[\tau_5,\tau_6,a_2] & \text{フレームの型} \\
F & ::= & ([\,]\ @^{a_3}\ e_2{}^{a_2})^a \mid (v_1{}^{\mathsf{p}}\ @^{a_3}\ [\,])^a \mid \langle[\,]\rangle^{\mathsf{p}} \quad (\text{任意の } i \text{ について } a_i \le a) & \text{フレーム} \\
F_p & ::= & ([\,]\ @^{a_3}\ e_2{}^{a_2})^a \mid (v_1{}^{\mathsf{p}}\ @^{a_3}\ [\,])^a \quad (\text{任意の } i \text{ について } a_i \le a) & \text{ピュアフレーム}
\end{array}
$$

$\boxed{\Gamma \vdash F : \sigma_{\mathsf{f}}}$ $\boxed{\Gamma \vdash F_p : \sigma_{\mathsf{f}}}$

$$
\frac{a_1 \le a \quad a_2 \le a \quad a_3 \le a \quad \tau_5 \ne \tau_6 \Rightarrow a_1 = \mathsf{i} \quad \tau_4 \ne \tau_5 \Rightarrow a_2 = \mathsf{i} \quad \tau_3 \ne \tau_4 \Rightarrow a_3 = \mathsf{i} \quad \Gamma \vdash e_2{}^{a_2} : \tau_2\ @\mathrm{cps}[\tau_4,\tau_5,a_2]}{\Gamma \vdash ([\,]\ @^{a_3}\ e_2{}^{a_2})^a : [(\tau_2 \to \tau_1@\mathrm{cps}[\tau_3,\tau_4,a_3])@\mathrm{cps}[\tau_5,\tau_6,a_1]]_{\mathsf{f}}\ \tau_1@\mathrm{cps}[\tau_3,\tau_6,a]}\ \begin{array}{l}(F\text{-}\textsc{App}_1) \\ (F_p\text{-}\textsc{App}_1)\end{array}
$$

$$
\frac{a_2 \le a \quad a_3 \le a \quad \tau_4 \ne \tau_5 \Rightarrow a_2 = \mathsf{i} \quad \tau_3 \ne \tau_4 \Rightarrow a_3 = \mathsf{i} \quad \Gamma \vdash v_1 : (\tau_2 \to \tau_1@\mathrm{cps}[\tau_3,\tau_4,a_3])\ @\mathrm{cps}[\tau_5,\tau_5,\mathsf{p}]}{\Gamma \vdash (v_1{}^{\mathsf{p}}\ @^{a_3}\ [\,])^a : [\tau_2@\mathrm{cps}[\tau_4,\tau_5,a_2]]_{\mathsf{f}}\ \tau_1@\mathrm{cps}[\tau_3,\tau_5,a]}\ \begin{array}{l}(F\text{-}\textsc{App}_2) \\ (F_p\text{-}\textsc{App}_2)\end{array}
$$

$$
\frac{\tau_1 \ne \tau_2 \Rightarrow a_1 = \mathsf{i}}{\Gamma \vdash \langle[\,]\rangle^{\mathsf{p}} : [\tau_1@\mathrm{cps}[\tau_1,\tau_2,a_1]]_{\mathsf{f}}\ \tau_2@\mathrm{cps}[\tau_3,\tau_3,\mathsf{p}]}\ (F\text{-}\textsc{Reset})
$$

図 4. DS 項のためのフレーム規則

2.4 DS 項のフレームとコンテキスト

2.5 節で DS 項のための簡約規則を定義するために、フレームとコンテキストを利用する。フレームは、項を評価する順序を示すために用いられ、そのフレームを何枚も重ねたものをコンテキストという。また、次に評価すべき場所を hole といい、[] と書く。hole の中に入った項が、次に評価されることになる。図 4 に、call-by-value, left-to-right のフレームとコンテキストの定義を示す。

フレームの型 σ_{f} は、hole の型とフレーム全体の型を組み合わせて書く。型 $\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]$ は hole の中に入る項の型、つまり次に評価される項の型をあらわす。また、その右側に書かれている型 $\tau_4@\mathrm{cps}[\tau_5,\tau_6,a_2]$ は、hole に項が入ったときのフレーム全体としての型をあらわす。

フレームには、関数適用と reset 項のためのフレーム F と、reset 項を除いたピュアフレーム[2] F_p の 2 種類がある。ピュアフレームを重ねたピュアコンテキストは、2.5 節の shift 項の簡約規則の中で、ある項を取り囲む直近の reset を明らかにするとき、その項には reset が含まれていないことを示すために利用される。

図 4 の中のフレーム規則の定義は、図 2 に示した DS 項の型付け規則から導くことができる。

規則 (F-$\textsc{App}_1$) と (F_p-$\textsc{App}_1$) は、関数適用の中の関数部分を評価するためのものである。そのため、図 2 の (App) 内の関数の型が $[\,]_{\mathsf{f}}$ の中に入ることになり、関数適用全体の型がフレーム全体の型になる。その他の制約などはすべて (App) から引き継いできたものである。また、規則 (F-$\textsc{App}_2$) と (F_p-$\textsc{App}_2$) も DS 項の関数適用のためのフレームの型付け規則だが、関数部分はすでに値 v_1 に評価されている。値の注釈は必ず pure になるので、規則 (App) での関数部分の注釈 a_1 は p に決定し、answer type の τ_5 と τ_6 は等しくなる。それに伴って、制約規則も一部削除されている。規則 (F-Reset) も同様に DS 項の型付け規則から導かれる。

次に、図 5 で、ピュアコンテキストを定義する。ピュアコンテキスト E_p は、ピュアフレームが 0 枚以上重なったものである。その型 σ_{c} では、ピュアコンテキスト内の hole の型 $\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]$ を $[\,]_{\mathsf{c}}$ の中に入れて、右側にピュアコンテキスト全体の型 $\tau_4@\mathrm{cps}[\tau_5,\tau_6,a_2]$ を書く。

規則 (E_p-Hole) のとき、つまりピュアコンテキストが空のとき、hole に入るべき型がピュアコンテキスト全体の型になる。また、(E_p-Frame) は新しいピュアフレーム F_p をピュアコンテキスト E_p の上にかぶせるためのものである。これまで E_p の中にあった hole は、そのまま新たなピュ

[2]本稿では、カタカナで表記された「ピュア」とは、「reset が含まれていない [9] 」ことを意味する。紛らわしい表現だが、継続が直接操作されるかどうかをあらわす impure や pure とは関係ない。

$$\begin{array}{lcll} \sigma_{\mathsf{c}} & ::= & [\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]]_{\mathsf{c}}\ \tau_4@\mathrm{cps}[\tau_5,\tau_6,a_2] & \text{ピュアコンテキストの型} \\ E_p & ::= & [\,] \mid F_p \circ E_p & \text{ピュアコンテキスト} \end{array}$$

$\boxed{\Gamma \vdash E_p : \sigma_{\mathsf{c}}}$

$$\frac{\tau_2 \neq \tau_3 \Rightarrow a = \mathsf{i}}{\Gamma \vdash [\,] : [\tau_1@\mathrm{cps}[\tau_2,\tau_3,a]]_{\mathsf{c}}\ \tau_1@\mathrm{cps}[\tau_2,\tau_3,a]}\ (E_p\text{-Hole})$$

$$\frac{\Gamma \vdash F_p : [\tau_4@\mathrm{cps}[\tau_5,\tau_3,a_2]]_{\mathsf{f}}\ \tau_6@\mathrm{cps}[\tau_7,\tau_3,a_3] \quad \Gamma \vdash E_p : [\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]]_{\mathsf{c}}\ \tau_4@\mathrm{cps}[\tau_5,\tau_3,a_2]}{\Gamma \vdash F_p \circ E_p : [\tau_1@\mathrm{cps}[\tau_2,\tau_3,a_1]]_{\mathsf{c}}\ \tau_6@\mathrm{cps}[\tau_7,\tau_3,a_3]}\ (E_p\text{-Frame})$$

図 5. DS 項のためのピュアコンテキスト

$\boxed{e^a \rightsquigarrow e'^{a'}}$

$$\frac{a_1 \leq a_3 \leq a \quad {e_1}^{a_1}[{v_2}^{\mathsf{p}}/y] = {e'_1}^{a_1}}{((\lambda^{a_3}x.\,{e_1}^{a_1})^{\mathsf{p}}\ @^{a_3}\ {v_2}^{\mathsf{p}})^a \rightsquigarrow {e'_1}^{a_1}}\ (\text{R-Beta}) \qquad \frac{{e_1}^{a_1} \rightsquigarrow {e'_1}^{a'_1}}{F[{e_1}^{a_1}] \rightsquigarrow F[{e'_1}^{a'_1}]}\ (\text{R-Frame})$$

$$\frac{a_1 \leq a_3 \leq a_4 \quad E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}}{\langle (E_p^{\mathsf{i}}[(S^{a_2}k.\,{e_1}^{a_1})^{\mathsf{i}}])\rangle^{\mathsf{i},\mathsf{p}} \rightsquigarrow \langle (((\lambda^{a_3}k.\,{e_1}^{a_1})^{\mathsf{p}}\ @^{a_3}\ (\lambda^{a_2}x.\,\langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{a_5}\rangle^{\mathsf{p}})^{\mathsf{p}})^{a_4}\rangle^{\mathsf{p}}}\ (\text{R-Shift})$$

$$\frac{}{\langle {v_1}^{\mathsf{p}}\rangle^{\mathsf{p}} \rightsquigarrow {v_1}^{\mathsf{p}}}\ (\text{R-Reset})$$

図 6. DS 項のための簡約規則

アコンテキスト全体の hole になる。評価文脈全体の型は、新しいピュアフレーム F_p の型になる。

この節で定義したフレーム・ピュアフレーム・ピュアコンテキストは [2] には含まれていなかったもので、定式化において項の実行順序を決めるために必要とされている。shift/reset を導入するためにはピュアなフレームやコンテキストが必要であることさえわかれば、これらのフレームなどの定義自体は基本的に図 2 から導けるため、実装することはそれほど難しくない。また、本稿では、ピュアではない「普通のコンテキスト」は定義していない。これは、2.5 節の簡約規則では、「普通のフレーム」のみ定義すれば十分だったからである。

2.5 DS 項の簡約規則

図 6 では、これまでに準備したフレームやピュアコンテキストを利用しながら、簡約規則を定義する。簡約規則の実装には、2.3 節と同様にリレーションを用いた。

(R-Beta) は、λ 計算の一般的な β_{v} 簡約の規則に、注釈や制約を付け加えたものである。${e_1}^{a_1}$ と ${e'_1}^{a_1}$ のあいだに代入の関係が成り立ち、さらに注釈が $a_1 \leq a_3 \leq a$ を満たすとき、規則に示したような簡約をおこなうことができる。この規則のうち、$((\lambda^{a_3}x.\,{e_1}^{a_1})^{\mathsf{p}}\ @^{a_3}\ {v_2}^{\mathsf{p}})^a$ にあらわれる注釈が $a_3 = a = \mathsf{i}$, $a_1 = \mathsf{p}$ の場合を取り上げて考える。このとき、関数部分 $(\lambda^{\mathsf{i}}x.\,{e_1}^{\mathsf{p}})^{\mathsf{p}}$ を外側から見たときの関数抽象の注釈は i だが、実際の body 部分は ${e_1}^{\mathsf{p}}$ で pure になっていることがわかる。これに値 ${v_2}^{\mathsf{p}}$ を適用すると、body 部分の注釈が簡約後の項の注釈になり、結果として ${e'_1}^{\mathsf{p}}$ が得られる。このような場合、(R-Beta) の簡約の前後で注釈が i から p に変化する。また、規則 (R-Frame) は、任意のフレーム中の項を簡約する。これを複数回使うことで、任意の評価文脈中の項を簡約できるようになる。さらに、(R-Reset) では、reset 項の body 部分がすでに値 v_1 に簡約されているとき、そのまま値 v_1 を返している。

(R-Shift) は、shift 項を簡約するための規則である。DS 項 $\langle E_p^{\mathsf{i}}[(S^{a_2}k.\,{e_1}^{a_1})^{\mathsf{i}}]\rangle^{\mathsf{p}}$ では reset 項の内側にあるピュアコンテキスト E_p の中に shift 演算子が出現している。この shift 演算子を実行するには、次のようにすればよい。(1) まず、現在の限定継続である E_p^{i} を切り取って、(2) 切り取ってきた限定継続を変数 k に束縛し、(3) さらにその限定継続を関数 $\lambda^{a_2}x.\,\langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{a_5}\rangle^{\mathsf{p}}$ の形に書き起こし、(4) その上で body 部分である ${e_1}^{a_1}$ を実行する。

この規則で特筆すべきは、簡約前後のピュアコンテキストの hole の注釈である。簡約前は E_p の hole には impure な項 $(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}$ が入っていたのに対して、簡約後は pure な項 x^{p} が入ることになっている。これを、いままでの定義を用いて実装しようとすると、hole の注釈が impure になるはずのピュアコンテキストに pure な項を入れることになり、型が合わないので (R-Shift) を実装することができなくなってしまう。そのため、ピュアコンテキストの hole にどの注釈の項が入るかを明確にして、同一のピュアコンテキストの hole に異なる注釈の項を入れられるようにしたい。つまり、異なる注釈の hole をもつ 2 つのピュアコンテキスト E_p^{i}, E_p^{p} を定義し、それらが同一のものであることを示すリレーション $E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}$ を定義する。そして (R-Shift) において、hole の注釈のみが異なる同一のピュアコンテキスト E_p^{i}, E_p^{p} を利用する。これらのピュアコンテキストは、簡約前は E_p^{i} の形だったが、簡約後は E_p^{p} の形になっている。この (R-Shift) にあらわれるピュアコンテキストについては、次の 2.6 節で詳しく述べる。

本稿の実装では PHOAS を利用しているので、わざわざ上で述べたようなリレーションを使わなくても、評価文脈そのものを高階関数で実装すればよいと思われるかもしれない。確かに、hole に入る項の注釈を受け取ったら、それに合うようなピュアコンテキストを返す関数を実装して、それを用いて shift 項のための簡約規則を作ることもできる。しかし、のちに shift 項のふるまいを規定する補題 (4 節の補題 5) をピュアコンテキストの構造による帰納法で証明しようとする際に問題が生じる。ピュアコンテキストを高階関数で定義してしまうと、ピュアコンテキストの構造によって場合分けすることができず、補題の証明が進まないという事態に陥る。そのため今回はリレーションによる実装を採用している。Biernacki と Polesiuk による実装 [3] でも、型付けされた評価文脈についての同様のリレーションを定義している。

また、簡約規則そのものの実装方法にも、リレーションと関数という 2 つの選択肢が考えられるが、この場合はリレーションを使って実装しなければならない。これは 4.5 節の正当性の証明で、DS 項の簡約方法によって場合分けをして証明する際にリレーションの形で定義していることが求められるからである。この簡約関係のうち、(R-Beta), (R-Frame), (R-Reset) を実装することは比較的容易で、定義をそのままプログラムにおこせばよい。しかし、(R-Shift) では、定義を決める段階だけでなく、のちに正当性を示す際にうまく証明が進むようなプログラムに落とし込む作業においても試行錯誤を必要とする。

2.6 impure/pure な hole をもつピュアフレームとピュアコンテキスト

2.5 節の簡約規則の (R-Shift) では、ピュアコンテキストの hole の注釈を明確にした上で、異なる注釈をもつピュアコンテキストのあいだのリレーションを用意しなければならないことがわかった。また、ピュアコンテキストの hole の注釈を考えるためには、同様にピュアフレームの hole の注釈も明示する必要がある。この節では、このような impure/pure な hole をもつピュアフレームとピュアコンテキストを図 7 に定式化する。

この 2.6 節では、2.4 節 の中でも特殊な状況を抜き出して説明していく。図 7 での定式化は、図 4, 5 内のいくつかの状況を明らかにするために書かれたものであり、実際に図 7 をそのまま実装したわけではない。Agda のプログラムで実装されているのは、基本的には、図 4, 5, 8 の 3 つである。

図 7 に、hole の注釈を明記したピュアフレームとピュアコンテキストを定義する。ピュアフレーム F_p やピュアコンテキスト E_p に添字 i (もしくは p) を付けたもの F_p^{i}, E_p^{i} は、その hole に impure (もしくは pure) な項が入らなければならないことを示している。もし hole が impure のとき、ピュアフレームやピュアコンテキスト全体も impure になる。一方 hole が pure のときは、全体が pure になるとは限らない。例えば F_p^{p} では、hole は pure だが、項 ${e_2}^{a_2}$ の注釈 a_2 によっては全体が impure になる可能性もある。そのため、E_p^{p} の定義内の $F_p \circ E_p^{\mathsf{p}}$ では E_p^{p} の hole が pure だとしても、新たにかぶせて追加するピュアフレーム F_p の hole には impure な項が入ってくる可能性があるということになる。

$$
\begin{array}{lcll}
F_p^{\mathsf{i}} & ::= & ([]^{\mathsf{i}}\ @^{a_3}\ e_2{}^{a_2})^{\mathsf{i}} \mid (v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{i}})^{\mathsf{i}} & \text{impure な hole をもつピュアフレーム} \\
F_p^{\mathsf{p}} & ::= & ([]^{\mathsf{p}}\ @^{a_3}\ e_2{}^{a_2})^{a} \mid (v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{p}})^{a} & \text{pure な hole をもつピュアフレーム} \\
 & & \text{(任意の } i \text{ について } a_i \le a) & \\
E_p^{\mathsf{i}} & ::= & []^{\mathsf{i}} \mid F_p^{\mathsf{i}} \circ E_p^{\mathsf{i}} & \text{impure な hole をもつピュアコンテキスト} \\
E_p^{\mathsf{p}} & ::= & []^{\mathsf{p}} \mid F_p \circ E_p^{\mathsf{p}} & \text{pure な hole をもつピュアコンテキスト}
\end{array}
$$

$\boxed{\Gamma \vdash F_p^{\mathsf{i}} : \sigma_{\mathsf{f}}}$

$$
\frac{\Gamma \vdash e_2{}^{a_2} : \tau_2\ @\mathrm{cps}[\tau_4, \tau_5, a_2] \quad \tau_4 \neq \tau_5 \Rightarrow a_2 = \mathsf{i} \quad \tau_3 \neq \tau_4 \Rightarrow a_3 = \mathsf{i}}{\Gamma \vdash ([]^{\mathsf{i}}\ @^{a_3}\ e_2{}^{a_2})^{\mathsf{i}} : [(\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_3]) @\mathrm{cps}[\tau_5, \tau_6, \mathsf{i}]]_{\mathsf{f}}\ \tau_1 @\mathrm{cps}[\tau_3, \tau_6, \mathsf{i}]}\ (F_p^{\mathsf{i}}\text{-}\textsc{App}_1)
$$

$$
\frac{\Gamma \vdash v_1 : (\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_3])\ @\mathrm{cps}[\tau_5, \tau_5, \mathsf{p}] \quad \tau_3 \neq \tau_4 \Rightarrow a_3 = \mathsf{i}}{\Gamma \vdash (v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{i}})^{\mathsf{i}} : [\tau_2 @\mathrm{cps}[\tau_4, \tau_5, \mathsf{i}]]_{\mathsf{f}}\ \tau_1 @\mathrm{cps}[\tau_3, \tau_5, \mathsf{i}]}\ (F_p^{\mathsf{i}}\text{-}\textsc{App}_2)
$$

$\boxed{\Gamma \vdash F_p^{\mathsf{p}} : \sigma_{\mathsf{f}}}$

$$
\frac{\Gamma \vdash e_2{}^{a_2} : \tau_2\ @\mathrm{cps}[\tau_4, \tau_5, a_2] \quad a_2 \le a \quad a_3 \le a \quad \tau_4 \neq \tau_5 \Rightarrow a_2 = \mathsf{i} \quad \tau_3 \neq \tau_4 \Rightarrow a_3 = \mathsf{i}}{\Gamma \vdash ([]^{\mathsf{p}}\ @^{a_3}\ e_2{}^{a_2})^{a} : [(\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_3]) @\mathrm{cps}[\tau_5, \tau_5, \mathsf{p}]]_{\mathsf{f}}\ \tau_1 @\mathrm{cps}[\tau_3, \tau_5, a]}\ (F_p^{\mathsf{p}}\text{-}\textsc{App}_1)
$$

$$
\frac{\Gamma \vdash v_1 : (\tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a_3])\ @\mathrm{cps}[\tau_4, \tau_4, \mathsf{p}] \quad a_3 \le a \quad \tau_3 \neq \tau_4 \Rightarrow a_3 = \mathsf{i}}{\Gamma \vdash (v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{p}})^{a} : [\tau_2 @\mathrm{cps}[\tau_4, \tau_4, \mathsf{p}]]_{\mathsf{f}}\ \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a]}\ (F_p^{\mathsf{p}}\text{-}\textsc{App}_2)
$$

$\boxed{\Gamma \vdash E_p^{\mathsf{i}} : \sigma_{\mathsf{c}}}$

$$
\frac{}{\Gamma \vdash []^{\mathsf{i}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_3, \mathsf{i}]]_{\mathsf{c}}\ \tau_1 @\mathrm{cps}[\tau_2, \tau_3, \mathsf{i}]}\ (E_p^{\mathsf{i}}\text{-}\textsc{Hole})
$$

$$
\frac{\Gamma \vdash F_p^{\mathsf{i}} : [\tau_4 @\mathrm{cps}[\tau_5, \tau_3, \mathsf{i}]]_{\mathsf{f}}\ \tau_6 @\mathrm{cps}[\tau_7, \tau_3, \mathsf{i}] \quad \Gamma \vdash E_p^{\mathsf{i}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_3, \mathsf{i}]]_{\mathsf{c}}\ \tau_4 @\mathrm{cps}[\tau_5, \tau_3, \mathsf{i}]}{\Gamma \vdash (F_p^{\mathsf{i}} \circ E_p^{\mathsf{i}})^{\mathsf{i}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_3, \mathsf{i}]]_{\mathsf{c}}\ \tau_6 @\mathrm{cps}[\tau_7, \tau_3, \mathsf{i}]}\ (E_p^{\mathsf{i}}\text{-}\textsc{Frame})
$$

$\boxed{\Gamma \vdash E_p^{\mathsf{p}} : \sigma_{\mathsf{c}}}$

$$
\frac{}{\Gamma \vdash []^{\mathsf{p}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]]_{\mathsf{c}}\ \tau_1 @\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]}\ (E_p^{\mathsf{p}}\text{-}\textsc{Hole})
$$

$$
\frac{\Gamma \vdash F_p : [\tau_4 @\mathrm{cps}[\tau_5, \tau_2, a_2]]_{\mathsf{f}}\ \tau_6 @\mathrm{cps}[\tau_7, \tau_2, a_3] \quad \Gamma \vdash E_p^{\mathsf{p}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]]_{\mathsf{c}}\ \tau_4 @\mathrm{cps}[\tau_5, \tau_2, a_2]}{\Gamma \vdash (F_p \circ E_p^{\mathsf{p}})^{\mathsf{p}} : [\tau_1 @\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]]_{\mathsf{c}}\ \tau_6 @\mathrm{cps}[\tau_7, \tau_2, a_3]}\ (E_p^{\mathsf{p}}\text{-}\textsc{Frame})
$$

図 7. impure/pure な hole をもつピュアフレームとピュアコンテキスト

$\boxed{F_p^{\mathsf{i}} \cong_{\mathsf{f}} F_p}$

$$
\frac{}{([]^{\mathsf{i}}\ @^{a_3}\ e_2{}^{a_2})^{\mathsf{i}} \cong_{\mathsf{f}} ([]^{\mathsf{p}}\ @^{a_3}\ e_2{}^{a_2})^{a}}\ (\cong_{\mathsf{f}}\text{-}\textsc{App}_1) \qquad \frac{}{(v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{i}})^{\mathsf{i}} \cong_{\mathsf{f}} (v_1{}^{\mathsf{p}}\ @^{a_3}\ []^{\mathsf{p}})^{a}}\ (\cong_{\mathsf{f}}\text{-}\textsc{App}_2)
$$

$\boxed{E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}}$

$$
\frac{}{[]^{\mathsf{i}} \cong_{\mathsf{c}} []^{\mathsf{p}}}\ (\cong_{\mathsf{c}}\text{-}\textsc{Hole}) \qquad \frac{F_p^{\mathsf{i}} \cong_{\mathsf{f}} F_p \quad E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}}{(F_p^{\mathsf{i}} \circ E_p^{\mathsf{i}})\,[]^{\mathsf{i}} \cong_{\mathsf{c}} (F_p \circ E_p^{\mathsf{p}})\,[]^{\mathsf{p}}}\ (\cong_{\mathsf{c}}\text{-}\textsc{Frame})
$$

図 8. impure/pure な hole をもつピュアフレーム・ピュアコンテキスト同士の関係

$$
\begin{array}{rcll}
\tau & := & \mathsf{Nat} \mid \tau_1 \to \tau_2 & \text{型} \\
v & := & n \mid x \mid \underline{\lambda} x.\, e & \text{値} \\
e & := & v \mid e_1 \mathbin{\underline{@}} e_2 \mid \underline{\mathsf{let}}\, x = e_1 \,\underline{\mathsf{in}}\, e_2 & \text{CPS 項}
\end{array}
$$

図 9. CPS 項

$$
\boxed{e[v/y] = e'} \quad \frac{}{y[v/y] = v}\ (\textsc{Scps-Var}{\neq}) \quad \frac{}{x[v/y] = x}\ (\textsc{Scps-Var}{=}) \quad \frac{}{n[v/y] = n}\ (\textsc{Scps-Nat})
$$

$$
\frac{\forall x.\, (e_1[v/y] = e_1')}{(\underline{\lambda} x.\, e_1)[v/y] = (\underline{\lambda} x.\, e_1')}\ (\textsc{Scps-Fun}) \quad \frac{e_1[v/y] = e_1' \qquad e_2[v/y] = e_2'}{(e_1 \mathbin{\underline{@}} e_2)[v/y] = (e_1' \mathbin{\underline{@}} e_2')}\ (\textsc{Scps-App})
$$

図 10. CPS 項の代入規則

図 7 では、続けて型付け規則を定めている。ここで示した規則は、どれも 図 4, 5 の型付け規則から導くことができる。例えば、規則 (F_p^{i}-$\textsc{App}_1$) では、図 4 の (F_p-$\textsc{App}_1$) の注釈 a_1 が i に固定されるので、それをピュアフレーム全体の注釈にも伝播させることで導出できる。つまり、$a_1 = \mathsf{i}$ が成り立つので、注釈 a_1 についての制約 $\tau_5 \neq \tau_6 \Rightarrow a_1 = \mathsf{i}$ が不要になる。また、$a_1 \leq a$ から $a = \mathsf{i}$ とわかるので、$a_i \leq a$ という形の制約は trivial に成り立つことがわかるため、省いてもよいことになる。さらに、規則 (F_p^{p}-$\textsc{App}_2$) について取り上げると、導出元の 図 4 の規則 (F_p-$\textsc{App}_2$) の注釈 a_2 が p に定まったことで answer type の τ_4, τ_5 が一致し、$\tau_4 \neq \tau_5 \Rightarrow a_2 = \mathsf{i}$ が不要になる。

図 8 では、注釈が異なる hole をもつような二枚の同じフレームの関係を定義している。$F_p^{\mathsf{i}} \cong_{\mathsf{f}} F_p$ と書くと、「二枚のフレーム F_p^{i} と F_p は等しいが、F_p^{i} のもつ hole は impure であるのに対して F_p のもつ hole の注釈は pure でも impure でもよい[3]」という意味になる。

ピュアフレームとピュアコンテキストの実装は基本的には 図 4, 5 の定義にしたがい、その中の hole に impure/pure の注釈をつけたものになっている。つまり、図 7 に相当する新たなデータ型を実装したわけではない。しかし、hole が impure/pure になる場合を定式化することにより、「2 つの同じフレーム (及びコンテキスト) があり、各々の hole の注釈のみが異なる」という関係をより理解しやすくなる。この関係は 2.5 節の (R-Shift) での問題をうまく処理するのに役立っている。

3 selective CPS 変換

この節では、selective CPS 変換の定義をおこなう。2 節で定義した DS 項は、この変換によって注釈や answer type を持たない標準的な λ 計算になる。変換後の項のことを CPS 項と呼ぶものとする。変換元の DS 項が限定継続の演算子を含んでいるだけでなく、CPS 変換も部分的に (selective に) おこなうため、CPS 項は完全な継続渡し形式にはならない。

3.1 CPS 項

図 9 では CPS 項の定義を示す。CPS 項は DS 項と異なり、右肩に注釈の添字がつかない形であらわされ、コンストラクタには下線が引かれている。また、selective CPS 変換の出力は次のような下線と上線を使って、2 レベルの言語として表現される。下線付きの項は CPS 項のコンストラクタであることを示し、上線付きの項は メタ言語 (Agda) のレベルで書かれていることを示す。メタ言語のレベルで書かれた項は 3.2 節での変換の最中にメタ言語によって実行される。このような変換は “one-pass” [6, 7] である、という。

CPS 項の型は自然数型もしくは矢印型であり、answer type や注釈のない形で表記される。値や項は、一般的な単純型付き λ 計算と同じである。項の定義には [2] にならって let 項が含まれてい

[3] $F_p^{\mathsf{i}} \cong_{\mathsf{f}} F_p$ の右辺にある F_p の hole には、任意の注釈の項が入ることができるため、F_p^{p} を定義する必要はない。

$$
\begin{array}{rcl}
[\![n^{\mathsf{p}}]\!]_{\mathsf{p}} & = & n \\
[\![x^{\mathsf{p}}]\!]_{\mathsf{p}} & = & x \\
[\![(\lambda^{\mathsf{p}} x.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}} & = & \underline{\lambda} x.\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(\lambda^{\mathsf{i}} x.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}} & = & \underline{\lambda} x.\, \underline{\lambda} k.\, k \mathbin{\underline{@}} [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(\lambda^{\mathsf{i}} x.\, e_1{}^{\mathsf{i}})^{\mathsf{p}}]\!]_{\mathsf{p}} & = & \underline{\lambda} x.\, \underline{\lambda} k.\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v.\, k \mathbin{\underline{@}} v) \\
[\![(e_1{}^{\mathsf{p}} \mathbin{@^{\mathsf{p}}} e_2{}^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}} & = & [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \mathbin{\underline{@}} [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![\langle e_1{}^{\mathsf{p}} \rangle^{\mathsf{p}}]\!]_{\mathsf{p}} & = & [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![\langle e_1{}^{\mathsf{i}} \rangle^{\mathsf{p}}]\!]_{\mathsf{p}} & = & [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v.\, v) \\
[\![(e_1{}^{\mathsf{p}} \mathbin{@^{\mathsf{p}}} e_2{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, (\underline{\lambda} v.\, k \mathbin{\overline{@}} v) \mathbin{\underline{@}} ([\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \mathbin{\underline{@}} [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}}) \\
[\![(e_1{}^{\mathsf{p}} \mathbin{@^{\mathsf{p}}} e_2{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, (\underline{\lambda} v_1.\, [\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_2.\, (\underline{\lambda} v.\, k \mathbin{\overline{@}} v) \mathbin{\underline{@}} (v_1 \mathbin{\underline{@}} v_2))) \mathbin{\underline{@}} [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(e_1{}^{\mathsf{i}} \mathbin{@^{\mathsf{p}}} e_2{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_1.\, (\underline{\lambda} v.\, k \mathbin{\overline{@}} v) \mathbin{\underline{@}} (v_1 \mathbin{\underline{@}} [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}})) \\
[\![(e_1{}^{\mathsf{i}} \mathbin{@^{\mathsf{p}}} e_2{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_1.\, [\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_2.\, (\underline{\lambda} v.\, k \mathbin{\overline{@}} v) \mathbin{\underline{@}} (v_1 \mathbin{\underline{@}} v_2))) \\
[\![(e_1{}^{\mathsf{p}} \mathbin{@^{\mathsf{i}}} e_2{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, ([\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \mathbin{\underline{@}} [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}}) \mathbin{\underline{@}} (\underline{\lambda} v.\, k \mathbin{\overline{@}} v) \\
[\![(e_1{}^{\mathsf{p}} \mathbin{@^{\mathsf{i}}} e_2{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, (\underline{\lambda} v_1.\, [\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_2.\, (v_1 \mathbin{\underline{@}} v_2) \mathbin{\underline{@}} (\underline{\lambda} v.\, k \mathbin{\overline{@}} v))) \mathbin{\underline{@}} [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(e_1{}^{\mathsf{i}} \mathbin{@^{\mathsf{i}}} e_2{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_1.\, (v_1 \mathbin{\underline{@}} [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}}) \mathbin{\underline{@}} (\underline{\lambda} v.\, k \mathbin{\overline{@}} v)) \\
[\![(e_1{}^{\mathsf{i}} \mathbin{@^{\mathsf{i}}} e_2{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_1.\, [\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v_2.\, (v_1 \mathbin{\underline{@}} v_2) \mathbin{\underline{@}} (\underline{\lambda} v.\, k \mathbin{\overline{@}} v))) \\
[\![(S^{\mathsf{p}} x.\, e_1{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, \underline{\mathsf{let}}\, x = \underline{\lambda} v.\, k \mathbin{\overline{@}} v \,\underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(S^{\mathsf{p}} x.\, e_1{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, \underline{\mathsf{let}}\, x = \underline{\lambda} v.\, k \mathbin{\overline{@}} v \,\underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v.\, v) \\
[\![(S^{\mathsf{i}} x.\, e_1{}^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, \underline{\mathsf{let}}\, x = \underline{\lambda} v.\, \underline{\lambda} k'.\, k' \mathbin{\underline{@}} (k \mathbin{\overline{@}} v) \,\underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} \\
[\![(S^{\mathsf{i}} x.\, e_1{}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}} & = & \overline{\lambda} k.\, \underline{\mathsf{let}}\, x = \underline{\lambda} v.\, \underline{\lambda} k'.\, k' \mathbin{\underline{@}} (k \mathbin{\overline{@}} v) \,\underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}} \mathbin{\overline{@}} (\overline{\lambda} v.\, v)
\end{array}
$$

図 11. DS 項から 2 レベルの CPS 項への selective CPS 変換 [2]

る。[2] では多相の let として shift 項の CPS 変換結果に使われているが、この論文では単相の let なので、関数適用の形で書くのと同じ意味である。

図 10 では、CPS 項のための代入規則を定義している。この規則の実装では、DS 項のときと同様に PHOAS を利用する。規則 (Scps-Fun) は、メタ言語のレベルで書かれた任意の変数 x について、関数抽象の body 部分の代入の関係が成り立つことを示している。

CPS 項の簡約規則は、簡約関係ではなく、等価関係を使って定義し、これを $\sim$ という記号であらわした。この等価関係は、標準的な単純型付き λ 計算の β_{v} 同値の定義に、反射律・対称律・推移律を加えたものになっている。さらに、上の規則のほかに β_Ω と β_{lift} [9] も利用している。β_Ω の規則は、まだ値まで評価が進んでいない項を、評価文脈の中に代入することができるというものである。これを式であらわすと、変数 x が、CPS 項の評価文脈 E に自由にあらわれないとき、$(\underline{\lambda} x.\, E[x]) \mathbin{\underline{@}} e \sim E[e]$ と書ける。また、β_{lift} は、外側の関数適用と let 項の順序を適切に入れ替えることで、let 項の in 以下を先に簡約するものである。これを式であらわすと、変数 x が値 v の中に自由にあらわれないとき、$(\underline{\mathsf{let}}\, x = e_1 \,\underline{\mathsf{in}}\, e_2) \mathbin{\underline{@}} v \sim \underline{\mathsf{let}}\, x = e_1 \,\underline{\mathsf{in}}\, (e_2 \mathbin{\underline{@}} v)$ のように書ける。

3.2 selective CPS 変換

図 11 に、shift/reset のための selective CPS 変換の定義を示す。この定義は、let 項の定義が単相であることを除けばこれまでの研究 [2] からそのまま持ってきたものとなっている。

この変換では、DS 項を受け取ると、2 レベルの CPS 項を返している。2 レベルの CPS 項は上線と下線付きで表記され、上線付きの項を static な項、下線付きの項を dynamic な項と呼ぶ [6]。このような 2 レベルの表現を導入することで、変換前の言語の簡約には対応していない余分な簡約基 (administrative β-redex) の生成を避けることができる。これは static な項として表現されるた

め、CPS 変換中にメタ言語 (Agda) によって実行され、変換後の項には残らない。

図 11 に示した定義には、pure な項のための変換 $[\![\,]\!]_{\mathsf{p}}$ と impure な項のための変換 $[\![\,]\!]_{\mathsf{i}}$ の 2 種類がある。ある項 e_1 が pure のとき限定継続の演算子は含まれていないので、改めて CPS に変換する必要はない。$[\![{e_1}^{\mathsf{p}}]\!]_{\mathsf{p}}$ は、基本的には受け取った項 ${e_1}^{\mathsf{p}}$ の注釈だけを取り除いて返すだけでよい。ただし、関数抽象や reset の中身が impure のときは、その部分のみを CPS 変換している。

一方、変換対象の項 e_1 が impure のとき、$[\![{e_1}^{\mathsf{i}}]\!]_{\mathsf{i}}$ の変換結果はどれも $\overline{\lambda}k.\ \ldots$ の形をしている。impure な DS 項を変換するためには、現在の継続 k を受け取ってくる必要があるからである。このような継続はメタ言語のレベルで計算されるため static な継続 (CPS 変換時に簡約される継続) と呼ばれる。

selective CPS 変換は一見複雑に思えるかもしれないが、関数適用の最後の変換規則 $[\![({e_1}^{\mathsf{i}}\,@^{\mathsf{i}}\,{e_2}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}}$ と shift 項の最後の規則 $[\![(S^{\mathsf{i}}x.\,{e_1}^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}}$ に着目すると、一般的な CPS 変換の規則によく似ていることがわかる。他の規則は、pure な項を DS 項の表記のまま残そうとして作られている。変換の定義の詳細は、[2] を参照されたい。

4　変換の正当性の証明

この節では、selective CPS 変換の正当性を証明する。まずはそのために必要な定義や補題から見ていく。

4.1　schematic

CPS 変換時にあらわれる static な継続 κ が schematic であるとは、κ が受け取ってくる項の構造を変更したり、破壊したりしないことを意味する。のちの正当性の証明 (定理 6) において、この schematic という性質が必要になる。

定義 1 (schematic)
変数 y は、static な継続 κ の中に自由に現れないものとする。任意の CPS 項の値 v について、$(\kappa\,\overline{@}\,y)[v/y] = \kappa\,\overline{@}\,v$ を満たすような継続 κ のことを schematic であるという。

例として、あえて schematic でない継続の例を取り上げる。ある継続 κ_0 が、引数として CPS 項の変数を受け取った時には自然数の 1 を返し、そうでないものを受け取ると 2 を返すようなとき、κ_0 は schematic ではない。これを式で表すと次のようになる。

$$(\kappa_0\,\overline{@}\,y)[v/y] = 1[v/y] = 1 \neq 2 = \kappa_0\,\overline{@}\,v$$

この例や、定義 1 にあらわれる式 $(\kappa_0\,\overline{@}\,y)[v/y]$ では、static な関数適用は代入の前に起こっていることに注意したい。例で κ_0 が schematic でなかったのは、引数の syntactic な構造を調べ上げて、その構造によって異なる値を返したことにより、引数の構造を保存できなかったからである。schematic という性質を定義することで、破壊的なふるまいをする継続をあらかじめ排除して証明を考えられるようになる。

4.2　代入規則に関連する補題

4.5 節での証明では、selective CPS 変換と代入演算が可換であるという性質が必要となる。それを示すために、まずは static な継続についての次の性質を定める。

定義 2 (継続への代入)
$v_1[v/y] = v_1'$ を満たす任意の v_1 と v_1'、static な継続 κ_1 と κ_2 について、$(\kappa_1\,\overline{@}\,v_1)[v/y] = \kappa_2\,\overline{@}\,v_1'$ が成り立つとき、$\kappa_1[v/y] = \kappa_2$ と書く。

この定義は、変数名が異なることを除けば、式の形は schematic に似ているように見える。定義 1 と定義 2 の基本的なアイディアは同じで、ある継続に対して、変数 y を関数適用してから値 v を代入した結果と、最初から値 v を代入した結果が等しくなるということを言っている。ただし、定義 2 では、継続が受け取る引数に対しても代入が起きている。

上の定義を使って、selective CPS 変換と代入演算が可換であるという補題を示す。selective CPS 変換には pure または impure な項のための変換があるので、補題もそれにしたがって場合分けが起きる。さらに、図 11 を見ると impure な変換の結果はいずれも $(\overline{\lambda}k.\ \ldots)$ の形で書かれており、実際には継続 κ_1 との関数適用の形 $([\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_1)$ で使われる。そのため、関数適用の関数部分全体に対して代入が起きる場合 (以下の (2) に相当) と、関数適用の引数部分のみで代入が起きる場合 (以下の (3) に相当) の 2 種類に場合分けされる。

補題 3 (変換と代入演算の可換性)
(1) $\Gamma, y:\tau \vdash e_1{}^{\mathsf{p}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_2,\mathsf{p}]$、$\Gamma \vdash e_2{}^{\mathsf{p}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_2,\mathsf{p}]$、$\Gamma \vdash v^{\mathsf{p}} : \tau\,@\mathrm{cps}[\alpha,\alpha,\mathsf{p}]$、かつ $e_1{}^{\mathsf{p}}[v^{\mathsf{p}}/y] = e_2{}^{\mathsf{p}}$ のとき、$[\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}[[\![v^{\mathsf{p}}]\!]_{\mathsf{p}}/y] = [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ が成り立つ。
(2) $\Gamma, y:\tau \vdash e_1{}^{\mathsf{i}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_3,\mathsf{i}]$、$\Gamma \vdash e_2{}^{\mathsf{i}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_3,\mathsf{i}]$、$\Gamma \vdash v^{\mathsf{p}} : \tau\,@\mathrm{cps}[\alpha,\alpha,\mathsf{p}]$、かつ $e_1{}^{\mathsf{i}}[v^{\mathsf{p}}/y] = e_2{}^{\mathsf{i}}$ であり、継続 κ_1, κ_2 が $\kappa_1[[\![v^{\mathsf{p}}]\!]_{\mathsf{p}}/y] = \kappa_2$ を満たすとき、$([\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_1)[[\![v^{\mathsf{p}}]\!]_{\mathsf{p}}/y] = ([\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_2)$ が成り立つ。
(3) $\Gamma \vdash e_1{}^{\mathsf{i}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_3,\mathsf{i}]$、$\kappa_1[v/y] = \kappa_2$ で、$e_1{}^{\mathsf{i}}$ の中に y が出現しないとき、$([\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_1)[v/y] = ([\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_2)$ が成り立つ。

補題 3 の (1) と (2) を証明するには、代入の式 $e_1{}^a[v^{\mathsf{p}}/y] = e_2{}^a$ についての構造的帰納法を利用し、さらに $e_1{}^a$ の構造にしたがって場合分けする。(1) では、変換 $[\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ のすべての場合を調べ、特に変数については代入が起きるときと起きないときの両方を考慮することで計 9 通りの場合分けになる。(2) では、$[\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}$ の全 12 通りに場合分けをする。また、補題 3 の (3) を示すには、$e_1{}^{\mathsf{i}}$ について構造的帰納法をかけ、変換 $[\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}$ の全 12 通りについて調べればよい。

4.3 継続の簡約補題

これまでの定義や補題に加えて、impure な変換が受け取る継続部分だけを簡約する補題も示す。

補題 4 (継続の簡約)
$\Gamma \vdash e_1{}^{\mathsf{i}} : \tau_1\,@\mathrm{cps}[\tau_2,\tau_3,\mathsf{i}]$ で、継続 κ_1, κ_2 が schematic であり、任意の v^{p} について $\kappa_1\,\overline{@}\,[\![v^{\mathsf{p}}]\!]_{\mathsf{p}} \sim \kappa_2\,\overline{@}\,[\![v^{\mathsf{p}}]\!]_{\mathsf{p}}$ のとき $[\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_1 \sim [\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa_2$ が成り立つ。

この補題の証明には、$e_1{}^{\mathsf{i}}$ の構造的帰納法を用いる。$e_1{}^{\mathsf{i}}$ が関数適用 (App) の形をしていた場合には、注釈 a_1, a_2, a_3 の全 8 通りのパターンを調べ上げる。また、$e_1{}^{\mathsf{i}}$ が shift 項 (Shift) の形をとる場合もあり、その場合も同様に注釈 a_1, a_2 の全 4 通りを調べることによって証明する。

4.4 shift 項の補題

この節では、shift 項のふるまいを規定する補題を示す。この補題では、ピュアコンテキストの中に shift 項が含まれていたとき、shift 項を外側に出して別々に評価をおこなう。

補題 5 (shift 項)
$\Gamma \vdash E_p^{\mathsf{i}} : [\tau_1@\mathrm{cps}[\tau_2,\tau_3,\mathsf{i}]]_{\mathsf{c}}\ \tau_4@\mathrm{cps}[\tau_5,\tau_3,\mathsf{i}]$、$\Gamma \vdash E_p^{\mathsf{p}} : [\tau_1@\mathrm{cps}[\tau_2,\tau_2,\mathsf{p}]]_{\mathsf{c}}\ \tau_4@\mathrm{cps}[\tau_5,\tau_2,a]$ かつ $E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}$ であるようなピュアコンテキスト E_p^{i} と E_p^{p} があり、また $\Gamma \vdash e_1 : \tau\,@\mathrm{cps}[\tau,\tau_3,a_1]$ を満たす e_1 と schematic な継続 κ に対して、次の式が成り立つ。

$$[\![(E_p^{\mathsf{i}}[S^{a_2}k.\,e_1{}^{a_1}])^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa \ \sim\ [\![((\lambda^{a_3}x.\,(E_p^{\mathsf{p}}[x])^a)^{\mathsf{p}}\,@^{a_3}\,(S^{a_2}k.\,e_1{}^{a_1})^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}}\,\overline{@}\,\kappa$$

この式の左辺では、shift 項はピュアコンテキスト E_p^{i} の中にあらわれている。しかし、E_p^{i} の具体的なピュアコンテキストの形は不明で、左辺の計算をこれ以上進めることはできない。そこで、右辺のように、ピュアコンテキストから shift 項を取り出すと、関数適用や shift 項のための selective CPS 変換の定義を使えるようになる。

この補題を示すには、ピュアコンテキスト間のリレーション $E_p^{\mathsf{i}} \cong_{\mathsf{c}} E_p^{\mathsf{p}}$ についての構造的帰納法を用いる。2.5 節でも述べたように、ピュアコンテキストをデータ型として定義し、さらに異なる注釈の hole をもつピュアコンテキストのリレーションを用意したことが、この補題の証明で生かされることとなる。

リレーション内のピュアコンテキストが両方とも hole だった場合 ($[]^{\mathsf{i}} \cong_{\mathsf{c}} []^{\mathsf{p}}$) は、前提の E_p^{p} の注釈 a は図 7 の (E_p^{p}-Hole) から $a = \mathsf{p}$ に決定する。さらに、結論の式にあらわれる 3 つの注釈 a_1, a_2, a_3 の全パターンを考えて、計 8 通りの場合分けとなる。一方で、リレーション内のピュアコンテキストのいずれかが少なくとも一枚以上のピュアフレームを含んでいるとき、一番上に新たにかぶせたピュアフレームについての場合分けをする。

一度 shift 項の補題の文言を定めたら、それを証明することはそれほど難しくない。ただし、似たような証明を何度も繰り返さねばならず、補題 5 を証明するコードは Agda で 4900 行程度になった。それよりも苦労したのは、shift 項のための適切な式を定める段階の作業である。例えば、補題の式の左辺にあらわれる項 $E_p^{\mathsf{i}}[S^{a_2}k.\, {e_1}^{a_1}]$ は、補題を利用する際は常に reset 演算子に囲まれている (図 6 の (R-Shift) の帰結部を参照)。しかし、reset 演算子で囲った形のままでは、補題の帰納法の仮定として利用するには弱すぎることがわかった。このような試行錯誤を繰り返した結果、現在の補題の式にいたっている。

これまでの研究 [2] では、shift 項の補題は次のような式だった。

$$[\![(E_p^{\mathsf{i}}[(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, k \;\sim\; [\![(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda}x.\, [\![E_p^{\mathsf{p}}[x^{\mathsf{p}}]]\!]_{\mathsf{i}}\, \overline{@}\, k)$$

この式を実装しようとしても、そもそも型が合わない。関数抽象 $(\overline{\lambda}x.\, [\![E_p^{\mathsf{p}}[x^{\mathsf{p}}]]\!]_{\mathsf{i}}\, \overline{@}\, k)$ は CPS 項の変数 x を受け取るが、ピュアコンテキスト $E_p^{\mathsf{p}}[x^{\mathsf{p}}]$ の中にある x は selective CPS 変換 $[\![]\!]_{\mathsf{i}}$ の引数なので DS 項の変数となっている。現在の補題 5 ではその問題は解決されていて、$(\lambda^{a_3}x.\, E_p^{\mathsf{p}}[x^{\mathsf{p}}]^a)$ のように関数抽象全体を DS 項で書いている。

また、以前の shift 項の補題と、現在の補題 5 のあいだに何かしらの関係がないか調べるために、補題 5 の右辺を $a = a_3 = \mathsf{i}$ として展開すると、次のようになる。

$$\begin{aligned}
& [\![((\lambda^{\mathsf{i}}x.\, (E_p^{\mathsf{p}}[x])^{\mathsf{i}})^{\mathsf{p}}\, @^{\mathsf{i}}\, (S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, \kappa \\
\equiv\;& (\underline{\lambda}v_1.\, [\![(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda}v_2.\, (v_1\, \underline{@}\, v_2)\, \underline{@}\, (\underline{\lambda}v.\, \kappa\, \overline{@}\, v)))\, \underline{@}\, [\![(\lambda^{\mathsf{i}}x.\, (E_p^{\mathsf{p}}[x])^{\mathsf{i}})^{\mathsf{p}}]\!]_{\mathsf{p}} \\
\sim\;& [\![(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda}v_2.\, ([\![(\lambda^{\mathsf{i}}x.\, (E_p^{\mathsf{p}}[x])^{\mathsf{i}})^{\mathsf{p}}]\!]_{\mathsf{p}}\, \underline{@}\, v_2)\, \underline{@}\, (\underline{\lambda}v.\, \kappa\, \overline{@}\, v)) \\
\equiv\;& [\![(S^{a_2}k.\, {e_1}^{a_1})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda}v_2.\, ((\underline{\lambda}x.\, \underline{\lambda}k.\, [\![(E_p^{\mathsf{p}}[x])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda}v.\, k\, \underline{@}\, v))\, \underline{@}\, v_2)\, \underline{@}\, (\underline{\lambda}v.\, \kappa\, \overline{@}\, v))
\end{aligned}$$

最後の式に着目すると、これに β_{v} 簡約をすれば以前のバージョンの式が得られるように思える。しかしこれを実装するには CPS 項の変数 v_2 を DS 項の変数 x に代入しなければならないが、そのためには逆方向の変換が必要になってしまう。また、上の式から、以前のバージョンの式は $a = a_3 = \mathsf{i}$ に近い場合にしか対応できていなかったこともわかる。

4.5 正当性の証明

これまでの定義や補題を利用して、正当性の証明に進む。示すべき内容は、DS 項を複数ステップ簡約できるとき、それを CPS 変換しても簡約の関係が保存されるというものである。DS 項の注釈によって ${e_1}^{\mathsf{p}} \rightsquigarrow {e_2}^{\mathsf{p}}$、${e_1}^{\mathsf{i}} \rightsquigarrow {e_2}^{\mathsf{p}}$、${e_1}^{\mathsf{i}} \rightsquigarrow {e_2}^{\mathsf{i}}$ の 3 つを示す必要がある。

定理 6 (正当性の証明)
(1) $\Gamma \vdash {e_1}^{\mathsf{p}} : \tau_1\, @\mathrm{cps}[\tau_2, \tau_2, \mathsf{p}]$ かつ ${e_1}^{\mathsf{p}} \rightsquigarrow {e_2}^{\mathsf{p}}$ のとき、$[\![{e_1}^{\mathsf{p}}]\!]_{\mathsf{p}} \sim [\![{e_2}^{\mathsf{p}}]\!]_{\mathsf{p}}$ が成り立つ。

$$
\begin{array}{lcll}
(1) & & [\![\langle (E_p^{\mathsf{i}}[(S^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{i}}])^{\mathsf{i}} \rangle^{\mathsf{p}}]\!]_{\mathsf{p}} & \\
(2) & \equiv & [\![(E_p^{\mathsf{i}}[(S^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{i}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v) & \\
(3) & \sim & [\![((\lambda^{\mathsf{i}} x.\, (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}})^{\mathsf{p}}\, @^{\mathsf{i}}\, (S^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{i}})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v) & \text{補題 5} \\
(4) & \equiv & (\underline{\lambda} v_1.\, \underline{\mathsf{let}}\, x = (\underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ((v_1\, \underline{@}\, x)\, \underline{@}\, (\underline{\lambda} v.\, v)))\, \underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}})\, \underline{@}\, (\underline{\lambda} x.\, \underline{\lambda} k.\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, k\, \underline{@}\, v))) & \\
(5) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, (((\underline{\lambda} x.\, \underline{\lambda} k.\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\underline{\lambda} v.\, k\, \underline{@}\, v)))\, \underline{@}\, x)\, \underline{@}\, (\underline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} & \beta_{\mathrm{v}} \\
(6) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ((\underline{\lambda} k.\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, k\, \underline{@}\, v)))\, \underline{@}\, (\underline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} & \text{補題 3 (2)},\ \beta_{\mathrm{v}} \\
(7) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, (\underline{\lambda} v.\, v)\, \underline{@}\, v))\, \underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} & \text{補題 3 (3)},\ \beta_{\mathrm{v}} \\
(8) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}} & \text{補題 4},\ \beta_{\mathrm{v}} \\
(9) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, ((\underline{\lambda} v.\, v)\, \underline{@}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}) & \beta_{\Omega} \\
(10) & \sim & \underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, ((\underline{\lambda} y.\, (y\, \underline{@}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}))\, \underline{@}\, (\underline{\lambda} v.\, v)) & \beta_{\mathrm{v}} \\
(11) & \sim & (\underline{\mathsf{let}}\, x = \underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v))\, \underline{\mathsf{in}}\, (\underline{\lambda} y.\, y\, \underline{@}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}))\, \underline{@}\, (\underline{\lambda} v.\, v) & \beta_{\mathit{lift}} \\
(12) & \equiv & ((\underline{\lambda} x.\, \underline{\lambda} y.\, y\, \underline{@}\, [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}})\, \underline{@}\, (\underline{\lambda} x.\, \underline{\lambda} k'.\, k'\, \underline{@}\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v))))\, \underline{@}\, (\underline{\lambda} v.\, v) & \\
(13) & \equiv & ([\![(\lambda^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}}\, \underline{@}\, [\![(\lambda^{\mathsf{i}} x.\, \langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}} \rangle^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}})\, \underline{@}\, (\underline{\lambda} v.\, v) & \\
(14) & \equiv & (\overline{\lambda} k.\, ([\![(\lambda^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}}\, \underline{@}\, [\![(\lambda^{\mathsf{i}} x.\, \langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}} \rangle^{\mathsf{p}})^{\mathsf{p}}]\!]_{\mathsf{p}})\, \underline{@}\, (\underline{\lambda} v.\, k\, \overline{@}\, v))\, \overline{@}\, (\overline{\lambda} v.\, v) & \\
(15) & \equiv & [\![((\lambda^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}\, @^{\mathsf{i}}\, (\lambda^{\mathsf{i}} x.\, \langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}} \rangle^{\mathsf{p}})^{\mathsf{p}})^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, v) & \\
(16) & \equiv & [\![\langle ((\lambda^{\mathsf{i}} k.\, e_1{}^{\mathsf{p}})^{\mathsf{p}}\, @^{\mathsf{i}}\, (\lambda^{\mathsf{i}} x.\, \langle (E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}} \rangle^{\mathsf{p}})^{\mathsf{p}})^{\mathsf{i}} \rangle^{\mathsf{p}}]\!]_{\mathsf{p}} &
\end{array}
$$

図 12. selective CPS 変換の正当性の証明 (一部抜粋)

(2) $\Gamma \vdash e_1{}^{\mathsf{i}} : \tau_1\, @\mathrm{cps}[\tau_2, \tau_2, \mathsf{i}]$ かつ $e_1{}^{\mathsf{i}} \rightsquigarrow e_2{}^{\mathsf{p}}$ のとき、$[\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, \kappa \sim (\underline{\lambda} v.\, \kappa\, \overline{@}\, v)\, \underline{@}\, [\![e_2{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ が成り立つ。
(3) $\Gamma \vdash e_1{}^{\mathsf{i}} : \tau_1\, @\mathrm{cps}[\tau_2, \tau_3, \mathsf{i}]$ かつ $e_1{}^{\mathsf{i}} \rightsquigarrow e_2{}^{\mathsf{i}}$ であり、さらに継続 κ が schematic のとき、$[\![e_1{}^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, \kappa \sim [\![e_2{}^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, \kappa$ が成り立つ。

これを証明するには、簡約関係 $e_1{}^{a_1} \rightsquigarrow e_2{}^{a_2}$ について、導出による帰納法をかける。最も単純な例は (2) で、元は impure だった項を簡約して pure になるような場合 $e_1{}^{\mathsf{i}} \rightsquigarrow e_2{}^{\mathsf{p}}$ は、簡約規則としてあてはまるのは (R-Beta) しかない。(R-Beta) にあらわれる注釈 a_3 について 2 通りの場合分けが生じる。

定理 6 (1) の証明が最も複雑な形をしている。この証明では、一般的な簡約規則の (R-Beta) と (R-Frame) に加えて、(R-Shift) と (R-Reset) を考えなくてはならない。その中でも、(R-Beta) と (R-Reset) は比較的簡単で、(R-Frame) もフレームの形によって場合分けが増えるが、やるべきことはそれほど難しくない。複雑なのは (R-Shift) で、この規則にあらわれる 5 つの注釈 a_1, a_2, a_3, a_4, a_5 について場合分けが起きる。これらの注釈の中でも、a_1, a_3, a_4 は $a_1 \le a_3 \le a_4$ を満たすので、具体的には次の 4 つの場合が考えられる。

$$(a_1,\, a_3,\, a_4) \in \{(\mathsf{p},\, \mathsf{p},\, \mathsf{p}),\, (\mathsf{p},\, \mathsf{p},\, \mathsf{i}),\, (\mathsf{p},\, \mathsf{i},\, \mathsf{i}),\, (\mathsf{i},\, \mathsf{i},\, \mathsf{i})\}.$$

それ以外の注釈 a_2, a_5 には制約が付いていないので、5 つの注釈すべてで 16 通りの場合分けが起きる。

図 12 には、上の 16 通りの場合分けのうち、$a_1 = \mathsf{p}$, $a_2 = a_3 = a_4 = a_5 = \mathsf{i}$ という複雑な例を示した。定理 6 の目標は、図 12 の式 (1) を変形して結論の式 (16) まで持っていくことである。最初に、式 (1) と (2) は等しいと書かれていて、式同士が $\equiv$ でつながれている。これは、メタ言語のレベルで処理が進んでいるという意味で、ここでは selective CPS 変換の定義にしたがって (2) に変形されたことを示す。その後、補題 5 を適用した結果を再びメタ言語のレベルで処理して式 (4) になっている。

次に、式 (5) から (8) にかけては、β_{v} 簡約をおこなっている。式 (4) では、変数 v_1 に、$(\underline{\lambda} x.\, \underline{\lambda} k.\, ([\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}\, \overline{@}\, (\overline{\lambda} v.\, k\, \underline{@}\, v)))$ を代入することで式 (5) が得られる。式 (5) では、impure な

変換 $[\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}$ の中身に変数 x^{p} を代入するので、補題 3.(2) を利用する。式 (6) では、impure な変換の継続部分でのみ代入が起こるので、補題 3.(3) を利用している。PHOAS を利用しているので、式 (6) の変数 k は $[\![(E_p^{\mathsf{p}}[x^{\mathsf{p}}])^{\mathsf{i}}]\!]_{\mathsf{i}}$ にはあらわれないようになっている。また、式 (7) では、変換の継続部分のみを β_{v} 簡約するために補題 4 を使い、式 (8) に到達している。

ここで一旦、結論の式 (16) からさかのぼって式変形を見ていくことにする。式 (16) で最も外側にあった reset 演算子を外して (15) になり、(15) で関数適用をおこない、(14) の static な簡約をおこなって、さらに (13) で変換の定義にしたがって式を展開すると、式 (12) に到達する。

式 (12) から式 (8) までは、逆向きの簡約をおこなっている。式 (8) は let 項の形をしているので、それに合わせて式 (12) の関数適用の形を単相の let 項に書き換えることで (11) になる。式 (11) を見ると、let 項の body 部分にあたる式 $(\underline{\lambda} y.\, y \mathbin{\underline{@}} [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}})$ は、いずれ $(\overline{\lambda} v.\, (\underline{\lambda} v.\, v) \mathbin{\underline{@}} v)$ を受け取ることがわかる。これを実行するために、β_{lift} を利用して、式 (11) の最も外側の関数適用と let 項の順序を入れ替える。これによって式 (10) が得られ、そこで β_{v} 簡約ができるようになる。最後に、式 (9) の $((\underline{\lambda} v.\, v) \mathbin{\underline{@}} [\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}})$ を簡約して、式 (8) になる。このとき、簡約は通常の β_{v} 簡約ではなく、β_Ω を用いる。関数適用の引数部分 $[\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ は CPS 項であり、値になるまで評価が進んでいない。そのため、β_{v} 簡約を利用しようとしても、引数部分が値になっていないので簡約規則を使うことができない。さらに、目指す式 (8) では $[\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ は評価せずにそのまま残っているので、できれば CPS 項の $[\![e_1{}^{\mathsf{p}}]\!]_{\mathsf{p}}$ をそのまま $(\underline{\lambda} v.\, v)$ に適用したい。そこで、β_Ω を使って、空のフレームをかぶせるように簡約をおこなった。

4.6 Agda での実装

4.5 節の証明をおこなう際、なるべく手で書いた証明に近い形で実装できるよう、equality reasoning を利用した。これは、Agda の標準ライブラリにある propositional equality の実装が元になっている。証明を書くユーザは 図 12 のような式を見ながら、次に適用できる規則を書く。この「適用できる規則」というのは、今回の実装では図 6 での簡約規則や 4 節の補題などがそれにあたり、ほとんどの場合、前の式の構造を見れば次に使うべき規則は一意に定まる。一旦規則を記入してしまえば、Agda の制約解消の機能を利用して次の式を自動で算出できる。この記法は、今回のように証明の量が膨大になるときに便利である。

今回の証明と equality reasoning の相性がよいことはわかったが、証明のための規則さえ定義できれば、その後、規則を入力して証明するのは比較的単純作業になってくる。そこで、さらに証明を自動化できないかと考えられる。現時点では、Agda はそれほど強力な証明探索の機能が備わっていないので、正当性の証明の実装全体で 7800 行程度の分量になっている。Coq などの証明探索の機能のある言語で同じことを実装すれば、より単純に証明を書くことができるだろうと考えている。

5 関連研究

selective CPS 変換の実装をおこなった論文に [3] が挙げられ、部分型と control effect の入った体系を論理関係を利用して定式化している。定式化の例として、shift_0 と reset_0 [10] のための selective CPS 変換の coherence を証明している。変換の定義そのものは本稿とは異なるものになっている。

一般的な CPS 変換の実装は、これまでにさまざまな形でおこなわれてきた。その中に、Isabelle/HOL による α 同値性を使った定式化 [11]、Coq による de Bruijn index を使った定式化 [8]、Twelf による HOAS を用いた定式化 [13] がある。

本稿では実装に Chlipala による PHOAS [4] を利用した。[4] では、単純型付き λ 計算と System F についての CPS 変換を Coq で実装し、実装中で代入規則をリレーションで定義している。

この Chlipala の証明方法を利用した例としては、単純型付き λ 計算に let 多相の入った体系のための、selective でない CPS 変換の正当性の証明 [14] がある。いずれは、本稿の selective CPS 変換を let 多相の体系にも導入したいと考えている。

6 まとめ

本稿は、単純型付き λ 計算と限定継続演算子 shift/reset のための selective CPS 変換の正当性を Agda で証明した。実装では、変数束縛の表現に PHOAS を利用し、評価文脈を使って shift/reset のふるまいを定式化した。また、これまでの研究における定義を確認しながら実装を進める中で、変換の正当性が示されただけでなく、補題のひとつを修正することもできた。

今後は let 多相の体系に selective CPS 変換を組み合わせて証明をおこないたいと考えている。

謝辞

有益なコメントを下さった査読者の皆様に感謝申し上げます。また、本研究は一部 JSPS 科研費 18H03218 の助成を受けたものです。

参考文献

[1] K. Asai and O. Kiselyov. Introduction to programming with shift and reset. *ACM SIGPLAN Continuation Workshop 2011*, 2011.

[2] K. Asai and C. Uehara. Selective CPS Transformation for Shift and Reset. *Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM'18)*, pp. 40–52, 2018.

[3] D. Biernacki and P. Polesiuk. Logical relations for coherence of effect subtyping. *Logical Methods in Computer Science*, Vol. 14, No. 1, 2018.

[4] A. Chlipala. Parametric Higher-Order Abstract Syntax for Mechanized Semantics. *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, pp. 143–156, September 2008.

[5] O. Danvy and A. Filinski. Abstracting Control. *Proceedings of the ACM conference on LISP and functional programming (LFP'90)*, pp. 151–160, 1990.

[6] 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.

[7] O. Danvy, K. Millikin, and L. R. Nielsen. On one-pass cps transformations. *Journal of Functional Programming*, Vol. 17, No. 6, pp. 793–812, 2007.

[8] Z. Dargaye and X. Leroy. Mechanized verification of CPS transformations. *Proceedings of the International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'05)*, pp. 211–225, 2007.

[9] Y. Kameyama and M. Hasegawa. A sound and complete axiomatization of delimited continuations. *Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, pp. 177–188, 2003.

[10] M. Materzok and D. Biernacki. Subtyping Delimited Continuations. *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP'11)*, pp. 81–93, September 2011.

[11] Y. Minamide and K. Okuma. Verifying CPS transformations in Isabelle/HOL. *Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding (MERLIN'03)*, pp. 1–8, 2003.

[12] U. Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

[13] Y. H. Tian. Mechanically Verifying Correctness of CPS Compilation. *Proceeding of the Twelfth Computing: The Australasian Theory Symposium (CATS'06)*, Vol. 51, pp. 41–51, 2006.

[14] U. Yamada and K. Asai. Certifying CPS Transformation of Let-polymorphic Calculus Using PHOAS. *Proceedings of the 16th Asian Symposium on Programming languages and systems (APLAS'18)*, 2018. 375–393.