shift/reset のための Selective CPS 変換の定式化とその自動証明

理学専攻　情報科学コース　1840647　石尾 千晶　(指導教員：浅井 健一)

1　はじめに

継続とは、プログラム中の残りの計算を表すものである。一般的な継続はプログラムの実行が終わるまでの全ての計算を指すのに対し、限定継続はプログラムの実行中のある時点までの残りの計算を表す。プログラム中で限定継続を扱うには、shift/reset [5] などの限定継続演算子を使うが、これらはそれほど多くのプログラミング言語でサポートされているわけではない。

そこで、shift/reset を含むプログラム全体に CPS (Continuation-Passing Style) 変換を施すことで、shift/reset を含まないが同じ挙動のプログラムに書き換えることが考えられる。一般に CPS 変換の出力結果は入力に比べてかなり長くなり、変換後のプログラムの性能にも影響を及ぼしていた。そこで、入力プログラムに注釈をつけて、入力の一部を CPS 変換する Selective CPS 変換 [1] が提案された。Selective CPS 変換は一般的な CPS 変換に比べて出力結果の性能が向上することが知られている。しかし、Selective CPS 変換では、注釈によって変換を施す場所が変わるために変換規則そのものが複雑になる上、正当性の証明そのものにも多くの場合分けが生じる。

本研究では、単純型付き λ 計算に shift/reset を含めた体系に対する Selective CPS 変換の正当性を、まずは定理証明系言語の Agda を使って手動で証明する。その結果、これまでの研究 [1] で示した補題の一部に誤りを発見し、それを修正した [6]。この証明の実装は約 7800 行となったが、多くの場合分けによって繰り返し似たような証明が現れており、自動化の余地があると考えられる。定理証明系言語では自動証明の取り組みが進んでいて、例えば Coq では Ltac [3] などを用いてユーザが各自の証明に特化した tactic を書くことができる。Agda では自動証明機能 [7] は開発されているものの、これを大規模な証明に使用することは難しい。そこで、最近新しく elaboration [4] の機能が追加された Agda の Reflection API [8] を用いて証明に特化した tactic を書き、証明の自動化を目指す。

2　CPS 変換の定式化

本節では、Selective CPS 変換の定式化をおこなう前に、まずはその基礎となる単純型付き λ 計算に対する CPS 変換 [5] を紹介する。また、これを Agda で定式化し、その正当性を証明する。

定理証明系言語を用いて λ 計算を定式化する際、変数束縛の実装方法は自明でない問題である。本稿では、メタ言語の変数束縛機能を利用する、PHOAS (Parameterized Higher-Order Abstract Syntax) [2] を採用した。PHOAS を利用することで、人が手で書く証明と同程度の複雑さで証明を書くことができている。

Agda で定式化をおこなった結果、Danvy と Filinski による証明 [5] を再現できている。

3　Selective CPS 変換の定式化

本節では、2 節の体系を拡張した Selective CPS 変換 [1] を Agda で定式化し、その正当性を証明する。

3.1　DS 項と CPS 項

Selective CPS 変換前の言語を DS 項と呼ぶ。これは単相の単純型付き λ 計算に shift/reset を加えたものである。以下に DS 項の定義を示す。

a	::=	$\mathsf{p} \mid \mathsf{i}$	注釈
τ	::=	$\mathsf{Nat} \mid \tau_2 \to \tau_1 @\mathrm{cps}[\tau_3, \tau_4, a]$	型
v	::=	$n \mid x \mid \lambda^a x.\, {e_1}^{a_1}$	値
e	::=	$v^{\mathsf{p}} \mid {e_1}^{a_1}\ @^{a_3}\ {e_2}^{a_2} \mid S^a c.\, {e_1}^{a_1} \mid \langle {e_1}^{a_1} \rangle$	項

DS 項の各項には p (pure) または i (impure) の注釈が付いている。継続を直接操作する可能性がある項には impure の注釈を付けて、CPS 変換をおこなう。それ以外の項を pure な項と定め、基本的に pure な項の変換は恒等関数の形になる。DS 項のうち、$S^a c.\, {e_1}^{a_1}$ は shift 演算子を表し、「その時点の継続を切り取って変数 c に束縛し、その上で ${e_1}^{a_1}$ を実行する」という意味になる。また $\langle {e_1}^{a_1} \rangle$ は reset 演算子を表し、「切り取る継続の範囲を ${e_1}^{a_1}$ に限定する」という意味になる。

また、Selective CPS 変換後の言語を CPS 項と呼ぶ。これは単相の単純型付き λ 計算になり、変換後の項なので注釈は付いていない。さらに、DS 項と CPS 項のそれぞれに対して代入規則・項の評価のためのフレームとコンテキスト・簡約規則をリレーションによって定義する。

3.2　Selective CPS 変換

Selective CPS 変換には、impure な項と pure な項のための 2 種類の変換があり、部分項の注釈の付き方によって多くの場合分けが生じる。例えば、関数適用 $({e_1}^{a_1}\ @^{a_3}\ {e_2}^{a_2})^a$ のための変換は 9 通りある。その内訳は、関数適用全体が pure ($a = \mathsf{p}$) で $a_1 = a_2 = a_3 = \mathsf{p}$ となる場合が 1 通りと、全体が impure ($a = \mathsf{i}$) で a_1, a_2, a_3 が p または i のいずれかになる場合が 8 通りである。これらの規則は一見複雑だが、$a_1 = a_2 = a_3 = \mathsf{i}$ の場合は、一般的な CPS 変換の関数適用の規則とほぼ一致している。残りの規則は一般的な CPS 変換の規則を単純にしたもので、pure な項をなるべく DS 項のまま残すように変換している。

3.3　正当性の証明

Selective CPS 変換が正しいことの証明をおこなう。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 1 (Selective CPS 変換の正当性)
(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。

(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。このうち最も複雑なのは定理 1 (1) の `shift` の簡約規則の場合で、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。

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、Agda での実装は全体で約 7800 行となった。この方法は今回のように証明の量が膨大になるときに便利で、証明のための規則や補題さえ定義できれば、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、この証明の単純作業の部分をさらに自動化できないかと考えられる。

4 Reflection API による自動証明

本節では、2 節の証明全体と、3 節の証明の一部を自動化する。自動化には、最近新しくなった Agda の Reflection API [8] を利用する。Reflection API を使えば、Agda のコードと、コードの内部表現の構文木を行き来できる。さらに、Agda の型チェック機能が「型チェックモナド」として API に新たに追加されたことで、コードの型情報の読取や、型チェックに通ったコードの生成も可能になっている。

Reflection API は現在も開発中の機能のため、これを使ってモナディックに Agda を書く方法はまだあまり知られていない。そこで、API を解説し、2 節の証明を例にとって自動化の方法を説明する。また、その方法を応用して 3 節の証明の一部を自動化する。

4.1 手動の作業を自動化する

Agda を使って証明を書く際、証明の中の未完成部分は hole と呼ばれ、そこでユーザがインタラクティブに証明を書くことができる。ユーザは hole の型と hole で使用可能な変数一覧 (コンテキスト) を調べ、予想される答えを hole に書いて、型チェックに通ることを確認する。型チェックに成功すると hole は消えて、ユーザが書いた答えと、必要に応じて引数が新しい hole として表示される。

この一連の作業を Reflection API で置き換えるには、解きたい証明部分 (これを goal と呼ぶことにする) を受け取ったら、それを解いて新たな複数の goal を返すような関数を作ればよい。そのような関数を再帰的に呼び出すことで、証明を完成させていく。

Reflection API を使った簡単な証明の例を見てから、2 節でおこなった証明に特化した tactic を書いて、証明を自動化していく。2 節では、代入規則や簡約規則はデータ型で定義されていたため、指定したコンストラクタを適用できるような関数を作成する。また、Coq での `intro` コマンドのように、ラムダ抽象を導入したい場合には、goal のコンテキストを拡張することによって対応する。さらに、Reflection API では再帰を書く方法は提供されていないので、induction principle を手動で定義し、それを適用する際の作業の一部を自動化する。

4.2 Selective CPS 変換の自動証明の結果と展望

3 節の正当性の証明は、2 節と同様、簡約規則の導出による帰納法をかけることによって証明できる。そこで、4.1 節で紹介した方法を適用することで、正当性の証明のコード 約 2000 行のうち、約 1300 行を生成できている。

ただし、Selective CPS 変換の正当性の自動証明ではいくつかの問題があらわれる。まず、定理 1 の文言 (1) と (3) の証明では相互再帰が起きるため、より複雑な induction principle が必要となる。また、3 節では 2 節よりも簡約規則の種類が増えるため、goal の構造を調べるだけでは答えが一意に定まらない場合がある。現時点では、このような場合にはユーザが判断せざるをえないが、将来的には、非決定性モナドを導入して、自動的に複数候補を試せるようにすることになるだろう。

5 まとめ

CPS 変換と Selective CPS 変換を Agda で PHOAS を用いて定式化し、その正当性を証明した。さらに、Agda の Reflection API を用いてこれらの証明の自動化をおこない、Selective CPS 変換の正当性の証明の一部を自動化できた。

参考文献

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

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

[3] A. Chlipala. *Certified Programming with Dependent Types*. MIT Press, 2013.

[4] D. Christiansen and E. Brady. Elaborator Reflection: Extending Idris in Idris. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, pp. 284–297. ACM, 2016.

[5] 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.

[6] C. Ishio and K. Asai. Verifying Selective CPS Transformation for Shift and Reset. In *Proceedings of the symposium on Trends in Functional Programming 2019 (TFP'19)*, 21 pages. Post proceeding version to appear in EPTCS.

[7] F. Lindblad and M. Benke. A Tool for Automated Theorem Proving in Agda. In *Types for Proofs and Programs*, pp. 154–169. Springer Berlin Heidelberg, 2006.

[8] Agda Development Team. Reflection – Agda 2.6.0.1 documentation. `https://agda.readthedocs.io/en/v2.6.0.1/language/reflection.html`, 2019. [Online; accessed 29-Dec-2019].