限定継続命令 shift/reset の Selective CPS 変換

理学専攻 情報科学コース　上原 千裕 (指導教員：浅井 健一)

1　はじめに

継続とは, 計算中のある時点における残りの計算を表す. 継続の中でも限定継続は, 一定範囲内のみの残りの計算を表す. shift/reset [2] は限定継続を扱うことができるオペレータで, shift はその時点での継続を切り取ってくる命令, reset は切り取られる継続の範囲を限定する命令である. 限定継続命令を使うと, ユーザがプログラムの実行順序を制御できるため, 例外処理や非決定プログラミングなど色々な場面で使われる.

これまで, 限定継続命令をサポートするには, システムに手を加えて直接実装する方法が多くとられてきた. 一方プログラム全体を CPS 変換すれば, 限定継続命令を使うことができるようになるが, 多くの場合効率が落ちる. しかし, shift/reset に関係した部分のみを selective に CPS 変換 [4] すれば, 効率を落とすことなく限定継続命令をサポートできる可能性がある.

本研究では, 関数型言語を対象にした, selective に CPS 変換を行うシステムを提案する. 最初に「CPS 変換を必要とするか」という情報を型システムの制約の形で表現する. 制約解消は通常「最も良い解」を得られるように行うが, ここで扱う制約の解には「最も良い解」が必ずしも存在するとは限らない. 本研究では, 過度に型推論と制約解消を複雑にすることなく, ある程度の解を得られる制約解消法を提案する. 解が得られたら, CPS 変換が必要な場所が特定されるので, その情報に従って selective に CPS 変換を行う. また簡単なプログラムで実際に selective CPS 変換を行い, selective に CPS 変換することで効率が向上することを確認した.

2　対象言語, 注釈, 型規則

本稿で扱うのは, let 多相の入った λ 計算に整数, 真偽値と if 文, 不動点演算子 fix, そして限定継続命令 shift/reset が加わった言語であり, 定義は以下である.

$$
\begin{array}{rcll}
\alpha & ::= & \epsilon \mid i & \text{annotation} \\
t & ::= & \tau \mid \texttt{bool} \mid \texttt{int} \mid t_1 \to t_2 \,@\mathrm{cps}[t_3, t_4, \alpha] & \\
 & & & \text{monomorphic type} \\
T & ::= & t \mid \forall \tau.T & \text{polymorphic type} \\
v & ::= & n \mid x \mid \texttt{true} \mid \texttt{false} \mid \lambda^{\alpha} x.A \mid \texttt{fix}^{\alpha}\ f.x.A & \text{value} \\
e & ::= & v \mid A_1 \,@^{\alpha}\, A_2 \mid Sk.A \mid \langle A \rangle \mid \texttt{let}\, x = v^{\,\epsilon}\, \texttt{in}\, A & \\
 & & \mid \texttt{if}\ A_1\ \texttt{then}\ A_2\ \texttt{else}\ A_3 & \text{expression} \\
A & ::= & e^{\,\alpha} & \text{expression with annotation}
\end{array}
$$

2.1　対象言語と注釈

α は「CPS 変換が必要かどうか」を示す注釈である. また, $A = e^{\alpha}$ は式 e に注釈 α が付いた式を示す. 注釈 ϵ は CPS 変換が不要であることを示す. CPS 変換が不要な場合, つまり shift によるコントロールエフェクトが起きない場合は, その式は pure である, という表現を使う. 一方, 注釈 i は CPS が必要であることを示す. そのような式は impure であるという. これはその式の実行中に shift が起こる可能性があることを示す.

$Sk.A$ は shift 命令である. これで現在の継続を切り取って k に入れた上で A を行うことを示す. また $\langle A \rangle$ は reset 命令である. これで, A の中で起こる shift で切り取られる継続を A までに限定する. 例えば,

$$\langle (Sk.\, k \,@\, 2) - 1 \rangle$$

は, $Sk.\, k \,@\, 2$ を実行する時, 計算結果から 1 を引くことが継続である. この式では, $\lambda x.\langle x - 1 \rangle$ を k に入れて,

$$\langle (\lambda x.\langle x - 1 \rangle) \,@\, 2 \rangle$$

という式になる. 後は関数適用が実行されて, 1 が返る. また `string_of_int` を, `int` を `string` に変える関数であるとして,

$$\langle (Sk.\, \texttt{string\_of\_int}\ (k \,@\, 2)) - 1 \rangle$$

という式を考える. この式は実行すると

$$\langle \texttt{string\_of\_int}\ ((\lambda x.\langle x - 1 \rangle) \,@\, 2) \rangle$$

となるため, $(\lambda x.\langle x - 1 \rangle) \,@\, 2$ の結果である `int` の 1 が `string` になって返る. この時, reset の中身が返す型を考える.

$$(Sk.\, \texttt{string\_of\_int}\ (k \,@\, 2)) - 1$$

は引き算であるから `int` を返す予定だったが, 実際は `string` を返した. このような, reset の中身が返す型を answer type と呼ぶ. この例では, shift 命令により answer type が `int` から `string` に変化した.

関数の型は $(t_1 \to t_2 \,@\mathrm{cps}[t_3, t_4, \alpha])$ という形になる. これで大雑把には「t_1 を受け取ったら t_2 を返す関数」を意味するが, この関数を実行すると answer type が t_3 から t_4 へ変化し, この関数の本体部分の注釈は α であることを示す. この α が ϵ なら, この関数の本体は pure であり, i なら impure であることを示す.

2.2　型規則

対象言語の型規則を定義した. その一部として型規則 fun は以下である. ただし

$$t' = (t_1 \to t_2 \,@\mathrm{cps}[t_3, t_4, \alpha_1])$$

とする.

$$\frac{(\alpha_2 \le \alpha_1) \quad (t_3 \ne t_4 \Rightarrow \alpha_2 = i) \qquad \Gamma, x^{\,\epsilon} : t_1 \vdash e^{\,\alpha_2} : t_2 \,@\mathrm{cps}[t_3, t_4, \alpha_2]}{\Gamma \vdash (\lambda^{\alpha_1} x.e^{\,\alpha_2})^{\,\epsilon} : t' \,@\mathrm{cps}[t_5, t_5, \epsilon]}\ \text{fun}$$

関数の本体部分の注釈 α_2 と返される関数の型の注釈 α_1 の間に, $(\alpha_2 \le \alpha_1)$ という制約がある. これは, 関数本体が impure なら関数の型も impure になることを示している. $(\alpha_2 = \alpha_1)$ ではなく $(\alpha_2 \le \alpha_1)$ となっているのは, 関数本体が pure でも, 周りの状況によってその関数を impure と扱いたい場合があるためである.

この型規則は, 基本的には Asai and Kameyama [1] の型規則に注釈と制約を加えたものである.

型判断は,

$$\Gamma \vdash e^{\alpha} : t_1 \ @\mathrm{cps}[t_2, t_3, \alpha]$$

の形をしている. これは, 型環境 Γ のもとで式 e^{α} は型 t_1 を持ち, answer type を t_2 から t_3 に変え, 注釈 α を持つことを意味する. 注釈 α は式 e が pure か impure かを示す. 式 e に付く注釈 α は, answer type に付く注釈 α と一致する.

制約は以下の 3 種類からなる.

$$(\_ = \_) \qquad (\_ \leq \_) \qquad (\_ \neq \_ \Rightarrow \_ = i)$$

これらの制約が満たされる場合は以下である.

$$(\epsilon = \epsilon) \qquad (i = i)$$
$$(\epsilon \leq \epsilon) \qquad (\epsilon \leq i) \qquad (i \leq i)$$
$$(t_1 \neq t_1 \Rightarrow \epsilon = i) \qquad (t_1 \neq t_2 \Rightarrow i = i)$$

制約 $(\alpha \leq \alpha')$ は, α が i なら α' も i にならなければならず, α' が ϵ なら α も ϵ にならなければならないことを示す. ϵ と i の間には, $\epsilon < i$ の大小関係がある.

最後の制約は型規則では, 最初の 2 つには型が入り

$$(t_1 \neq t_2 \Rightarrow \alpha = i)$$

という形になる. これで, t_1 と t_2 が異なる型ならば α は i にならなければならないことを示す. 逆に, α が ϵ にならば, t_1 と t_2 は同じであると解釈しても良い.

型規則の前提部に $@\mathrm{cps}[t_1, t_2, \alpha]$ が出てくる度に $(t_1 \neq t_2 \Rightarrow \alpha = i)$ という制約が加わっている. これで, answer type が変わるなら α は i になり, α が ϵ なら $t_1 = t_2$ となることを示している.

3 制約解消

入力の式を与えられたら, まず型規則に従って型推論を行う. その際, 注釈と制約はそのまま保持する. 注釈と制約を除けば型規則はこれまでの規則とほぼ同一なので, 普通の let 多相の入った型推論をすれば良い. その結果, 入力の各式にはユニークな注釈の変数が割り振られ, それに対する制約が生成される.

制約に対する解とは, 全ての注釈の変数に対する注釈の割り当てのうち, 全ての制約を満たしているものである. 生成された制約に対して, 全ての注釈に i を入れると解になっている. これは直感的には, 全ての式を CPS 変換することが, 解の 1 つであるということを意味する. 従って, 解は必ず 1 つは存在することが分かる. 本稿の制約解消の目標は, なるべく小さい解, つまりなるべく ϵ が多い解を見つけることである.

しかし, なるべく小さい解を求めるのは簡単ではない. 一般には「最良の解」は存在しないことが分かっている. 本研究では, 型推論と制約解消を過度に複雑にすることのない範囲での制約解消を提供する. 制約解消規則は, この修論要旨には載せない.

4 Selective CPS 変換

制約解消が済み, 式の各部に ϵ か i の注釈が付いたら, それに従って i の注釈が付いている箇所のみ CPS 変換する selective CPS 変換を定義した. 注釈付きの式の型規則は注釈を無視した簡約規則のもとで subject reduction を満たすことを示すことができる. Pure な式 e^{ϵ} に対する selective CPS 変換を $\| e^{\epsilon} \|_p$, impure な式 e^{i} に対する selective CPS 変換を $\| e^{i} \|$ とすると, 以下の定理が満たされる. ただし, E は CPS 変換後の式の任意の評価文脈とする.

定理 4.1

$$\vdash e^{\epsilon} : t_1 \ @\mathrm{cps}[t_2, t_2, \epsilon] \wedge e^{\epsilon} \rightarrow^* v^{\epsilon} \quad \Rightarrow \quad \| e^{\epsilon} \|_p \rightarrow^* \| v^{\epsilon} \|_p$$
$$\vdash e^{\epsilon} : t_1 \ @\mathrm{cps}[t_2, t_2, i] \wedge e^{i} \rightarrow^* v^{\epsilon} \quad \Rightarrow$$
$$\| e^{i} \| \ @ \ (\lambda x.\, E[x]) \rightarrow^* E[\| v^{\epsilon} \|_p]$$

これは, 注釈のついたプログラムの実行結果と selective CPS 変換後のプログラムの実行結果が対応することを示している. この定理は, Danvy and Filinski [3] の証明に沿った形で示すことができる.

5 実行例

N-Queen 問題を解く簡単なプログラムで selective CPS 変換を行った. このプログラムはバックトラックを shift/reset を使って実装している. Selective に CPS 変換することで, プログラム全体を CPS 変換したものに比べて, 約 28% ほどの実行時間の改善が見られた.

6 まとめ

本稿では shift/reset を含むプログラムの selective CPS 変換を定義した. 具体的には以下を行った.

- 型と式に注釈を含む型規則を定義した
- 型推論で生成された制約に対する制約の解消法を定義した. これは, 過度に型推論と制約解消を複雑にしないである程度の解を得られるものである
- 注釈が付いた項の selective CPS 変換を定義した

今のところ, 本稿で定義した制約解消法で, 一般的なプログラムには良い注釈が付いている. しかし, 我々は限られたプログラムしか考察できていない. そのため, 現在の制約解消法よりも強力なものが必要なプログラムを探すことは, 今後の課題である.

参考文献

[1] Asai, K. and Kameyama, Y.: Polymorphic Delimited Continuations, *5th Asian Symposium on Programming Languages and Systems (APLAS 2007)*, pp.239–254, 2007

[2] Danvy, O. and Filinski, A.: Abstracting control, *the 1990 ACM Conference on Lisp and Functional Programming*, pp.151–160, 1990

[3] Danvy, O. and Filinski, A.: Representing Control : a Study of the CPS Transformation, *Mathematical Structures in Computer Science*, Volume 2, Issue 4, pp.361–391, 1992

[4] Nielsen, L. R.: A Selective CPS Transformation, BRICS Research Report RS-01-30, Department of Computer Science, Aarhus University, 2001