日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

コントロールオペレータを持つマルチステージ言語の型推論

小鍛治 雄一郎　亀山 幸義

マルチステージ・プログラミング言語は，型システムを用いて，実行時コード生成の安全性, すなわち, コード生成が型安全であるだけでなく，生成されたコードが構文的に正しく，自由変数を持たず，型安全であることを静的に保証する言語である．マルチステージ・プログラミングに関する先行研究が, 純粋なラムダ計算を対象にして安全性を保証する型システムを構築したのに対し，我々は, 計算のエフェクト (副作用) をマルチステージ・プログラミングに導入する研究を行っている．杉浦と亀山は，2009 年に,「タグ付き shift/reset」というコントロールオペレータを導入したマルチステージの体系を構築し，コード生成時に let-insertion や if-insertion と呼ばれる有用な手法が安全に利用可能であることを示した．本研究は，彼らの型システムを整理した上で，この型システムに対する型推論アルゴリズムを設計した．

1　はじめに

メタプログラミング (プログラムを生成するプログラムの作成) は、プログラムの保守性と実行効率の両立や、ドメイン特化言語処理系の高速化など様々な場面で有用である。プログラム言語の立場からのメタプログラミングとしては、主に以下の 3 つの方法が知られている。(本稿では、便宜上、生成されるプログラムをコードと呼び、メタプログラムを単にプログラムと呼ぶことによって区別する。)

- 文字列としてのコードを生成
- quasiquote(擬似引用) と unquote (backquote)
- マルチステージプログラミング (MSP)

第 1 の方法は、特別なサポートがないプログラム言語で利用可能であるが、生成されるコードの安全性を保証することは難しく、実際、エラーが起きやすいことが知られている。

Lisp 系言語では、quasiquote/unquote の利用によるメタプログラミングが可能であり、様々なメタプログラムが開発されてきた。次の例は Scheme で `n` 乗を計算するコードを生成したものである。

```
(define (staged_power n)
  (if (= n 0) `1
    `(* x ,(staged_power (- n 1)))))
`(lambda (x) ,(staged_power 3))
=> (lambda (x) (* x (* x (* x 1))))
```

MSP では、quasiquote/unquote と同様な機能を持つ計算体系に型システムを与えることにより、生成されたコードの安全性を静的に保証するものである。ここで言う安全性は、構文に誤りがないこと、自由変数を持たないこと、型安全であることなどを指す。次の例は、関数型言語 OCaml の MSP 拡張である MetaOCaml で、`staged_power` と同様のメタプログラムを記述したものである。

```
let rec staged_power n cx =
  if n=0 then .<1>. else
  .< .~cx * .~(staged_power (n-1) cx)>.;;
.<fun x -> .~(staged_power 3 .<x>.)>.;;
=> .<fun x_1 -> (x_1 * (x_1 * (x_1 * 1)))>.
```

Type Inference for a Multi-Stage Language with Control Operators

Yuichiro Kokaji, Yukiyoshi Kameyama, 筑波大学 システム情報工学研究科コンピュータサイエンス専攻, Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba.

生成されたコードは、上記の Scheme の例と同様であるが、束縛変数 `x` が `x_1` に名前替えされているところから、α 同値を意識した体系であることがわかる。

MSP のための型システムはこれまでいろいろなものが研究されてきたが、純粋なラムダ計算に MSP の機能を追加したものがほとんどであった。(Davies と Pfenning による $\lambda\Box$[3]、Davies による $\lambda\bigcirc$[2]、Taha と Nielsen による λ^{α} [5]、Calcagno,Moggi,Taha による λ^{i} [1]、Tsukada と Igarashi による $\lambda^{\triangleright}$ [6] など。)

MSP の適用範囲を広げるため、我々は、計算エフェクト (副作用) をもつ MSP 体系とその型システムの構築を行っている。(Kameyama, Kiselyov, Shan による $\lambda_1^{\oslash}$ [4]、 杉浦と亀山による λ_1^{mp} [7] など。)

本研究は、この方向を更に一歩進め、杉浦と亀山による λ_1^{mp} 体系に対する型推論アルゴリズムを構築することを目標とする。λ_1^{mp} の型システムは、いわゆる type-and-effect system であり、複雑なエフェクトを持つために、人間が直接型を記述するのは比較的困難である。また、この体系は OCaml などの ML 系言語との共存を意図しているため、暗黙の型付けであることが強く望まれる。これらの理由により、λ_1^{mp} に対する自動的な型推論アルゴリズムの構築が必要であった。

本研究では、型推論アルゴリズムを設計し、その性質を証明する作業を行うにあたり、体系 λ_1^{mp} の型システムを整理・変形し、より理解しやすい体系とした。†1本稿では、新しい体系を単に λ_1^{mp} と呼ぶ。新しい λ_1^{mp} の設計と型推論アルゴリズムの設計が本稿の主要な貢献である。最後に、let 多相への拡張について述べる。

2 体系 λ_1^{mp} の構文の定義

本章では、体系 λ_1^{mp} を定義する。

マルチステージ言語は、コードを生成する段階と生成されたコードが実行される段階という複数の段階 (ステージ) を持つ。Taha らに従い、ステージを区別するため、environment classifier α を導入する。α は、現在のステージより 1 つ先の世界で使える変数環境に付けた名前であり、「1 つ先のステージ」を抽象的に表す記号である。たとえば、$\alpha_1\alpha_3\alpha_2$ という長さ 3 の列は、現在より 3 つ先の将来のステージ (の 1 つ) を表す。ただし、本稿の範囲では、煩雑さを避けるため、長さが 2 以上となるステージは扱わない。つまり、本稿では、ステージは、長さ 0 の列 (これ自身を「0」と表す) か、あるいは、environment classifier 1 つだけからなる列「α_i」のどちらかである。environment classifier のことを以下では単に classifier と呼ぶ。

また、計算エフェクトの種類を区別するため、エフェクトに対するタグ p を導入する。本稿では、計算エフェクトを起こすのは shift/reset というコントロールオペレータだけであり、p は shift/reset の種類を区別するための名前の役割を果たす†2。

$$\begin{array}{ll} \alpha ::= \alpha_1, \alpha_2, \cdots & \text{classifier} \\ L ::= 0 \mid \alpha & \text{stage} \\ p ::= p_1, p_2, \cdots & \text{tag} \end{array}$$

次に、図 1 により、λ_1^{mp} の構文の定義を与える。

e^0 と e^1 は、それぞれ、第 0 および第 1 ステージの式である。ここで 0 はコードを生成するステージ、1 はコードを実行するステージを意味する。本稿の型システムのもとでは、α_1 ステージの式と α_2 ステージの式がまざることはないので、これらのステージを総称して「第 1 ステージ」と呼んでいる。

一方、v^0 と v^1 はそれぞれのステージにおける値をあらわす。変数 x、整数 i、加算 $e + e$、条件式 `if` e `then` e `else` e、λ 抽象 $\lambda x.e$、適用 ee は通常の λ 計算と同様である。$\mathcal{S}_p k.e$ はタグ p を持つ shift 式で、e において変数 k が束縛される。shift 式で捕捉される継続の範囲を限定するのが reset であり、$\{e\}_p$ という形式をもつ。

本稿では、この体系の操作的意味論は与えない。関心のある読者は、[7] を参照されたい。

†1 ただし、この変形は表層的なものであり、本質的な変更は行っていない。

†2 なお、先行研究では p を「プロンプト」と呼んでいるが、この名称は実態とは合わないので、本稿ではこの名称は使わない。

$$
\begin{aligned}
e^0 ::= {} & v^0 \mid e^0 e^0 \mid e^0 + e^0 \mid \texttt{if } e^0 \texttt{ then } e^0 \texttt{ else } e^0 \\
& \mid \mathcal{S}_p k.e^0 \mid \{e^0\}_p \mid \texttt{<}e^1\texttt{>} \mid \texttt{run } e^0 \\
e^1 ::= {} & v^1 \mid e^1 e^1 \mid e^1 + e^1 \mid \texttt{if } e^1 \texttt{ then } e^1 \texttt{ else } e^1 \\
& \mid \mathcal{S}_p k.e^1 \mid \{e^1\}_p \mid {\sim}e^0 \\
v^0 ::= {} & x \mid i \mid \texttt{true} \mid \texttt{false} \mid \lambda x.e^0 \mid \texttt{<}v^1\texttt{>} \\
v^1 ::= {} & x \mid i \mid \texttt{true} \mid \texttt{false} \mid \lambda x.v^1 \mid v^1 v^1 \mid v^1 + v^1 \mid \texttt{if } v^1 \texttt{ then } v^1 \texttt{ else } v^1 \\
& \mid \mathcal{S}_p k.v^1 \mid \{v^1\}_p
\end{aligned}
$$

図 1　λ_1^{mp} の構文

3　体系 λ_1^{mp} の型システム

まず、型とエフェクトの構文を与える。

$$
\begin{aligned}
\sigma, \tau ::= {} & \texttt{bool} \mid \texttt{int} \mid \sigma \to \tau/\epsilon && \text{type} \\
\epsilon ::= {} & [\,] \mid \epsilon, p && \text{effect} \\
\Gamma ::= {} & [\,] \mid \Gamma, (x:\sigma)^L && \text{typing context}
\end{aligned}
$$

型は、$\texttt{bool}, \texttt{int}$ という基底型のほか、関数型 $\sigma \to \tau/\epsilon$ である。ここで、関数型に付加された ϵ は、エフェクトであり、その中身は、タグの有限列である。また、後に使う typing context Γ は $(x:\sigma)^L$ の形の有限列である。ここで L はステージをあらわし、変数 x がステージ L の変数であることを意味する。なお、有限列の順番は意味がないので、後には、しばしば有限列を集合と同一視する。

後述する型システムでは、エフェクト族を表す必要があるので、ここで構文を以下のように与えておく。

$$\overline{\epsilon} = \{\epsilon^\ell\}_{\ell \le L}$$

ただし、L はステージであり、$\ell \le L$ とは、ℓ が L と等しいか、L の prefix であることを意味する。たとえば、$L = \alpha$ のとき、$\{\epsilon^\ell\}_{\ell=0,\alpha}$ という集合がエフェクト族である。各 ℓ に対する ϵ^ℓ は、エフェクト (タグの有限列) である。インデックスされた各エフェクトがすべて空列 $[\,]$ であるエフェクト族を $\overline{[\,]}^L$ (L が文脈から明らかな時は、$\overline{[\,]}$) と表す。

型システムにおける型判断 (judgement) は以下の形を取る。

$$\Gamma; \overline{\epsilon} \vdash^L e : \tau$$

これは、ステージ L において、typing context Γ とエフェクト族 $\overline{\epsilon}$ のもとで、式 e は τ という型をもつことを意味する。

図 2 に、型システムの推論規則を掲載する。

なお、先行研究にならい、すべてのタグ p に対して、あらかじめ、その型、エフェクト、ステージが定められていると仮定する。以下の型付け規則では $(p : \tau/\epsilon_1)^L \in \Sigma$ は、タグ p が型 τ, エフェクト ϵ_1, ステージ L を持つことが、環境 Σ で宣言されているという意味である。(Σ は固定して考える。)

型システムの定義において、$\overline{\epsilon} \oplus \overline{\epsilon}'$ は、各インデックスごとに $\overline{\epsilon}$ の要素と $\overline{\epsilon}'$ の要素の和集合 (あるいは列の連接) を取ってできるエフェクト族を表す。また、$\overline{\epsilon} \oplus^L \epsilon_1$ は、$\overline{\epsilon}$ の第 L 要素を、$\epsilon^L \cup \epsilon_1$ に置きかえて得られるエフェクト族を表す。

4　型推論アルゴリズム

本稿で述べる型推論アルゴリズムは、Principal Typing ではなく Principal Type を返すように設計する[†3]。let 多相型のない本稿の型システムで Principal Type を返す型推論アルゴリズムを設計するのは奇妙であるが、近い将来に let 多相型を導入する計画があるため、このような形とした。

4.1　型変数とエフェクト変数等の導入

型推論アルゴリズムを記述するため、型変数 t を導入する。

$$\sigma, \tau ::= t \mid \texttt{bool} \mid \texttt{int} \mid \sigma \to \tau/\epsilon$$

また、エフェクトには、エフェクトをあらわす変数 δ のほか、エフェクト (列) を集合と見なしたときの和

†3　Principal であることの証明は未完成である。

$$\frac{\Gamma;\overline{\epsilon}\vdash^L e:\tau}{\Gamma,\Gamma';\overline{\epsilon}\oplus\overline{\epsilon}'\vdash^L e:\tau}\ \text{weaken}$$

$$\frac{(i \text{ is an integer literal})}{[\,];\overline{[\,]}\vdash^L i:\texttt{int}}\ \text{int}\qquad\frac{(b=\texttt{true},\texttt{false})}{[\,];\overline{[\,]}\vdash^L b:\texttt{bool}}\ \text{bool}$$

$$\frac{}{(x:\tau)^L;\overline{[\,]}\vdash^L x:\tau}\ \text{var}$$

$$\frac{\Gamma;\overline{\epsilon}\vdash^L e_1:\texttt{bool}\quad\Gamma;\overline{\epsilon}\vdash^L e_2:\tau\quad\Gamma;\overline{\epsilon}\vdash^L e_3:\tau}{\Gamma;\overline{\epsilon}\vdash^L \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3:\tau}\ \text{if}$$

$$\frac{\Gamma;\overline{\epsilon}\vdash^L e_1:\texttt{int}\quad\Gamma;\overline{\epsilon}\vdash^L e_2:\texttt{int}}{\Gamma;\overline{\epsilon}\vdash^L e_1+e_2:\texttt{int}}\ \text{plus}$$

$$\frac{\Gamma,(x:\sigma)^L;\overline{\epsilon}\vdash^L e:\tau\quad(\overline{\epsilon}^\ell=[\,]\ \text{ for all }\ \ell<L)}{\Gamma;\overline{[\,]}\vdash^L \lambda x.e:\sigma\to\tau/\epsilon^L}\ \lambda$$

$$\frac{\Gamma;\overline{\epsilon}\vdash^L e_1:\sigma\to\tau/\epsilon_1\quad\Gamma;\overline{\epsilon}\vdash^L e_2:\sigma}{\Gamma;\overline{\epsilon}\oplus^L\epsilon_1\vdash^L e_1e_2:\tau}\ \text{app}$$

$$\frac{(p:\tau/\epsilon_1)^L\in\Sigma,\quad\epsilon^L-\{p\}\subset\epsilon_1\qquad\Gamma,(k:\sigma\to\tau/\epsilon_1)^L;\overline{\epsilon}\vdash^L e:\tau}{\Gamma;\overline{\epsilon}\oplus^L\{p\}\vdash^L\mathcal{S}_pk.e:\sigma}\ \text{shift}$$

$$\frac{(p:\tau/\epsilon_1)^L\in\Sigma,\quad\epsilon^L-\{p\}\subset\epsilon_1\qquad\Gamma;\overline{\epsilon}\vdash^L e:\tau}{\Gamma;\overline{\epsilon}\ominus^L\{p\}\vdash^L\{e\}_p:\tau}\ \text{reset}$$

$$\frac{\Gamma;\overline{\epsilon}\vdash^{L,\alpha} e:\tau}{\Gamma;\{\epsilon^\ell\}_{\ell\le L}\vdash^L \texttt{<}e\texttt{>}:\texttt{<}\tau/\epsilon^L\texttt{>}^\alpha}\ \text{brackets}$$

$$\frac{\Gamma;\overline{\epsilon}\vdash^L e:\texttt{<}\tau/\epsilon_1\texttt{>}^\alpha}{\Gamma;\overline{\epsilon}\oplus^L\epsilon_1\vdash^{L,\alpha}\sim e:\tau}\ \text{escape}$$

$$\frac{\Gamma;\overline{[\,]}\vdash^L e:\texttt{<}\tau/[\,]\texttt{>}^\alpha\quad(\alpha\notin\texttt{FC}(\Gamma))}{\Gamma;\overline{[\,]}\vdash^L\texttt{run}\ e:\tau}\ \text{run}$$

図 2　型システム

集合や差集合の演算を追加する。

$$\epsilon ::= \delta \mid [\,] \mid \epsilon\cup\{p\} \mid \epsilon-\{p\}$$

ここでのポイントは、$\epsilon\cup\epsilon'$ のようなエフェクトは必要としないこと、すなわち、(拡張された) エフェクトは、たかだか 1 つのエフェクト変数しか持たないことである。これは、型推論によって生成される制約が効率よく解けるための鍵となる性質である。

4.2　型推論アルゴリズムの概要

型推論アルゴリズムの入力は、以下のいずれかの型判断である。(ステージ L は 0 か α の形であるため。) ただし、各要素は型変数やエフェクト変数を含み得ることに注意されたい。

$$\Gamma;\epsilon_0\vdash^0 e:\tau$$
$$\Gamma;\epsilon_0,\epsilon_\alpha\vdash^\alpha e:\tau$$

本稿の型推論アルゴリズムは、いわゆる制約に基づく型推論アルゴリズムであり、以下の 2 つのステッ

プから構成される。

- 型およびエフェクトに関する制約の生成
- 上記の制約の解消

4.3 制約生成

制約生成とは、与えられた型判断 ($\Gamma; \epsilon_0 \vdash^0 e : \tau$ など) に対応する型制約とエフェクト制約を生成するフェーズである。生成される制約は、それぞれ以下の形をした制約の論理積 (conjunction) である。

$$\sigma = \tau \qquad \text{型制約}$$
$$\epsilon \subset \epsilon \qquad \text{エフェクト制約}$$

具体的な制約の生成方法は、型システムにおける型付け規則によって定まる。一例として app 規則について考える。ここでは、$L = \alpha$ と仮定する。

$$\frac{\Gamma; \overline{\epsilon} \vdash^L e_1 : \sigma \to \tau/\epsilon_1 \quad \Gamma; \overline{\epsilon} \vdash^L e_2 : \sigma}{\Gamma; \overline{\epsilon} \oplus^L \epsilon_1 \vdash^L e_1 e_2 : \tau} \text{ app}$$

いま、$\Gamma; \epsilon_0, \epsilon_\alpha \vdash^\alpha e_1 e_2 : \tau$ の型推論を行っているとする。これに app 規則を (下から上に) 適用したとき、以下のようなステップで型推論が行われる。

- 新しい型変数 t を生成する。(上記規則の σ に該当する。)
- 新しいエフェクト変数 δ を生成する。(上記規則の ϵ_1 に該当する。)
- $\delta \subset \epsilon_\alpha$ というエフェクト制約を生成する。
- $\Gamma; \epsilon_0, \epsilon_\alpha \vdash^L e_1 : t \to \tau/\delta$ を型推論する。
- $\Gamma; \epsilon_0, \epsilon_\alpha \vdash^L e_2 : t$ を型推論する。
- 上記で生成された型制約とエフェクト制約を返す。

ここでのポイントは、エフェクト制約は $\epsilon \subset \epsilon'$ の形のみであるという点である。

4.4 制約解消

制約解消とは、生成された型制約とエフェクト制約を解消して、型変数およびエフェクト変数への制約に還元することである。

1 つ注意すべき点は、型にはエフェクトを含み、型制約の解消のたびに新たなエフェクト制約が生成されることがあるため、両者の制約の解消を同時に行う必要がある点である。

- 型制約の解消 (型変数制約への還元): 通常の単一化アルゴリズムとほぼ同様であるので省略する。
- エフェクト制約の解消: 型システムを注意深く追うことにより、生成されるエフェクト制約は、両辺にエフェクト変数を 1 つ以下しか持たないものに変形できることがわかる。これを用いて、エフェクト制約を、下記の形の制約の論理積に還元できる。ただし、C は具体的なエフェクト、つまり、$\{p_1, p_2, \cdots, p_n\}$ の形のエフェクトである。

$$\delta \subset C$$
$$C \subset \delta$$
$$\delta \subset \delta' \cup C$$

どのタグもグローバル環境 Σ で宣言されていなければいけないため、エフェクト変数 δ の変域は有限である。よって、上記の形のエフェクト制約が解を持つかどうかは決定可能である。

最後に、本稿で与えた型推論アルゴリズムは、以下の形の健全性および完全性を満たすと予想しているが、詳細は将来課題である。

- 型判断 $\Gamma; \overline{\epsilon} \vdash^L e : \tau$ に対して型推論アルゴリズムが、型変数に対する代入 θ_t、および、エフェクト変数に対する制約 ϕ_e をもって成功するならば、ϕ_e を満たす任意のエフェクト変数への代入 θ_e に対して、$(\Gamma; \overline{\epsilon} \vdash^L e : \tau)\theta_t\theta_e$ が導出可能である。
- 型判断 $\Gamma; \overline{\epsilon} \vdash^L e : \tau$ および代入 θ_t と θ_e に対して、$(\Gamma; \overline{\epsilon} \vdash^L e : \tau)\theta_t\theta_e$ が導出可能であれば、型推論アルゴリズムは、$\Gamma; \overline{\epsilon} \vdash^L e : \tau$ に対して成功する。また、その返す結果 θ'_t と ϕ_e に対して、θ'_t は θ_t より一般的であり、θ_e は ϕ_e を満たす代入である。

5 まとめ

杉浦と亀山による体系 λ_1^{mp} [7] の型システムを (表層的に) 改良した型システムを与え、それに対する型推論アルゴリズムを構築した。

本稿の執筆時点では、Principal Type や型推論アルゴリズムの正しさ (健全性、完全性) の証明は完成していないが、特に問題は見つかっておらず、近いう

ちに証明できると考えている。

本稿の次なる目標は、以下のような let 多相の導入である。

$$\frac{\Gamma;\overline{[\,]} \vdash^L v : \sigma \quad \Gamma, (x : \forall\sigma)^L; \overline{\epsilon} \vdash^L e : \tau}{\Gamma; \overline{\epsilon} \vdash^L \texttt{let}\ x = v\ \texttt{in}\ e : \tau}\ \text{let}$$

ここで $\forall\sigma$ は、σ を多相的にした型を表すが、ここでは、単に型変数についての多相化だけではなく、classifier およびエフェクトについても多相化することが望ましい。classifier の多相化ができていない λ_1^{mp} では、run (コード実行) の機能が実際にはあまり有用でないことがわかっているため、この拡張は是非必要なものである。さらに、`fun x -> x` のような関数は、任意のエフェクトのもとでも利用可能とすべきであるため、エフェクトに関する多相性も必要である。

型システムに対するこれらの拡張の際、本稿で述べた型推論アルゴリズムは自然に拡張できると考えているが、その詳細は、将来課題である。

参 考 文 献

[1] Calcagno, C., Moggi, E., and Taha, W.: ML-Like Inference for Classifiers, *ESOP*, Schmidt, D. A.(ed.), Lecture Notes in Computer Science, Vol. 2986, Springer, 2004, pp. 79–93.

[2] Davies, R.: A Temporal Logic Approach to Binding-Time Analysis, *LICS*, 1996, pp. 184–195.

[3] Davies, R. and Pfenning, F.: A Modal Analysis of Staged Computation, *Journal of the ACM*, Vol. 48, No. 3(2001), pp. 555–604.

[4] Kameyama, Y., Kiselyov, O., and Shan, C.: Shifting the stage: staging with delimited control, *PEPM*, 2009, pp. 111–120.

[5] Taha, W. and Nielsen, M. F.: Environment classifiers, *POPL*, 2003, pp. 26–37.

[6] Tsukada, T. and Igarashi, A.: A Logical Foundation for Environment Classifiers, *TLCA*, 2009, pp. 341–355.

[7] 杉浦啓介, 亀山幸義: コード実行機能と計算エフェクトを持つ型付きマルチステージ言語, コンピュータソフトウェア, (掲載予定).