副作用を持つプログラミング言語を型安全に定義するためのライブラリの開発に向けて

東京工業大学 情報理工学院 数理・計算科学系
18B10155 津山勝輝　　指導教員 増原 英彦

1　はじめに

型安全インタプリタ [1] とはプログラミング言語の意味論仕様の記述方法のひとつである。型安全インタプリタの重要な特徴は、メタ言語の型検査を通して対象言語の型安全性を証明できることである。型安全インタプリタの評価関数は、対象言語で型付け可能な項を表現する、**型付き抽象構文木**の変換として実装される。メタ言語上の型安全インタプリタの型で対象言語の型安全性を表明することで、カリー=ハワード同型により、型安全インタプリタの本体は対象言語の型安全性の証明とみなすことができる。

実用的な言語機能を持った対象言語の型安全インタプリタには、その機能のための証明項によって、記述が複雑になるという問題がある。先行研究 [3, 10] は、インタプリタの実装を抽象化し、証明項を隠すことでこの問題に対処している。例えば、可変状態を持つ言語の型安全インタプリタは、可変状態の数が評価に従って増えることを示す、繁雑な証明項を持つ。Poulsen ら [3] はモナドを利用した抽象化で証明項をカプセル化している。

本研究の目的は、エフェクト [9] やコエフェクト [2] を持つ言語の型安全インタプリタの簡潔な実装を支援するライブラリの開発である。本論文では、特にエフェクトを持つ対象言語の型安全インタプリタに有効な抽象化を分析するため、限定継続命令 (`shift`/`reset`) を持つ言語 $\lambda_{s/r}$ [11] と、代数的エフェクトとハンドラを持つ言語 λ_e [11] の型安全インタプリタを作成した。インタプリタは依存型言語 Agda [7] で、継続渡しスタイル (CPS) で記述した。また、型付き抽象構文木の変数束縛には Parameterized Higher-Order Abstract Syntax (PHOAS) [4] を用いた。

- $\lambda_{s/r}$ の型付き抽象構文木は、実行前と実行後の文脈が返す型の 2 つを持つ。型安全インタプリタは、`shift`/`reset` のための CPS 意味論 [5] に基づいて実装した。
- λ_e の型付き抽象構文木は、項が実行しうるオペレーションの集合を持つ。代数的エフェクトとハンドラのための CPS 変換 [6] に基づき、λ_e の CPS 意味論を構築し、それに対応する CPS インタプリタを実装した。

本概要では λ_e の実装を紹介する。

2　λ_e の型付き構文木

代数的エフェクトとハンドラとは、例外・状態などの副作用を表すための機能である [9]。代数的エフェクトを持つ言語では、オペレーション呼び出しによって副作用を発生させ、副作用はハンドラによって処理される。

代数的エフェクトとハンドラを持つ言語 λ_e を、以下のように定義する。

$$
\begin{array}{rrcl}
\text{(値)} & V, W & ::= & \mathit{unit} \mid x \mid \lambda x.C \\
\text{(計算)} & M, N & ::= & V\ W \mid \mathit{return}\ V \mid \\
 & & & \mathit{do}\ l\ V \mid \\
 & & & \mathit{handle}\ M\ \mathit{with}\ H \\
\text{(ハンドラ)} & H & ::= & \{\mathit{return}\ x \to M\} \mid \\
 & & & \{l\ p\ k \to M\} \uplus H
\end{array}
$$

λ_e の項は、エフェクトを起こさない (純粋な) **値**とエフェクトを起こしうる**計算**からなる。$\mathit{do}\ l\ V$ は、l でラベル付けされたオペレーションを引数 V で呼び出す。$\mathit{handle}\ M\ \mathit{with}\ H$ は、計算 M が起こす副作用をハンドラ H で処理する。

型付き抽象構文木は、対象言語の構文と型システムに基づいて定義する。以下は λ_e の型付き抽象構文木の実装である。

```
data Val (Var : VTy → Set) : VTy → Set
data Cmp (Var : VTy → Set) : CTy → Set
data Handler (Var : VTy → Set) : HTy → Set
```

Val、Cmp、Handler は λ_e の型付け可能な値、計算、ハンドラを表すデータ型である。これらのデータ型は、それぞれの型を表すデータ型の VTy、CTy、HTy の値に依存している。例えば、

Val $Var\ A$ は λ_e の値型を表す VTy の要素 A に依存する型であり、λ_e で A 型に型付けされる値を表す。ここで Var は PHOAS 特有のパラメータで、変数が参照する値の型を表す。

型付き抽象構文木のコンストラクタは、型規則に対応して定義する。例えば、以下はオペレーション呼び出し (do) の定義であり、型規則 T-Do に対応している。

$$\mathsf{Do} : (\mathsf{op}\ A\ B) \in E \to \mathsf{Val}\ \ Var\ A \to \mathsf{Cmp}\ \ Var\ (B\ ,\ E)$$

$$\frac{(l : A \to B) \in E \qquad \Gamma \vdash V : A}{\Gamma \vdash do\ l\ V : B!E}\ (\text{T-Do})$$

Do の第一、第二引数、返り値の型は、それぞれ T-Do の 2 つの仮定 $(l : A \to B) \in E$、$\Gamma \vdash V : A$、結論 $\Gamma \vdash do\ l\ V : B!E$ に対応する。

3 λ_e の型安全インタプリタの実装

以上で定義した型付き抽象構文木を用いて、λ_e の型安全インタプリタを定義する。

$$\mathsf{eval} : \mathsf{Cmp}\ \tilde{}\mathsf{t}\ (A\ ,\ []) \to \tilde{}\mathsf{t}\ A$$
$$\mathsf{evalv} : \mathsf{Val}\ \tilde{}\mathsf{t}\ A \to \tilde{}\mathsf{t}\ A$$
$$\mathsf{evalc} : \mathsf{Cmp}\ \tilde{}\mathsf{t}\ C \to \tilde{}\mathsf{c}\ C$$

eval はトップレベルの λ_e プログラムの評価関数であり、これが型安全インタプリタそのものである。eval の型は λ_e の型安全性の言明を表しており、その第一引数 Cmp ˜t $(A, [])$ は空のエフェクトを持つ計算の型である。これは、トップレベルのプログラムがハンドラによって全ての副作用を処理する必要があることを示している。

eval の本体は evalv と evalc のを使って定義される。evalv は値のための、evalc は計算のためのインタプリタである。これらを λ_e の CPS 意味論に基づいて定義する。例えば、evalc による Do の評価は以下のように定義される。

$$\xi_c[do\ l\ V]\rho\kappa h = h(l,\ \xi_v[V]\rho,\ \lambda x.\kappa\ x\ h) \qquad (\text{E-Do})$$
$$\mathsf{evalc}\ (\mathsf{Do}\ label\ v)\ k\ h = h\ label\ (\mathsf{evalv}\ v)\ (\lambda\ x \to k\ x\ h)$$

評価規則 E-Do に定義されているように、ξ_c による計算 $do\ l\ V$ の評価は、ハンドラ h にラベル l、引数 V、継続 $\lambda x.\kappa\ x\ h$ を渡してオペレーションを処理する。インタプリタの実装では、E-Do に対応するように、ξ_c や ξ_v を Agda の evalc や evalv とし、継続 $\lambda x.\kappa\ x\ h$ を Agda の関数 ($\lambda\ x \to k\ x\ h$) として書き下し、それを E-Do の h と同様に Agda 上のハンドラ h に渡した。他の評価規則についても同様に、CPS 意味論と直接対応するように実装できることを確認した。

4 考察と今後の課題

考察 限定継続命令・代数的エフェクトとハンドラは類似した制御抽象機構である。本論文では、実際に双方の機能を持つ言語の型付き抽象構文木を PHOAS で実装することによって、共に CPS 意味論の書き下しとしてインタプリタを実装できることを確認した。この方法は、継続を操作する他のエフェクトを持つ言語に対しても有効であると予想され、共通部分のライブラリ化が可能だと期待される。

また、双方のインタプリタが CPS 意味論の書き下しであることから、本論文では明示的に引数に受け取っていた継続とハンドラを、モナドによって隠蔽できる見通しが立った。このモナドにより継続やハンドラを直接扱うことなく、手続き的に意味論を記述できることが期待できる。

今後の課題 今後は、エフェクトを持つ言語の型安全インタプリタをモナドで記述しライブラリ化する。また、コエフェクトを持つ言語の分析のため、quantitative types [2] などの機能を持つ言語のための型安全インタプリタを実装する。

References

[1] Altenkirch, T. and Reus, B.: Monadic Presentations of Lambda Terms Using Generalized Inductive Types, CSL '99 (1999).

[2] Atkey, R.: Syntax and Semantics of Quantitative Type Theory, LICS '18 (2018).

[3] Bach Poulsen, C., Rouvoet, A., Tolmach, A., Krebbers, R. and Visser, E.: Intrinsically-Typed Definitional Interpreters for Imperative Languages, *POPL* (2017).

[4] Chlipala, A.: Parametric Higher-Order Abstract Syntax for Mechanized Semantics, ICFP '08 (2008).

[5] Danvy, O. and Filinski, A.: Abstracting Control, LFP '90 (1990).

[6] Hillerström, D., Lindley, S., Atkey, R. and Sivaramakrishnan, K. C.: Continuation Passing Style for Effect Handlers, FSCD 2017 (2017).

[7] Norell, U.: *Towards a practical programming language based on dependent type theory*, Vol. 32 (2007).

[8] Plotkin, G. D.: Call-by-name, call-by-value and the λ-calculus, *Theoretical computer science* (1975).

[9] Pretnar, M.: An Introduction to Algebraic Effects and Handlers. Invited Tutorial Paper, *Electron. Notes Theor. Comput. Sci.*.

[10] Rouvoet, A., Bach Poulsen, C., Krebbers, R. and Visser, E.: Intrinsically-Typed Definitional Interpreters for Linear, Session-Typed Languages, CPP 2020 (2020).

[11] Tsuyama, S., Cong, Y. and Masuhara, H.: Intrinsically-Typed Interpreters for Effectful and Coeffectful Languages, Presented at the first Workshop on the Implementation of Type Systems (WITS 2022) (2022).