モジュールの静的解釈に関する より網羅的かつ堅牢な正当性の証明を目指して

2024 年 3 月 5–7 日
於 PPL 2024

諏訪 敬之 *1,*2　五十嵐 淳 *1

*1： 京都大学大学院情報学研究科
*2： 国立情報学研究所

概要　ML のモジュールシステムのファンクタ（例： `Set.Make`）は高い抽象化能力の反面パフォーマンスを悪化させやすく，オーバヘッド除去のために**ファンクタの適用を型検査時に解消する静的解釈** (*static interpretation*) という手法が提案されている

しかし → 既存手法での変換とその正当性の証明は，一部言語機能を除いたり，限定的な性質しか示していなかったりする

そこで → **より広範なモジュールシステムの機能を網羅し，より強い性質まで正当性が担保された静的解釈を実現したい**

本発表はその過程で得られている幾分か有力な定式化案の紹介です

静的解釈の具体例： `MakeSet`（`Set.Make` 相当）

```
module MakeSet = fun(Elem : Ord) → struct
  type t = BinTree.t (Elem.t × int)
  val empty = BinTree.leaf   val height btr = E_height
  val add x set = E_add
  ⋯
end :> sig
  type t :: •   val empty : t   val add : Elem.t → t → t  …
end
module WordSet = MakeSet(String)
val confNames = WordSet.add "PPL" WordSet.empty
```

集合を平衡二分木で実装

平衡二分木への要素挿入箇所決定のため，比較演算 `Elem.compare : Elem.t → Elem.t → int` を使う

↓

```
val z_StringCompare = e_StringCompare ; val z_BinTreeLeaf = e_BinTreeLeaf ; …
val z_WordSetEmpty = z_BinTreeLeaf ;
val z_WordSetHeight btr = e_WordSetHeight ;
val z_WordSetAdd x set = e_WordSetAdd ; …
val z_confNames = z_WordSetAdd "PPL" z_WordSetEmpty
```

非公開のメンバも出てくる

E_{add} から得られた式．特に `Elem.compare` だった箇所が $z_{StringCompare}$ に置換されている

既存手法と本研究の見込みとの比較

	1 階	高階	:>	変換後に型つく保証	分割コンパイル
初出，MLKit [Elsman 1999]	✔	✖	✖	✔	✖
Futhark [Elsman+ 2018]	▲※	▲※	✖	✖	✖
SML# [Bochao+ 2010]	✔	✖	✖?	✖	✔
本研究の見込み	✔	なんとか可能？ ✔	✔	✔	或る意味で達成 ✔

※[Elsman+ 2018] は高階ファンクタもサポートしているとするが，Coq による正当性証明では型の抽象化を行なわないものしか扱わず

特に，:> (*sealing*, *opaque signature ascription*) による型の抽象化と静的解釈との両立はどの既存研究でも未実現

本研究のアイディア

- SML# 向けの定式化 [Bochao+ 2010] を援用する
  - 分割コンパイルのためだけでなく，正当性証明にも有用に見える
- **:> による型の抽象化をサポートするためには，型変数として以下の 2 種類を区別して扱うとよいのではないか？**
  1. :> によって抽象化された型に対応する型変数 α
     - 裏に実体となる型がひとつ隠れている
  2. ファンクタの引数のシグネチャにより導入される型変数 δ
     - どんな型が実体になるかが本質的に不定

参考文献
- L. Bochao and A. Ohori. A flattening strategy for SML module compilation and its implementation. *Information and Media Technologies*, **5**(1), 2010.
- M. Elsman. Static interpretation of modules. In *Proc. of ICFP'99*, 1999.
- M. Elsman, T. Henriksen, D. Annenkov, and C. E. Oancea. Static interpretation of higher-order modules in Futhark: functional GPU programming in the large. In *Proc. of ICFP'18*, 2018.
- A. Rossberg, C. Russo, and D. Dreyer. F-ing modules. *Journal of Functional Programming*, **24**(5), 2014.

ソース言語（[Rossberg+ 2014] と同一，多くの糖衣構文あり）

モジュール式・束縛　（ファンクタ抽象 / 適用 / sealing）

$$M ::= X \mid M.X \mid \mathbf{struct}\ \overline{B}\ \mathbf{end} \mid \mathbf{fun}(X : S) \to M \mid X(X) \mid X :> S$$
$$B ::= \mathbf{val}\ X = E \mid \mathbf{type}\ X = T \mid \mathbf{module}\ X = M \mid \mathbf{include}\ M$$

シグネチャ・宣言

$$S ::= \mathbf{sig}\ \overline{D}\ \mathbf{end} \mid (X : S) \to S \mid \cdots$$
$$D ::= \mathbf{val}\ X : T \mid \mathbf{type}\ X :: K \mid \mathbf{module}\ X : S \mid \cdots$$

式 $E ::= \cdots \mid X \mid M.X$　　型 $T ::= \cdots \mid X \mid M.X$　　カインド $K ::= \cdots$

シグネチャの内部表現

静的解釈なし版（[Rossberg+ 2014] と等価）

ファンクタシグネチャ中で 2 種類の型変数を区別

$$\Sigma ::= \{\tau\} \mid \{= \tau :: \kappa\} \mid \{\overline{\ell : \Sigma}\} \mid \forall \Delta\,.\,\Sigma \to \exists A\,.\,\Sigma$$

型変数の量化子　$\Delta ::= \overline{\delta :: \kappa}$　　$A ::= \overline{\alpha :: \kappa}$　　カインド $\kappa ::= \bullet \mid \kappa \to \kappa$

抽象型を含む通常の型 $\tau ::= \delta \mid \alpha \mid \tau \to \tau \mid \cdots$

静的解釈用に拡張したシグネチャ

ターゲット型環境 γ を変数名の量化子としても使う

$$\mathcal{I} ::= \{\tau \rightsquigarrow z\} \mid \{= \tau :: \kappa\} \mid \{\overline{\ell : \mathcal{I}}\} \mid \forall(\Delta, \gamma)\,.\,\mathcal{I} \to \exists(\hat{A}, \gamma)\,.\,\langle \mathcal{I}, \boldsymbol{c} \rangle$$

値は変換後の変数名を伴う [Elsman 1999]　　適用時の変換結果の大枠をもつ [Bochao+ 2010]

抽象化しつつ "密かに実体を持ち出す" 量化子　$\hat{A} ::= \overline{\alpha :: \kappa \rightsquigarrow \hat{\tau}}$

抽象型を含まない実体型 $\hat{\tau} ::= \delta \mid \hat{\tau} \to \hat{\tau} \mid \cdots$　　ターゲット型環境 $\boldsymbol{\gamma} ::= \overline{z : \hat{\tau}}$

変換結果の束縛列 $\boldsymbol{c} ::= \overline{\mathbf{val}\ z = e}$　　変換結果の式　$e ::= z \mid e\ e \mid \cdots$

型環境　$\mathcal{G} ::= \cdot \mid \mathcal{G}, X : \mathcal{I} \mid \mathcal{G}, \alpha :: \kappa \rightsquigarrow \hat{\tau} \mid \mathcal{G}, \delta :: \kappa \mid \mathcal{G}, z : \hat{\tau}$

`MakeSet` につくシグネチャ

$$\begin{aligned}
&\forall(\delta_{elem} :: \bullet,\ z_{compare} : \delta_{elem} \to \delta_{elem} \to \mathtt{int})\,. \\
&\quad \{\ell_t : \{= \delta_{elem} :: \bullet\}, \ell_{compare} : \{\delta_{elem} \to \delta_{elem} \to \mathtt{int} \rightsquigarrow z_{compare}\}\} \to \\
&\exists(\alpha_{set} :: \bullet \rightsquigarrow \mathtt{bintr}\ (\delta_{elem} \times \mathtt{int}),\ z_{empty} : \mathtt{bintr}\ (\delta_{elem} \times \mathtt{int}), \\
&\quad z_{height} : \mathtt{bintr}\ (\delta_{elem} \times \mathtt{int}) \to \mathtt{int}, \\
&\quad z_{add} : \delta_{elem} \to \mathtt{bintr}\ (\delta_{elem} \times \mathtt{int}) \to \mathtt{bintr}\ (\delta_{elem} \times \mathtt{int}), \ldots)\,. \\
&\langle \{\ell_t : \{= \alpha_{set} :: \bullet\}, \ell_{empty} : \{\alpha_{set} \rightsquigarrow z_{empty}\}, \ell_{add} : \{\delta_{elem} \to \alpha_{set} \to \alpha_{set} \rightsquigarrow z_{add}\}, \ldots\}, \\
&\quad \mathbf{val}\ z_{empty} = z_{BinTreeLeaf} ; \mathbf{val}\ z_{height}\ \mathtt{btr} = e_{height} ; \mathbf{val}\ z_{add}\ \mathtt{x}\ \mathtt{set} = e_{add} ; \ldots \rangle
\end{aligned}$$

型つけ規則案

$$\frac{\mathcal{G} \vdash S \Rightarrow \exists(\Delta, \gamma)\,.\,\mathcal{I}}{\mathcal{G} \vdash D \Rightarrow \exists(\Delta, \gamma)\,.\,\overline{\ell : \mathcal{I}}}$$

$$\frac{\mathcal{G} \vdash B : \exists(\hat{A}, \gamma)\,.\,\langle \overline{\ell : \mathcal{I}}, \boldsymbol{c} \rangle}{\mathcal{G} \vdash B : \exists(\hat{A}, \gamma)\,.\,\langle \overline{\ell : \mathcal{I}}, \boldsymbol{c} \rangle}$$

$$\frac{\mathrm{forget}(\mathcal{G}) \vdash K \Rightarrow \kappa}{\mathcal{G} \vdash \mathbf{type}\ X :: K \Rightarrow \exists(\delta :: \kappa, \varepsilon)\,.\,\langle \ell_X : \{= \delta :: \kappa\}, \varepsilon \rangle}$$

$$\frac{\mathrm{forget}(\mathcal{G}) \vdash T :: \kappa \Rightarrow \tau \qquad \mathcal{G} \vdash \tau \rightsquigarrow \hat{\tau}}{\mathcal{G} \vdash \mathbf{val}\ X : T \Rightarrow \exists(\varepsilon, z : \hat{\tau})\,.\,\langle \ell_X : \{\tau\}, \varepsilon \rangle}$$

$$\frac{\mathrm{forget}(\mathcal{G}) \vdash T :: \kappa \Rightarrow \tau}{\mathcal{G} \vdash \mathbf{type}\ X = T : \exists(\varepsilon, \varepsilon)\,.\,\langle \ell_X : \{= \tau :: \kappa\}, \varepsilon \rangle}$$

$$\frac{\mathcal{G} \vdash E : \tau \rightsquigarrow e}{\mathcal{G} \vdash \mathbf{val}\ X = E : \exists(\varepsilon, \varepsilon)\,.\,\langle \ell_X : \{\tau \rightsquigarrow z\}, \mathbf{val}\ z = e \rangle}$$

型が本質的に未確定なことを表す内部表現に変換し，引数由来の変数名たち γ_1 もここで仮置き用に生成

$$\frac{\mathcal{G} \vdash S_1 \Rightarrow \exists(\Delta_1, \gamma_1)\,.\,\mathcal{I}_1 \qquad \mathcal{G}, \Delta_1, \gamma_1, X : \mathcal{I}_1 \vdash M_2 : \exists(\hat{A}_2, \gamma_2)\,.\,\langle \mathcal{I}_2, \boldsymbol{c} \rangle}{\mathcal{G} \vdash \mathbf{fun}(X : S_1) \to M_2 : \exists(\varepsilon, \varepsilon)\,.\,\langle \forall(\Delta_1, \gamma_1)\,.\,\mathcal{I}_1 \to \exists(\hat{A}_2, \gamma_2)\,.\,\langle \mathcal{I}_2, \boldsymbol{c} \rangle, \varepsilon \rangle}$$

$$\frac{\mathcal{G} \vdash M : \exists(\hat{A}, \gamma)\,.\,\langle \mathcal{I}, \boldsymbol{c} \rangle}{\mathcal{G} \vdash M : \exists(\hat{A}, \gamma)\,.\,\langle \mathcal{I}, \boldsymbol{c} \rangle}$$

$$\frac{\mathcal{G}(X_1) = \forall(\Delta_1, \gamma_1)\,.\,\mathcal{I}_{11} \to \exists(\hat{A}_2, \gamma_2)\,.\,\langle \mathcal{I}_{12}, \boldsymbol{c} \rangle \qquad \mathcal{G}(X_2) = \mathcal{I}_2 \qquad \mathcal{I}_2 \succ \mathcal{I}'_2 \qquad \forall(\Delta_1, \gamma_1)\,.\,\mathcal{I}_{11} \to \exists(\hat{A}_2, \gamma_2)\,.\,\langle \mathcal{I}_{12}, \boldsymbol{c} \rangle \geq^{\mathrm{inst}} \mathcal{I}'_2 \to \exists(\hat{A}', \gamma')\,.\,\langle \mathcal{I}', \boldsymbol{c}' \rangle}{\mathcal{G} \vdash X_1\,(X_2) : \exists(\hat{A}', \gamma')\,.\,\langle \mathcal{I}', \boldsymbol{c}' \rangle}$$

$$\frac{\mathcal{G}(X_1) = \mathcal{I}_1 \qquad \mathrm{forget}(\mathcal{G}) \vdash S_2 \Rightarrow \exists A_2\,.\,\Sigma_2 \qquad \mathcal{I}_1 \succ \mathcal{I}'_1 \qquad \mathcal{G} \vdash \mathcal{I}'_1 \leqslant \exists A_2\,.\,\Sigma_2 \rightsquigarrow \exists \hat{A}_2\,.\,\mathcal{I}_2}{\mathcal{G} \vdash (X_1 :> S_2) : \exists(\hat{A}_2, \varepsilon)\,.\,\langle \mathcal{I}_2, \varepsilon \rangle}$$

抽象化用の内部表現に変換

部分型判定をしつつ，変数名や実体型を $\mathcal{I}'_1$ から $\mathcal{I}_2$ や $\hat{A}_2$ へとコピー

示したい正当性の定理

$\Gamma \vdash M : \exists A\,.\,\Sigma$（静的解釈なしで型がつく）かつ $\mathrm{forget}(\mathcal{G}) = \Gamma$ かつ $\vdash^{\mathrm{wf}} \mathcal{G}$ ならば，或る $\hat{A}, \boldsymbol{\gamma}, \mathcal{I}, \boldsymbol{c}$ が存在して以下が成り立つ：

- $\mathcal{G} \vdash M : \exists(\hat{A}, \boldsymbol{\gamma})\,.\,\langle \mathcal{I}, \boldsymbol{c} \rangle$（静的解釈ありでも型がつく）
- $\mathrm{forget}(\mathcal{I}) = \Sigma$ かつ $\mathcal{G} \vdash^{\mathrm{wf}} \mathcal{I}$（シグネチャが元と同形で well-formed）
- $\mathrm{forget}(\hat{A}) = A$　　• $\mathrm{target}(\mathcal{G}) \vdash \boldsymbol{c} : \boldsymbol{\gamma}$（変換結果に適切な型がつく）