必要呼び高階関数型言語のコンパイラの形式的検証

水野雅之

大学院情報科学研究科
住井・松田研究室

2017年2月17日

背景：コンパイラを形式的に検証する意義

コンパイラのバグ

生成されるコードに影響
デバッグが困難

$\Downarrow$

形式的検証が有用

背景: コンパイラの形式的検証の既存研究

- **値呼び・高階関数あり**
  - CompCert [Leroy+ 2006]
    - C **コンパイラ**
- **値呼び・高階関数なし**
  - Lambda Tamer [Chlipala 2007]
    - **単純型付き λ 計算のコンパイラ**
  - CakeML [Kumar+ 2014]
    - SML **のコンパイラ**

本研究の概要

- 必要呼び高階関数型言語のコンパイラを Coq で検証
  - De Buijn インデックスを採用
    - 束縛の表現が簡潔に
  - 小ステップ意味論による定式化 [Ariola+ 1995] を採用
    - 束縛の対応を保つのが容易

名前による束縛の表現の問題点

- 項の等価性が Coq の等価性でなく α 等価性となる

$$\lambda x.\lambda y.x \overset{\alpha}{=} \lambda a.\lambda b.a$$

- Capture を避ける必要がある

$$\begin{aligned}&[y \mapsto {\color{red}x}](\lambda x.y)\\ \overset{\alpha}{\neq}\ &\lambda {\color{red}x}.{\color{red}x}\end{aligned}$$

名前による束縛の表現の問題点

- 項の等価性が Coq の等価性でなく α 等価性となる

$$\lambda x.\lambda y.x \overset{\alpha}{=} \lambda a.\lambda b.a$$

- Capture を避ける必要がある

$$\begin{aligned}&[y \mapsto \textcolor{red}{x}](\lambda x.y)\\ \overset{\alpha}{=}\ &\lambda z.\textcolor{red}{x}\end{aligned}$$

De Bruijn インデックスによる束縛の表現

- 内側から数えて何番目の束縛かで表現

$$\lambda x.x\ (\lambda y.x\ y) \Rightarrow \lambda.0\ (\lambda.1\ 0)$$

- α 等価な式は文面上も同じ

$$\lambda x.\lambda y.x \Rightarrow \lambda.\lambda.1$$
$$\lambda a.\lambda b.a \Rightarrow \lambda.\lambda.1$$

必要呼び高階関数型言語の操作的意味論

- 抽象機械によるもの
  - STG machine [Jones 1992]
- 対象言語の構文でヒープを表現
  - 大ステップ意味論を採用
    - [Launchbury 1993]
  - 小ステップ意味論を採用
    - [Ariola+ 1995]

$\Rightarrow$ 自明ではない

大ステップ意味論による表現 [Launchbury 1993]

- サンクのポインタを変数で表現

$$\boxed{H; t \Downarrow H'; v}$$

$$\frac{(x \mapsto t_1, H); t_2 \Downarrow H'; v}{H; (\textbf{let}\ x = t_1\ \textbf{in}\ t_2) \Downarrow H'; v}$$

$$\frac{H_2; t \Downarrow H_2'; v}{(H_1, x \mapsto t, H_2); x \Downarrow (H_1, x \mapsto v, H_2'); v}$$

評価の導出例 [Launchbury 1993]

$$\dfrac{\dfrac{\dfrac{\emptyset; 1 \Downarrow \emptyset; 1}{(z \mapsto 1); z \Downarrow (z \mapsto 1); 1}}{\emptyset; \textbf{let}\ z = 1\ \textbf{in}\ z \Downarrow (z \mapsto 1); 1}}{(x \mapsto y, y \mapsto \textbf{let}\ z = 1\ \textbf{in}\ z); y \Downarrow (x \mapsto y, y \mapsto 1, z \mapsto 1); 1}$$

評価の導出例 [Launchbury 1993]

$$\cfrac{\cfrac{\cfrac{\emptyset; 1 \Downarrow \emptyset; 1}{(z \mapsto 1); z \Downarrow (z \mapsto 1); 1}}{\emptyset; \textbf{let } z = 1 \textbf{ in } z \Downarrow (z \mapsto 1); 1}}{(x \mapsto y, y \mapsto \textbf{let } z = 1 \textbf{ in } z); y \Downarrow (x \mapsto y, y \mapsto 1, z \mapsto 1); 1}$$

- ヒープ内の式も自由変数をもつ
- 評価中にヒープの要素が増える
  - 大ステップ意味論なので何個増えるか分からない

評価の導出例 [Launchbury 1993]

$$\cfrac{\cfrac{\cfrac{\emptyset; 1 \Downarrow \emptyset; 1}{(z \mapsto 1); z \Downarrow (z \mapsto 1); 1}}{\emptyset; \mathbf{let}\ z = 1\ \mathbf{in}\ z \Downarrow (z \mapsto 1); 1}}{(x \mapsto y, y \mapsto \mathbf{let}\ z = 1\ \mathbf{in}\ z); y \Downarrow (x \mapsto y, y \mapsto 1, z \mapsto 1); 1}$$

- **ヒープ内の式も自由変数をもつ**
- **評価中にヒープの要素が増える**
  - **大ステップ意味論なので何個増えるか分からない**

$\Rightarrow$De Bruijn index **で束縛の対応を保ちにくい**

小ステップ意味論による表現 [Ariola+ 1995]

$$
\begin{array}{rcl}
v & ::= & \lambda x.t \\
a & ::= & v \mid \mathbf{let}\ x = t\ \mathbf{in}\ a
\end{array}
$$

$\boxed{t \longrightarrow t'}$

$$
\begin{array}{rcl}
\mathbf{let}\ x = v\ \mathbf{in}\ E[x] & \longrightarrow & \mathbf{let}\ x = v\ \mathbf{in}\ E[v] \\
(\mathbf{let}\ x = t_1\ \mathbf{in}\ a)\ t_2 & \longrightarrow & \mathbf{let}\ x = t_1\ \mathbf{in}\ a\ t_2
\end{array}
$$

- **評価文脈を活用**
- **簡約で位置が入れ替わる束縛の数は一定**

⇒De Bruijn index **でも束縛の対応を保てる**

Coq による定式化 (1/2)

```
Inductive red_need_body : term -> term -> Prop :=
  | RedNeed_V t1 t1' v2 E :
      value v2 ->
      evaluation_context_need E ->
      t1 = Subst_context E (Var (bindings_context E)) ->
      t1' = Subst_context E (rename (+bindings_context E) v2) ->
      red_need_body (App (Lam t1) v2) (App (Lam t1') v2)
  | RedNeed_C a1 t2 t2' t3 :
      answer a1 ->
      t2' = rename (+1) t2 ->
      red_need_body (App (App (Lam a1) t2) t3) (App (Lam (App a1 t2')) t3)
  | RedNeed_A t1 t1' a2 t3 E :
      answer a2 ->
      evaluation_context_need E ->
      t1 = Subst_context E (Var (bindings_context E)) ->
      t1' = rename (upren (+1)) t1 ->
      red_need_body (App (Lam t1) (App (Lam a2) t3)) (App (Lam (App (Lam t1') a2)) t3).
```

Coq による定式化 (2/2)

```
Lemma answer_or_reducible_or_stuck : forall t,
  answer t ¥/
  (exists E s s', evaluation_context_need E /¥ t = Subst_context E s /¥ red
_need_body s s') ¥/
  (exists E x, evaluation_context_need E /¥ t = Subst_context E (Var x) /¥
bindings_context E <= x).
Proof.
  Local Hint Resolve Peano.le_0_n le_n_S.
  fix 1.
  intros [? | t1 t2 |].
  - right. right. exists CHole. simpl. eauto.
  - remember t1 as t1'.
    destruct t1' as [ | | t11].
    + right. right. exists (CAppL CHole t2). simpl. eauto.
    + destruct (answer_or_reducible_or_stuck t1) as [Hanswer1 | [[E1 [? [?
[? [Hsub1]]]]] | [E1 [? [? [Hsub1]]]]]]; subst.
      * inversion Hanswer1; subst; clear Hanswer1.
        right. left. exists CHole. simpl. eauto 7.
      * right. left. exists (CAppL E1 t2). rewrite Hsub1 in *. simpl. eauto
 7.
      * right. right. exists (CAppL E1 t2). rewrite Hsub1 in *. simpl. eaut
o 7.
```

まとめ

- 必要呼び高階関数型言語の意味を Coq で定式化できた
  - De Bruijn index を必要呼びにも適用、束縛の取り扱いを簡単化
  - 簡約の決定性程度は証明済
    - Ariola らの意味論の side condition の欠如を発見
- 必要呼び高階関数型言語のコンパイラを Coq で検証
  - 短期的には名前呼びとの対応を証明