平成 27 年度 卒業研究発表会

MinCaml の K 正規化の形式的検証

B2TB2512 水野雅之

工学部　情報知能システム総合学科
住井・松田研究室

2016 年 3 月 11 日

研究目的

MinCamlのK正規化の正当性を検証

- 証明を簡潔に
  - 余帰納的大ステップ意味論
  - de Buijnインデックス
- スケーラビリティの確保
  - 証明自動化機能を活用

アウトライン

1. 研究背景
2. MinCaml
3. 束縛の表現
4. 意味論の定義
5. 正当性の検証
6. 結論

アウトライン

既存研究

CompCert [Leroy ら 2005]

- C コンパイラの正当性の検証

[Chlipala 2010]

- 非純粋関数型言語処理系の正当性の検証

アウトライン

MinCaml の特徴

住井による教育用コンパイラ [FDPE 2005]

- OCaml で 2000 行程度
  - 非純粋な関数型言語
  - 型推論
  - 定数畳み込み等の最適化

対象言語

- **高階関数**
- N 引数の構文
- 副作用
- 外部関数呼び出し (入出力)

$$
\begin{array}{l}
M, N, f ::= \\
\qquad \vdots \\
\qquad \textbf{let rec } x\ y_1\ \cdots\ y_n = M \textbf{ in } N \\
\qquad f\ N_1\ \cdots\ N_n \\
\qquad (M_1,\ \cdots\ , M_n) \\
\qquad \textbf{let } (M_1,\ \cdots\ , M_n) = M \textbf{ in } N \\
\qquad \textbf{Array.create } M\ N \\
\qquad M_1.(M_2) \\
\qquad M_1.(M_2) \leftarrow M_3
\end{array}
$$

対象言語

- 高階関数
- N 引数の構文
- 副作用
- 外部関数呼び出し (入出力)

$$
\begin{aligned}
M, N, f ::= & \\
& \vdots \\
& \textbf{let rec } x\ y_1\ \cdots\ y_n = M \textbf{ in } N \\
& f\ N_1\ \cdots\ N_n \\
& (M_1,\ \cdots\ , M_n) \\
& \textbf{let } (M_1,\ \cdots\ , M_n) = M \textbf{ in } N \\
& \textbf{Array.create } M\ N \\
& M_1.(M_2) \\
& M_1.(M_2) \leftarrow M_3
\end{aligned}
$$

対象言語

- 高階関数
- N 引数の構文
- 副作用
- 外部関数呼び出し (入出力)

$$
\begin{aligned}
M, N, f ::= \quad & \\
& \vdots \\
& \textbf{let rec } x\ y_1\ \cdots\ y_n = M \textbf{ in } N \\
& f\ N_1\ \cdots\ N_n \\
& (M_1,\ \cdots\ , M_n) \\
& \textbf{let } (M_1,\ \cdots\ , M_n) = M \textbf{ in } N \\
& \textbf{Array.create } M\ N \\
& M_1.(M_2) \\
& M_1.(M_2) \leftarrow M_3
\end{aligned}
$$

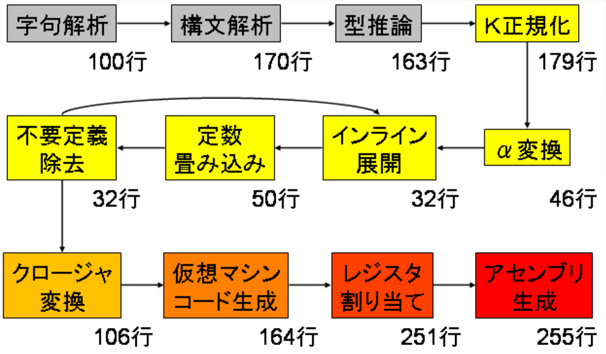
対象言語

- 高階関数
- N 引数の構文
- 副作用
- 外部関数呼び出し (入出力)

$$
\begin{aligned}
M, N, f ::= \\
& \vdots \\
& \mathbf{let\ rec}\ x\ y_1\ \cdots\ y_n = M\ \mathbf{in}\ N \\
& f\ N_1\ \cdots\ N_n \\
& (M_1,\ \cdots\ , M_n) \\
& \mathbf{let}\ (M_1,\ \cdots\ , M_n) = M\ \mathbf{in}\ N \\
& \mathbf{Array.create}\ M\ N \\
& M_1.(M_2) \\
& M_1.(M_2) \leftarrow M_3
\end{aligned}
$$

内部設計

- 様々なコンパイルフェーズ
- 疎結合



http://esumii.github.io/min-caml/index8.html

K 正規化

全ての部分式に名前を付ける

$$a + b * c + d$$

$$\begin{aligned}&\mathbf{let}\ x = b * c\ \mathbf{in}\\&\mathbf{let}\ y = a + x\ \mathbf{in}\\&y + d\end{aligned}$$

束縛に関する操作

全ての部分式に名前を付ける

$$a + b * c + d$$

$$\begin{aligned}&\textbf{let } x = b * c \textbf{ in}\\&\textbf{let } y = a + x \textbf{ in}\\&y + d\end{aligned}$$

束縛に関する操作

期待される正当性

K 正規化後の項を評価してみる

$$\lambda x.\lambda y.\lambda z.\ x + y + z$$

$$\begin{array}{l}\lambda x.\lambda y.\lambda z. \\ \textbf{let}\ a = x + y\ \textbf{in} \\ a + z\end{array}$$

定理 1

項 t が値 v に評価される場合、項 t を K 正規化した結果 K(t) は値 v を K 正規化した結果 K(v) に評価される

定理 2

項 t の評価が停止しない場合、項 t を K 正規化した結果 K(t) の評価は停止しない

アウトライン

1. 研究背景
2. MinCaml
3. **束縛の表現**
4. 意味論の定義
5. 正当性の検証
6. 結論

名前による表現

α 等価性の議論が面倒

$$\lambda x.\lambda y.\ x \simeq \lambda a.\lambda b.\ a$$

fresh な名前が必要になる

- 束縛の関係を乱さないよう変数名を選ぶ

$$\begin{aligned}[x \mapsto z](\lambda z.\ x) &\simeq \lambda z'.\ z \\ &\not\simeq \lambda z.\ z\end{aligned}$$

何番目の束縛かで変数を表現

- 内側から外側へ数える

$$\lambda x.\lambda y.\lambda z.\ xz(yz) \qquad \lambda.\lambda.\lambda.\ 2\ 0\ (1\ 0)$$

α 等価な式は構文的に等価

- 名前の freshness を保証

$$\lambda x.\lambda y.\ x$$

$$\lambda a.\lambda b.\ a$$

$$\lambda.\lambda.\ 1$$

シフト

自由変数のインデックスをずらす

$$\uparrow^d t$$

束縛に関する操作

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

$$\begin{array}{l} \textbf{let } a = \lambda x.\ x \textbf{ in} \\ \textbf{let } b = \lambda x.\ y \textbf{ in} \\ a\ b \end{array} \qquad \begin{array}{l} \textbf{let } \_ = \lambda.\ 0 \textbf{ in} \\ \textbf{let } \_ = \uparrow^1 (\lambda.\ 1) \textbf{ in} \\ 1\ 0 \end{array}$$

K 正規化の実装

```
Fixpoint knormal e :=
  match e with
  | Exp.Var x => K.Var x
  | Exp.Abs e => K.Abs (knormal e)
  | Exp.App e1 e2 =>
      K.Let (knormal e1)
        (K.Let (shift 1 (knormal e2))
          (App 1 0))
  end.
```

アウトライン

1. 研究背景
2. MinCaml
3. 束縛の表現
4. **意味論の定義**
5. 正当性の検証
6. 結論

大ステップ意味論

比較的単純な
プログラム変換の検証に適する

が

無限ループとエラーの区別が困難

例:型無しラムダ計算

構文

$$
\begin{aligned}
t \;::=\;& b \\
\mid\;& x \\
\mid\;& \lambda x.\ t \\
\mid\;& t\ t
\end{aligned}
\qquad\qquad
\begin{aligned}
v \;::=\;& b \\
\mid\;& \lambda x.\ t
\end{aligned}
$$

意味論

$$
\frac{}{b \Downarrow b} \qquad\qquad \frac{}{\lambda x.\ t \Downarrow \lambda x.\ t}
$$

$$
\frac{t_1 \Downarrow \lambda x.\ t_0 \qquad t_2 \Downarrow v_2 \qquad [x \mapsto v_2]t_0 \Downarrow v}{t_1\ t_2 \Downarrow v}
$$

大ステップ意味論の問題点

エラー

$$\mathbf{true\ true} \not\Downarrow v$$

適用できる規則が無い

無限ループ

$$(\lambda x.xx)(\lambda x.xx) \not\Downarrow v$$

有限回の規則適用で導出できない

区別できない

余帰納的に定義 [Leroy 2006]

$$\frac{t_1 \Uparrow}{t_1\ t_2 \Uparrow}\qquad\qquad \frac{t_1 \Downarrow v_1 \qquad t_2 \Uparrow}{t_1\ t_2 \Uparrow}$$

$$\frac{t_1 \Downarrow \lambda x.\ t_0 \qquad t_2 \Downarrow v_2 \qquad [x \mapsto v_2]t_0 \Uparrow}{t_1\ t_2 \Uparrow}$$

余帰納的大ステップ意味論 (2/2)

エラー

$$\mathbf{true}\ \mathbf{true} \not\Uparrow$$

適用できる規則がない

無限ループ

$$(\lambda x.xx)(\lambda x.xx) \Uparrow$$

無限回の規則適用を許す

区別できる

参考:入出力を含む言語への拡張

どのような入出力を行ったかを表すラベルを付与

$$\textbf{read\_line}\ ()\Downarrow {}^{\prime\prime}hoge^{\prime\prime}\ \ /\ (\textbf{read\_line}\ () = {}^{\prime\prime}hoge^{\prime\prime})$$

$$\textbf{while true do}\ ()\ \textbf{done}\Uparrow\ /\ \epsilon$$

$$\begin{array}{l}(\textbf{while true do}\\ \textbf{print\_endline}\ {}^{\prime\prime}hoge^{\prime\prime}\\ \textbf{done})\Uparrow\\ /\ \textbf{print\_endline}\ {}^{\prime\prime}hoge^{\prime\prime}\cdots\end{array}$$

アウトライン

コンパイラの検証

言語拡張のたび全ての証明の修正が必要

```
Inductive t :=
  | Var : nat -> t
⋮
Proof.
  intros t.
  induction t.
  Case "Var".
```

```
Inductive t :=
  | Int : Z -> t
  | Var : nat -> t
⋮
Proof.
  intros t.
  induction t.
  Case "Nat".
    ⋮
  Case "Var".
```

構文の違いを自動証明で吸収

$tactic_1; tactic_2$	$tactic_1$ で生じたサブゴール全てに $tactic_2$ を適用
f_equal	$t_1\ t_2 \cdots t_n = t'_1\ t'_2 \cdots t'_n$ から $t_1 = t'_1,\ t_2 = t'_2 \cdots$
solve$[tactic_1 \mid \cdots \mid tactic_n]$	どれかでサブゴールを閉じられるなら閉じる

```
Lemma shift_0 : forall e c,
  shift c 0 e = e.
Proof.
  intros e.
  induction e; intros ?; simpl;
    f_equal;
    solve [ apply shift_var_0 | eauto ].
Qed.
```

拡張性の評価

プリミティブ、if、let を追加

	構文	証明の行数
拡張前	**変数 匿名関数 関数適用**	110
拡張後	20 **種類**	141

高いスケーラビリティを示した

アウトライン

結論

K 正規化を Coq で検証できた

- de Buijn インデックス、余帰納的大ステップ意味論の採用で証明が簡潔に
- 証明自動化による再利用性の高い証明

今後の課題

さらに言語を拡張し、MinCaml と同等に

- 組や複数引数の関数
- 配列
- 外部関数呼び出し

K 正規化以外の検証