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
  - 余帰納的意味論
  - ド・ブラン インデックス
- スケーラビリティの確保
  - 半自動証明

アウトライン

アウトライン

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

関連研究

アウトライン

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

MinCaml

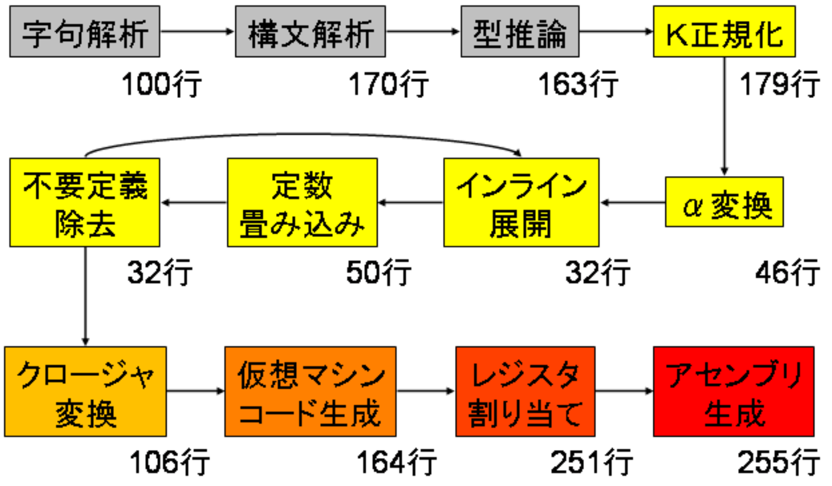
住井による教育用コンパイラ (FDPE 2005)

- 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

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

内部設計




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

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

アウトライン

1. 関連研究
2. MinCaml
3. **意味論の定義**
4. 束縛の表現
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

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

意味論

$$
\frac{}{n \Downarrow n} \qquad \frac{}{\lambda x.\, t \Downarrow \lambda x.\, t}
$$

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

大ステップ意味論の問題点

エラー

$$0\ 0 \not\Downarrow v$$

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

余帰納的定義 (Leroy 2006)

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

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

余帰納的大ステップ意味論 (2/2)

エラー

$$0\ 0 \not\Uparrow$$

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

アウトライン

名前による表現

α 等価性の議論が面倒

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

捕獲が起こりうる

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

ド・ブラン インデックス

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

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

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

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

捕獲も回避できる

シフト

自由変数のインデックスをずらす

$$\uparrow^d t$$

束縛の付け替え

$\lambda x.\ x$	$\lambda.\ 1$
$\lambda x.\lambda y.\ x$	$\lambda.\lambda.\ \uparrow^1 0$ $= \lambda.\lambda.\ 1$

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

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

```
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

- ド・ブラン 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