平成 26 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　電気情報物理工学科
住井・松田研究室

2016 年 3 月 11 日

研究概要

MinCaml の K 正規化を Coq で検証

- 余帰納的意味論
- ド・ブラン インデックス
- 半自動証明

アウトライン

1. **研究背景**
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**

アウトライン

コンパイラのバグは伝播

- UNIX 初期のバックドア

正常な cc　→　異常な cc

正常な login

コンパイラのバグは伝播

- UNIX 初期のバックドア

正常な cc $\rightarrow$ 異常な cc

$\downarrow$

正常な login 異常な cc

コンパイラのバグは伝播

- UNIX 初期のバックドア

正常な cc $\rightarrow$ 異常な cc

$\downarrow$

正常な login $\rightarrow$ 異常な cc $\rightarrow$ 異常な login

現実の処理系

現実の処理系は複雑

OCaml	34 万行
SML#	24 万行
GCC	1500 万行

素朴な検証では破綻

アウトライン

1. 研究背景
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

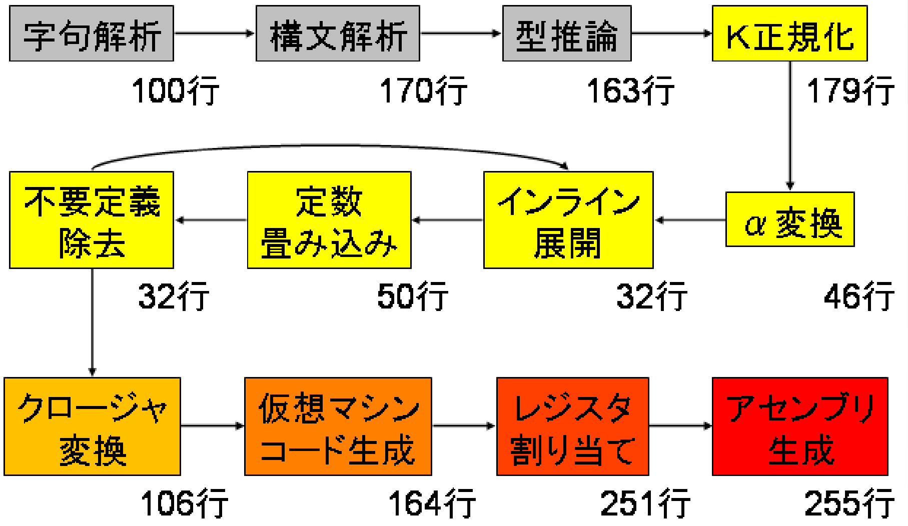
住井による教育用コンパイラ

- 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

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

内部設計


字句解析
100行
構文解析
170行
型推論
163行
K正規化
179行
α変換
46行
インライン展開
32行
定数畳み込み
50行
不要定義除去
32行
クロージャ変換
106行
仮想マシンコード生成
164行
レジスタ割り当て
251行
アセンブリ生成
255行

K 正規化

全ての部分式に名前を付ける

$$a + b * c + d$$

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

束縛の付け替えが伴う

アウトライン

大ステップ意味論

プログラム変換の検証に適する

$$\Updownarrow$$

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

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

意味論

$$
n \Rightarrow n \qquad\qquad \lambda x.\, t \Rightarrow \lambda x.\, t
$$

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

例:型無しラムダ計算

エラー

$$0\ 0 \not\Rightarrow v$$

適用できる規則が無い

無限ループ

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

有限回の規則適用で導出できない

区別できない

余帰納的大ステップ意味論

余帰納的定義

$$\frac{t_1 \overset{\infty}{\Rightarrow}}{t_1\ t_2 \overset{\infty}{\Rightarrow}} \qquad \frac{t_1 \Rightarrow v_1 \quad t_2 \overset{\infty}{\Rightarrow}}{t_1\ t_2 \overset{\infty}{\Rightarrow}}$$

$$\frac{t_1 \Rightarrow \lambda x.\ t_0 \quad t_2 \Rightarrow v_2 \quad [x \mapsto v_2]t_0 \overset{\infty}{\Rightarrow}}{t_1\ t_2 \overset{\infty}{\Rightarrow}}$$

余帰納的大ステップ意味論

エラー

$$0\ 0 \not\overset{\infty}{\Rightarrow}$$

適用できる規則がない

無限ループ

$$(\lambda x.xx)(\lambda x.xx) \overset{\infty}{\Rightarrow}$$

無限回の規則適用を許す

区別できる

アウトライン

1. 研究背景
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

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

$$\begin{aligned}[x \mapsto z](\lambda z.\ x) &= \lambda w.\ z \\ &\neq \lambda z.\ z\end{aligned}$$

ド・ブラン インデックス

何番目の束縛かで変数を表現

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

α 等価な式は文面上も等価

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

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

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

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

$$\lambda x.\ x \qquad \lambda.\ 1$$

$$\lambda x.\lambda y.\ x \qquad \begin{aligned} &\lambda.\lambda.\ \uparrow^1 0 \\ &= \lambda.\lambda.\ 1 \end{aligned}$$

K 正規化の実装

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

アウトライン

言語拡張のたび全ての証明を修正

```
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.
```

言語の拡張

四則演算、if等を追加

	構文の種類	証明の行数
拡張前	3	110
拡張後	20	141

高い再利用性を示した

アウトライン

1. 研究背景
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**

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

- タプルや複数引数の関数
- 配列
- 外部関数呼び出し

K 正規化以外の検証