# CoqからScalaへのコード抽出とその妥当性

田辺, 良則
国立情報学研究所



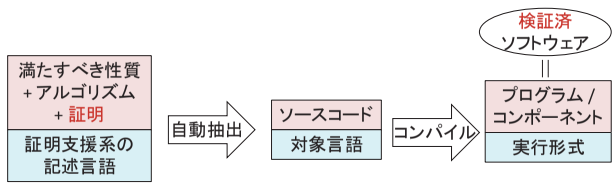
出版情報：MI lecture note series. 61, pp.81-87, 2015-03-06. 九州大学マス・フォア・インダストリ研究所
バージョン：
権利関係：

# CoqからScalaへのコード抽出とその妥当性
# Coq code extraction to Scala and its correctness

TPP2014
2014年12月4日

逸見港* 田辺 良則** 今井 宜洋*** 萩谷 昌己*
Henmi, Ko Tanabe, Yoshinori Imai, Yoshihiro Hagiya, Masami
* 東京大学 ** 国立情報学研究所 *** ITプラニング

## コード抽出

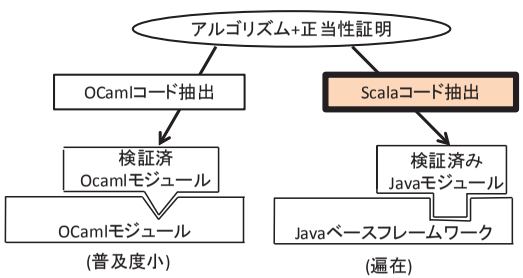



- Coqでの対象言語: OCaml, Haskell, Scheme

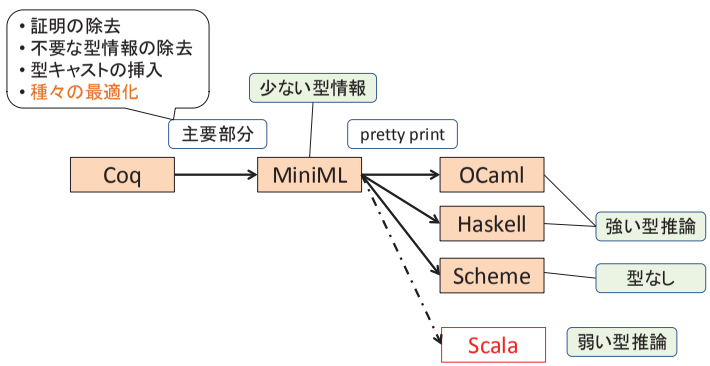
Scalaを対象言語としたコード抽出を実現したい

## モジュールの抽出

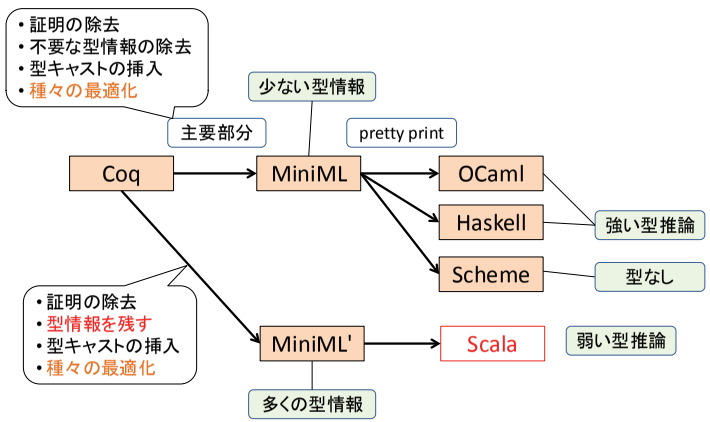
- 全プログラムをCoqで記述するのは非現実的
- キーとなる一部のモジュールだけを検証したい



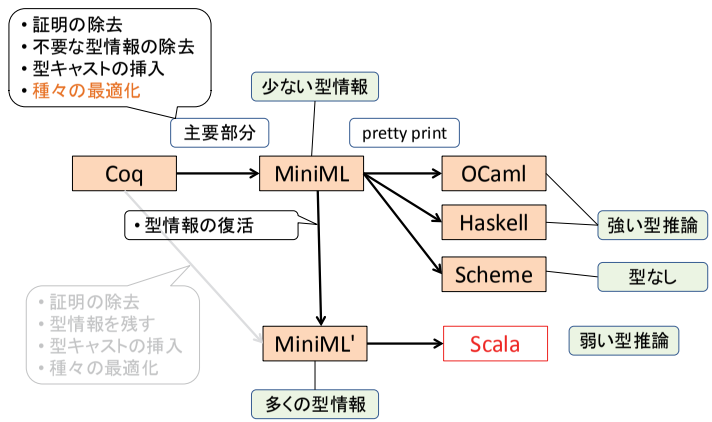

## 既存機能



## Scala抽出 (案1)




## Scala抽出 (案2)




## CoqからMiniMLへの変換

- Coq → MiniML

  e : T の抽出

  1. eをMiniMLの項に変換する
  2. TをMiniMLの型に変換する
  3. 1の項が2の型が持つように型キャストを挿入する

     型推論アルゴリズムM[Lee and Yi 1998]を元にした

     アルゴリズムM'による

     M 推論した型をもとに型検査する

     M' 推論した型をもとに型キャストを挿入する



## MiniMLの構文

MiniMLの項(ml_ast)、MiniMLの型(ml_type)の構文

$\mathrm{ml\_ast}\ e \equiv$

| | |
|---|---|
| $\mid x$ | 変数参照 |
| $\mid e\ e$ | 関数適用 |
| $\mid \mathrm{fun}\ x \to e$ | 関数 |
| $\mid c$ | グローバル参照 |
| $\mid \mathrm{let}\ x = e\ \mathrm{in}\ e$ | let 式 |
| $\mid \mathrm{match}\ e\ \mathrm{with}$ | |
| $\quad c\vec{x} \to e \mid \cdots \mid c\vec{x} \to e$ | match 式 |
| $\mid \mathrm{fix}\ x = e\ \mathrm{with}\ \cdots\ \mathrm{with}$ | |
| $\quad x = e\ \mathrm{for}\ x$ | 再帰関数 |
| $\mid \mathrm{magic}\ e$ | magic 式 |

$\mathrm{ml\_type}\ T \equiv$

| | |
|---|---|
| $\mid \alpha$ | (型変数) |
| $\mid C[\vec{T}]$ | (定数型) |
| $\mid T \to T$ | (関数型) |

$\mathrm{ml\_type\_scheme}\ \tau \equiv$

$\mid \forall\alpha.\tau$

$\mid T$

## 型キャスト

- 例: 下の const1 を抽出する.

```
Definition nat_or_bool : forall b:bool, Set :=
  fun b => if b then nat else bool.

Definition zero_or_false
                : forall b:bool, nat_or_bool b :=
  fun b => if b then 0 else false.

Definition const1 : nat :=
  (zero_or_false true) + 1.
```

## 型キャスト

- Propソートは現れないので, 削除する部分はない.

```
def zero_or_false : Bool -> Any = {
  (b:Bool) => b match {
      case True  => 0()
      case False => False() }}

def const1 : Nat =
  (zero_or_false true) + 1.
```

- しかし, これでは型が合わないので....

## 型キャスト

```
def zero_or_false : Bool -> Any = {
  (b:Bool) => b match {
      case True  => 0().asInstanceOf[Any]
      case False => False().asInstanceOf[Any] }}

def const1 : Nat =
  (zero_or_false.asInstanceOf[Bool->Nat] true) + 1
```

## MiniML では不足する情報

```
def zero_or_false : Bool -> Any = {
  (b:Bool) => b match {
      case True  => 0().asInstanceOf[Any]
      case False => False().asInstanceOf[Any] }}

def const1 : Nat =
  (zero_or_false.asInstanceOf[Bool->Nat] true) + 1
```

## MiniML'

MiniMLの項に型情報を付加した言語MiniML'を考える
MiniML'の項（ml_ast'）からScalaコードを書き出せる

$\mathrm{ml\_ast}'\ e' \equiv$

| | |
|---|---|
| $\mid x[\vec{T}]$ | 変数参照 |
| $\mid e'\ e'$ | 関数適用 |
| $\mid \mathrm{fun}\ x : T \to e'$ | 関数 |
| $\mid c[\vec{T}]$ | グローバル参照 |
| $\mid \mathrm{let}\ x : \tau = e'\ \mathrm{in}\ e'$ | let 式 |
| $\mid \mathrm{match}\ e'\ \mathrm{with}$ $c\vec{x} \to e' \mid \cdots \mid c\vec{x} \to e'$ | match 式 |
| $\mid \mathrm{fix}\ x : T = e'\ \mathrm{with}\ \ldots\ \mathrm{with}$ $x : T = e'\mathrm{for}\ x$ | 再帰関数 |
| $\mid \mathrm{magic}\ (e', T)$ | magic 式 |

MiniML'の型（ml_type)は
MiniMLの型と共通

$\mathrm{ml\_type}\ T \equiv$

| | |
|---|---|
| $\mid \alpha$ | (型変数) |
| $\mid C[\vec{T}]$ | (定数型) |
| $\mid T \to T$ | (関数型) |

$\mathrm{ml\_type\_scheme}\ \tau \equiv$

| | |
|---|---|
| $\mid \forall\alpha.\tau$ | |
| $\mid T$ | |



## アルゴリズムM''

推論した型をもとに型情報を挿入した項を返す
　M 推論した型をもとに型検査する
　M' 推論した型をもとに型キャストを挿入する

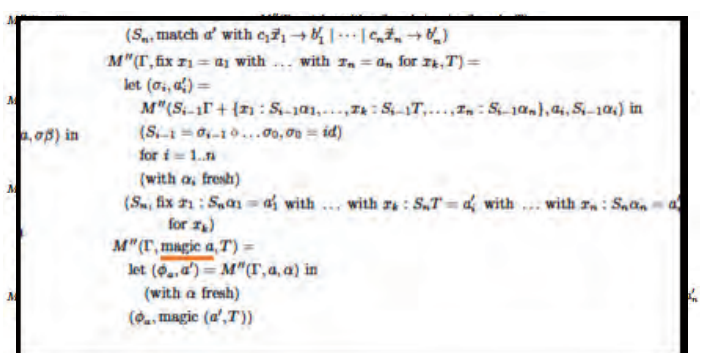
M'' : env × ml_ast × ml_type → subst × ml_ast'
　型環境　　代入
M''(Γ, e, T) を実行すると
Γの下で e : T であるために必要な型変数への代入を返す
同時に e に型情報を付加したMiniML'の項を返す



$$
\begin{aligned}
&M''(\Gamma, x, T) = \\
&\quad \text{let } \sigma = unify(T, Inst(\vec{b}, \Gamma(x))) \text{ in} \\
&\quad\quad \text{(with } \vec{b} \text{ fresh)} \\
&\quad (\sigma, x[\sigma\vec{b}]) \\
&M''(\Gamma, a\ b, T) = \\
&\quad \text{let } (\phi_a, a') = M''(\Gamma, a, \alpha \to T) \text{ in} \\
&\quad \text{let } (\phi_b, b') = M''(\phi_a\Gamma, b, \phi_a\alpha) \text{ in} \\
&\quad\quad \text{(with } \alpha \text{ fresh)} \\
&\quad (\phi_a \circ \phi_b, a'\ b') \\
&M''(\Gamma, \mathrm{fun}\ x \to a, T) = \\
&\quad \text{let } \sigma = unify(T, \alpha \to \beta) \text{ in} \\
&\quad\quad \text{(with } \alpha \text{ and } \beta \text{ fresh)} \\
&\quad \text{let } (\phi_a, a') = M''(\sigma\Gamma + \{x : \sigma\alpha\}, a, \sigma\beta) \text{ in} \\
&\quad \text{let } \phi = \sigma \circ \phi_a \text{ in} \\
&\quad (\phi, \mathrm{fun}\ x : \phi\alpha \to a') \\
&M''(\Gamma, c, T) = \\
&\quad \text{let } \sigma = unify(T, Inst(\vec{b}, E(c))) \text{ in} \\
&\quad\quad \text{(with } \vec{b} \text{ fresh)} \\
&\quad (\sigma, c[\sigma\vec{b}]) \\
&M''(\Gamma, \text{let } x = a \text{ in } b, T) = \\
&\quad \text{let } (\phi_a, a') = M''(\Gamma, a, \alpha) \text{ in} \\
&\quad\quad \text{(with } \alpha \text{ fresh)} \\
&\quad \text{let } \tau_a = Gen_{\phi_a\Gamma}(\phi_a\alpha) \text{ in} \\
&\quad \text{let } (\phi_b, b') = M''(\phi_a\Gamma + \{x : \tau_a\}, b, T) \text{ in} \\
&\quad (\phi_a \circ \phi_b, \text{let } x : \phi_b\tau_a = a' \text{ in } b')
\end{aligned}
$$

$$
\begin{aligned}
&M''(\Gamma, \mathrm{match}\ a\ \mathrm{with}\ c_1\vec{x}_1 \to b_1 \mid \cdots \mid c_n\vec{x}_n \to b_n, T) = \\
&\quad \text{let } (\phi_a, a') = M''(\Gamma, a, \tau_a) \text{ in} \\
&\quad\quad \text{(with } \tau_a \text{ fresh)} \\
&\quad \text{let } (\sigma_i, b_i') = \\
&\quad\quad M''(S_{i-1}\Gamma + \{x_{i1} : S_{i-1}Inst(\vec{\beta}, T_{i1}), \ldots, x_{ik_i} : S_{i-1}Inst(\vec{\beta}, T_{ik_i}), b_i, S_{i-1}T) \text{ in} \\
&\quad\quad (S_{i-1} = \sigma_{i-1} \circ \cdots \circ \sigma_0 \circ \phi_a, \sigma_0 = id) \\
&\quad\quad \text{(with } E(c_i) = \forall\vec{a}.(T_{i1} \to \cdots \to T_{ik_i} \to I[\vec{a}]), \vec{\beta} \text{ fresh)} \\
&\quad\quad \text{for } i = 1..n \\
&\quad (S_n, \mathrm{match}\ a'\ \mathrm{with}\ c_1\vec{x}_1 \to b_1' \mid \cdots \mid c_n\vec{x}_n \to b_n') \\
&M''(\Gamma, \mathrm{fix}\ x_1 = a_1\ \mathrm{with}\ \ldots\ \mathrm{with}\ x_n = a_n\ \mathrm{for}\ x_k, T) = \\
&\quad \text{let } (\sigma_i, a_i') = \\
&\quad\quad M''(S_{i-1}\Gamma + \{x_1 : S_{i-1}\alpha_1, \ldots, x_k : S_{i-1}T, \ldots, x_n : S_{i-1}\alpha_n\}, a_i, S_{i-1}\alpha_i) \text{ in} \\
&\quad\quad (S_{i-1} = \sigma_{i-1} \circ \ldots \sigma_0, \sigma_0 = id) \\
&\quad\quad \text{for } i = 1..n \\
&\quad\quad \text{(with } \alpha_i \text{ fresh)} \\
&\quad (S_n, \mathrm{fix}\ x_1 : S_n\alpha_1 = a_1'\ \mathrm{with}\ \ldots\ \mathrm{with}\ x_k : S_nT = a_k'\ \mathrm{with}\ \ldots\ \mathrm{with}\ x_n : S_n\alpha_n = a_n' \\
&\quad\quad \text{for } x_k) \\
&M''(\Gamma, \mathrm{magic}\ a, T) = \\
&\quad \text{let } (\phi_a, a') = M''(\Gamma, a, \alpha) \text{ in} \\
&\quad\quad \text{(with } \alpha \text{ fresh)} \\
&\quad (\phi_a, \mathrm{magic}\ (a', T))
\end{aligned}
$$






magic式を扱えるようにMを拡張

## M''の性質

定理1
任意のMiniMLの項と型 e, T に対して、|- e : T ならば
M''([], e, T) は失敗せずにMiniML'の項e' を返して
|- e' : T
が成り立つ。

証明
Mの健全性、完全性の証明の拡張

17

## コード抽出の妥当性

抽出後のコードが
- 型の妥当性
  型検査を通る
- 計算の妥当性
  抽出前のコードと抽出後のコードが,「同じ計算」を実行する.

18

## 型の妥当性

[Letouzey, 2004]

標準のコード抽出は型の妥当性を満たす

定理2
任意のCoq上の定義 e : T から抽出した
MiniMLの項 $e_{MiniML}$ と型 $T_{MiniML}$ は
|- $e_{MiniML}$ : $T_{MiniML}$
を満たす。

19

## 型の妥当性

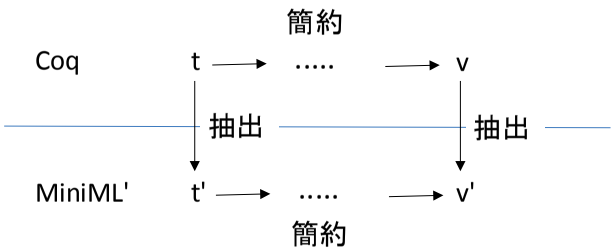
本研究の方式によって抽出されるコードは型の妥当性を満たす

定理3
任意のCoq上の定義 e : T からこの方式によって得た MiniML'の項 e' と型 T' は
|- e' : T'
を満たす。
証明: 定理1, 2 の組み合わせ

20

## 実行の妥当性

- 抽出したコードが，抽出前のコードと「同じ計算」を実行する．




## 型キャストの問題

- 実行の妥当性の証明は，OCaml の場合を真似するだけではない部分がある．
- OCaml では，型キャストに用いる Obj.magic はエラーを起こさない(実行時は no-op)．

```
(Obj.magic true) + 1
```

ここは通る　ここでエラー

- Scala では，asInstanceOf[] 自体がエラーを起こす．

```
True().asInstanceOf[Nat] + 1
```

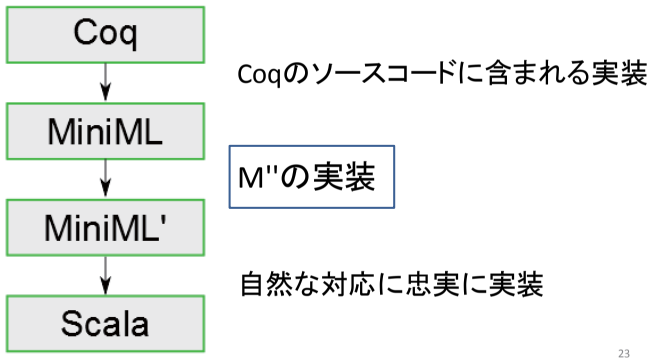
BoolとNatが異なる帰納型なのでエラー

- 下のようになる例があると破綻する

```
True().asInstanceOf[Nat].asInstanceOf[Bool]  &&  b
```

## 実装

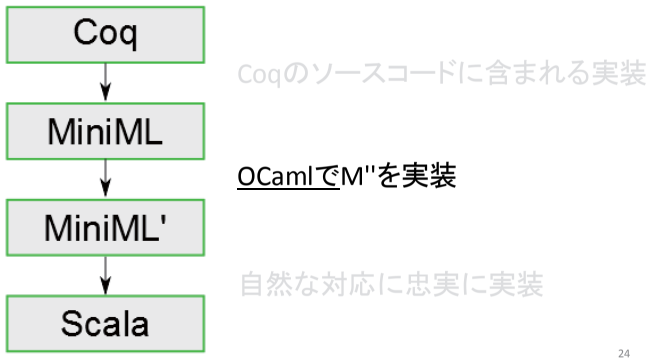
CoqからScalaへのコード抽出プログラム (OCaml)




23

## 実装1

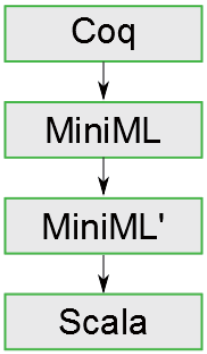
実装済み




24

## 実装2

実装途中




Coqのソースコードに含まれる実装

Coq上でアルゴリズムM''の実装と
定理1の証明(4000行)をした後、
OCamlへ抽出して利用

自然な対応に忠実に実装

25

## 先行研究

W [Damas and Milner, 1982]の形式的証明
(Mのもととなった型推論アルゴリズム)

- Coq
  - [C. Dubois and V. Menissier-Morain, 1999]
  - [J. Garrigue, 2010]
- Isabelle/HOL
  - [D. Nazareth and T. Nipkow, 1999]
  - [C. Urban and T. Nipkow, 2009]

26

## Coq上のM''の実装

今回はCoq上で証明して終わりではなく
抽出後のコードを考慮する必要がある

- アルゴリズムの失敗の表現
- 新しい型変数の生成

27

## まとめ

- MiniMLの項に型の情報を付加するアルゴリズムM''を定義し、Scalaへのコード抽出を設計，実装
- Coq上でM''の定義と定理1の証明を実装
- 今後の課題
  - 計算の妥当性を満たすことの証明
  - 実際のコードへの適用
  - Scalaへのコード抽出全体の正当性が保証された実装

28