PHOAS を応用した Rocq 上での代数的効果の形式化手法の検討と証明の自動化

井本有哉　池渕未来

京都大学

背景　代数的効果とハンドラを証明支援計上で形式化したい

代数的効果とハンドラ：例外処理機構の一般化として，副作用を統一的に扱う仕組み

```
handle (
  let x ← random ()
  in raise x
) with (
  raise → ...
  random → ...
)
```

操作：副作用を伴う「impure な関数」
　ほかイベント管理，メモリ操作など

ハンドラ：操作の処理をになう
　プログラムの再開制御も可能

```
handle (
  let x ← random ()
  in raise x
) with (
  random → ...
)
```

raise が呼ばれると
実行時エラー

全ての操作は，必ずいずれかのハンドラで処理される

実行時エラーを減らすため，
「閉じた」プログラムだけ認めたい

cf. 型による操作の管理：
この類のエラーを全て検知できるが，複雑なので OCaml 5.0 では取り入れられず

既存研究　Parametric HOAS (PHOAS) [Chlipala, 2008]

HOAS のアイデア：**メタ言語**（本研究では Rocq）の
　変数管理機構を利用し，
　対象言語の open な項を弾く
PHOAS：HOAS を Rocq 上で定義可能に

```
fun x → x + 1
```

```
fun x → y + 1
```

下側のプログラムは
定義不可能とする
（＝**静的に**エラー検知）

```
Inductive term : Type := ...
| Lam : (term → term) → term.
```

HOAS
（Rocq 上で定義不可能）

PHOAS

```
Inductive term' v : Type := ...
| Lam :
  (v → term' v) → term' v.
Definition term := ∀v, term' v.
```

メタ言語の機能を利用するので，**実装負担も軽減される**

貢献 1　PHOAS による操作の管理

アイデア：Rocq の変数管理機構を，
操作とハンドラの対応にも利用できないか？

ある操作の外側に，その操作を処理できるハンドラがないならば，そのプログラムは定義不可能とする

```
Inductive term'  {var : ty → Type} {ops : baseTy → baseTy → Type}
: ty → Type := ...
| HandleWith : forall {t A B t'},
                 (ops A B → term' t)
                → (var t → term' t')
                → (var (Base A) → var (Fun (Base B) t') → term' t')
                 → term' t'.
Definition term (t: ty) : Type :=
    forall var ops, term' var ops t.
```

処理される操作を引数とし，
ハンドルされる項を返す関数

return 文に
対する処理

操作に
対する処理

```
handle (
  let x ← random ()
  in raise x
) with (
  random → ...
)
```

raise に対応するハンドラがなく，このプログラムは定義不可能とする

貢献 2　提案した設計の妥当性の証明

OCaml 5.0 のように，操作の管理を標準的に実装した言語を定義
さらに，提案した言語から，上記の言語への変換を定義

対応するハンドラの
有無は確かめない

定理：提案した言語における任意のプログラムに対し，
上記の変換を施す前後で意味論は等しい

```
Theorem semantics_preservation :
  forall t eff
  (e: opPHOAS.term t eff),
opNormal.denote (translate e)
 = opPHOAS.denote e.
```

自動化により，言語の変更に対し，証明の変更が軽微となることを確認

展望

提案した設計では実行時エラーを完全には防げない．静的検査をより強くできないか？