Haskell MPTC, FunDeps and type improvement

OCaml-Nagoya
今井敬吾

2008.10.22

(ほぼ, これの解説資料です⇒)

[1] Martin Sulzmann, Gregory J Duck, Simon Peyton-Jones, Peter J Stuckey. Understanding functional dependencies via constraint handling rules.
J. Funct. Program., Vol. 17, No. 1. (January 2007), pp. 83-129.

型クラス(type classes)おさらい

1. オーバーロード (多重定義)

例 : (==) 演算子を様々な型で定義したい... !

```
data Fruit =
  Apple | Orange -- 型Fruit のデータは Apple と Orange

class Eq a where -- 型クラス Eq
  (==) :: a -> a -> Bool

instance Eq Fruit where -- FruitはEqのインスタンス
  Apple == Apple = True
  Orange == Orange = True
  _ == _ = False
```

多重定義と多相性

2. 型変数の動く範囲を限定する

例：要素のメンバシップ判定 elem

型変数 a を Eq のインスタンスに限定

```
elem :: Eq a => a -> [a] -> Bool
elem x (y:ys) = if x==y then True else elem x ys
elem _ []     = False
```

==演算子が使える

アドホック多相 … 型がプログラムの振る舞いに影響する (cf. パラメトリック多相)

MPTC
(Multi-parameter type classes)

- 複数の型引数を取る型クラス
- 例 : コレクション型の操作を一般化

```
class Coll c e where
  empty :: c
  insert :: c -> e -> c

instance Coll [a] a where
  empty = []
  insert x l = x:l
```

MPTC の問題点

- 推論される型が一般的すぎて使えない
- 例： 2つの要素を一度に insert する ins2...

```
:t let ins2 xs x y = insert (insert xs x) y in ins2
let .. ins2 :: (Coll c e1, Coll c e) => c -> e -> e1 -> c
```

挿入する2つの要素の型が違う

MPTC の問題点

- インスタンスが意図通りに解決されず不便

```
:t insert empty 'a' :: [Char]
<interactive>:1:7:
    No instance for (Coll [Char] e)
      arising from a use of `empty' at <interactive>:1:7-11
    Possible fix: add an instance declaration for (Coll [Char] e)
    In the first argument of `insert', namely `empty'
```

- ここでは empty :: Coll [Char] e => [Char] が要求されるが そんなインスタンスは無い。

ない？

- (前ページ) empty :: Coll [Char] e => [Char] が要求されるが そんなインスタンスは無い。

```
instance Coll [a] a where
  empty = []
  insert x l = x:l
```

これは？

➡型変数 e は自由であり Char=e と 推論できない

FunDeps (functional dependencies)

- MPTC の引数間に 関数的な依存制約を導入する
- MPTC の引数リストの後に書く

```
class Coll c e | c->e where
  empty :: c
  insert :: c -> e -> c
```

cが決まればeが決まる

参照：http://www.haskell.org/haskellwiki/Functional_dependencies

FunDepsの例

- Coll は コレクションの型が決まれば要素の型が決まるはず (c->e)

```
class Coll c e | c->e where
  ....
instance Coll [a] a where
  empty = []
  insert x l = x:l
```

cが決まればeが決まる

[a]の要素型は a
(関数的依存)

FunDeps は型をimproveする

- 以前の empty の型

  Coll [Char] e => [Char] は

  クラス宣言のFunDepsにより

  Coll [Char] Char => [Char] となる

  instance Coll [a] a にマッチするため

  文脈から Collが消えて empty :: [Char] .

FunDeps は型をimproveする

- ins2の型は無事

  ins2 :: Coll c e => c -> e -> e -> c
  になる。

FunDepsの使いどころ

- Stateモナドの型クラス

  class MonadState m s | m -> s
  (状態モナドが決まると状態も決まる,
  ex. instance MonadState (StateT s) s)

- 線形代数のかけ算

  class Mult a b c | a b -> c where
  mult :: a -> b -> c
  instance Mult Matrix Matrix Matrix where ..
  instance Mult Matrix Vector Vector where ..

Haskell の 型推論系

- 判定 $\Gamma, p \vdash_{inf} (C, t)$ を導出する推論系

- 入力 : Γ, p　出力 : (C, t)

- 型環境 Γ のもとで プログラム p は型 (C, t) をもつ (C は型変数の制約)

[1] Martin Sulzmann, Gregory J Duck, Simon Peyton-Jones, Peter J Stuckey. Understanding functional dependencies via constraint handling rules. J. Funct. Program., Vol. 17, No. 1 (January 2007), pp. 83-129.

関数適用の型推論

e_1 は型 $C_1 \Rightarrow t_1$ と推論された

e_2 は型 $C_2 \Rightarrow t_2$ と推論された

$C_1 \wedge C_2 \wedge t_1 = t_2 \rightarrow a$ が 現在のモジュールPで CHRで 充足可能

$$\frac{\Gamma, e_1 \vdash_{inf} (C_1, t_1) \quad \Gamma, e_2 \vdash_{inf} (C_2, t_2) \quad C' = C_1 \wedge C_2 \wedge (t_1 = t_2 \rightarrow a) \quad a \text{ new} \quad sat(P, C')}{\Gamma, e_1\ e_2 \vdash_{inf} (C', a)} \text{(App)}$$

新しい型変数 a

上が満たされたら

e_1 e_2は型 (C',a) と推論する

型推論の例

簡単な例 (MPTCもFunDepsもなし)：

(I+) :: Num y => y -> y を推論する。

$\Gamma, I \vdash_{inf}$ (Num x, x)

x, y は fresh
(Var規則より)

$\Gamma, (+) \vdash_{inf}$ (Num y, y->y->y)

new a

sat (P, Num x ∧ Num y ∧ y->y->y = x->a)

$\Gamma, (x+) \vdash_{inf}$ (Num y ∧ y->y=a , a)

単純化すると Num y => y->y

sat (P, Num x ∧ Num y ∧ (y->y->y = x->a))

- Num x ∧ Num y ∧ (y->y->y = x->a)

→P (Solve step, [y/x])

Num y ∧ Num y ∧ (y->y->y = x->a)

(∴ Num y ∧ (y->y = a))

ここからCHRの話

- CHR (Constraint Handling Rules) ...
  - 一階述語論理の制約解消系
- 操作的意味論

※ Sultzmanらの定義を示す[7]。本家は(Fruhwirth, 1998)

CHRの構文

- 2種類のルール
  - (Simplification) rule c ⇔ $d_1, \ldots, d_m$
  - (Propagation) rule $c_1, \ldots, c_n \Rightarrow d_1, \ldots, d_m$
- c, c_i は 原子論理式 (ここでは型クラス)
- d_i は 原子論理式 か 等式

CHRの意味
(一階述語論理)

- rule $c \Leftrightarrow d_1, \ldots, d_m$
  - $\forall \tilde{a}. (c \Leftrightarrow \exists \tilde{b}. d_1 \wedge \ldots \wedge d_m)$

- rule $c_1, \ldots, c_n \Rightarrow d_1, \ldots, d_m$
  - $\forall \tilde{a}. (c_1 \wedge \ldots \wedge c_n \Rightarrow \exists \tilde{b}. d_1 \wedge \ldots, \wedge d_m)$

- ここで $\tilde{a}$ は$c_1 \ldots c_n$に出現する自由変数,
  $\tilde{b}$ は $d_1, \ldots, d_m$ にのみ出現する自由変数

CHRの意味 (操作的意味)

- 制約ストア C を rule に従って書き換える。

  $C \rightarrow_R C'$: rule集合R のもとで ストア C を C' に

  - (Simp規則) $C \rightarrow_R C \setminus \{c'\} \cup \theta(\bar{d})$

    ただし rule $c \Leftrightarrow \bar{d} \in R$, $\theta(c)=c'$, $c' \in C$

  - (Prop規則) $C \rightarrow_R C \cup \theta(\bar{d})$

    ただし rule $\bar{c} \Rightarrow \bar{d} \in R$, $\theta(\bar{c})=\bar{c}'$, $\bar{c}' \subseteq C$

  - (Solve規則) $C_u \cup C_e \rightarrow_R \varphi C_u \cup C_e$

    C_uは型クラス制約、C_eは等式、

    φ は C_e の mgu (most general unifier)

CHRの性質

- 停止性、合流性とか
- rule によっては停止しない/合流しない (が、十分条件として coverage condition 等がある)

変換: 型クラス → CHR

- class C => TC a b c d | a b -> c のとき
  - (class CHR)
    rule TC a b c d ⇒ C
  - (functional dependency CHR)
    rule TC a b c_1 d_1, TC a b c_2 d_2 ⇒ c_1=c_2

変換：インスタンス→CHR

- class C => TC a b c d | a b -> c のもとで
  instance C => TC T U V W
- rule TC T U V W ⇔ C
  (Cが空のときは右辺を True に)
- rule TC T U c d ⇒ c=V

時間切れ (T_T

あとはホワイトボードで説明します m(_ _)m