

関数・論理型プログラミング実験

第8回

江口 慎悟
酒寄 健
塚田 武志
松下祐介

講義のサポートページ

<http://www.kb.is.s.u-tokyo.ac.jp/~tsukada/cgi-bin/m/>

- 講義資料等が用意される
- レポートの提出先
- 利用にはアカウントが必要
- 名前/学籍番号/希望アカウント名をメールを
tsukada@kb.is.s.u-tokyo.ac.jp
までメールしてください。
 - 件名は「FL/LP実験アカウント申請」
 - アカウント名/パスワードを返信
 - PCからのメールを受け取れるように

インタプリタを作る（全5回）

第5回 基本的なインタプリタの作成

- 字句解析・構文解析、変数の扱い方

第6回 関数型言語への拡張

- 関数定義・呼び出し機構の作成

第7回 型システムと単純型推論

- 単純型検査器

第8回 単一化、let多相

- 単一化の定義とアルゴリズム、let多相

第9回 様々な拡張

- パターンマッチング

今日の参考資料

- Benjamin C. Pierce:
 Types and Programming Languages,
 The MIT Press, Cambridge, MA, 2002.
- 特に 22章 Type Reconstruction
- 邦訳:
 型システム入門 -プログラミング言語と型の理論-

型システム (前回の復習)

型システム

- 式を「**型**」で分類することで、
プログラムが実行時に不正な動作をしない
ことを検査・保障する仕組み
 - 「**型**」とは評価結果の値の分類
 - 例：整数、真偽値、関数など
- 「式が与えられた型を持つか」の検査を
「型検査」と呼ぶ

ML系言語の型システムの特徴

- 静的 (static)
 - 型検査は実行前に行われる
→ 実行時のオーバーヘッドがない
- 健全 (sound)
 - 型検査が成功したら、そのプログラムは実行時に型エラーを生じない
- 型推論 (type inference)
 - プログラマが型を明示せずとも、コンパイラが適切な型を推論してくれる

单一化

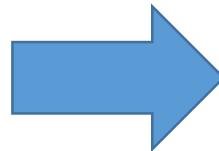
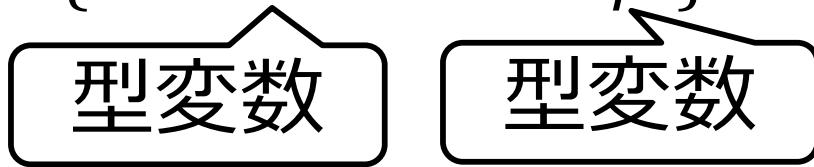
定義

单一化アルゴリズム

单一化

- 型の等式（の集まり）の解を求めるここと

- 例： $\{ \alpha \rightarrow \alpha = \text{Int} \rightarrow \beta \}$



$[\alpha := \text{Int}, \beta := \text{Int}]$

※ ここでは「型を表す式」の間の等式を扱うが
一般的の（一階の）項の間の等式を扱うことができる

单一化の例

- $\text{unify}(\{\alpha=\text{bool}\})$
= [$\alpha := \text{bool}$]
- $\text{unify}(\{\text{bool} \rightarrow \text{bool} = \alpha \rightarrow \beta\})$
= [$\alpha := \text{bool}, \beta := \text{bool}$]
- $\text{unify}(\{\text{bool} = \alpha \rightarrow \beta\})$
 - 単一化不能、エラー

单一化

定義

单一化アルゴリズム

型

- 通常の型に型変数を加えたもの

- 次の構文で与えられる

$$t ::= \text{Int} \mid \text{Bool} \mid t \rightarrow t \mid \alpha$$

型変数

(型) 代入

$T_1, \bar{I}a \vdash$
 $\vdash T_2, \bar{Y}a, \bar{a}$

- 型変数から型への有限のマッピング

$$[\alpha_1 := t_1, \alpha_2 := t_2, \dots, \alpha_n := t_n]$$

- 代入 $\sigma = [\alpha_1 := t_1, \alpha_2 := t_2, \dots, \alpha_n := t_n]$ の型 t への適用を次で定義する

$$\sigma(\alpha) = \begin{cases} t_i & (\text{if } \alpha = \alpha_i) \\ \alpha & (\text{otherwise}) \end{cases} \quad \underbrace{\sigma(\text{Int})}_{\sim\sim} = \text{Int}$$

$$\sigma(t \rightarrow t') = (\sigma(t)) \rightarrow (\sigma(t')) \quad \underbrace{\sigma(\text{Bool})}_{\sim\sim} = \text{Bool}$$

- 代入の合成を $(\sigma \circ \sigma')(\alpha) = \sigma(\sigma'(\alpha))$ で定義

- $[\alpha := t]\beta = \beta$ (if $\alpha \neq \beta$) に注意

制約集合と单一化子

- 制約集合とは $t = t'$ の形の等式の有限集合
- 制約 $C = \{ ty_i = ty_i' \}_{i=1,\dots,n}$ の单一化子とは
$$\sigma(ty_i) = \sigma(ty_i') \quad (i = 1, \dots, n)$$
を満たす代入 σ のこと

最汎単一化子

- 制約集合の单一化子はただ一つとは限らない

例 : $\{ \alpha = \text{Int} \rightarrow \beta \}$ の单一化子は、

$$\sigma_1 = [\alpha := \text{Int} \rightarrow \beta]$$

$$\sigma_2 = [\alpha := \text{Int} \rightarrow \text{Int}, \beta := \text{Int}]$$

$$\sigma_3 = [\alpha := \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}, \beta := \text{Int} \rightarrow \text{Bool}]$$

などがある

- $\exists \gamma. \sigma' = \gamma \circ \sigma$ のとき σ は σ' より一般的 という
 - $\sigma_2 = [\beta := \text{Int}] \circ \sigma_1$ なので σ_1 は σ_2 よりも一般的
- 最も一般的な单一化子を **最汎単一化子** という
 - σ_1 が例の制約集合の最汎単一化子

单一化

定義

单一化アルゴリズム

単一化アルゴリズム

- 与えられた等式制約に対する
最汎单一化子を求めるアルゴリズム
 - 入力：制約集合
 - 出力：最汎单一化子

单一化アルゴリズム

$(ty^*ty) \text{ list} \rightarrow (ty \text{ var}^* ty) \text{ list}$

① 終了時に得たものを
こうえん.

unify {} = []

unify ({ s=s } \cup C) = unify(C)

unify ({ s \rightarrow t = s' \rightarrow t' } \cup C) = unify ({ s=s', t=t' } \cup C)

unify ({ α =t } \cup C) = (unify (C [α := t])) \circ [α :=t]

unify ({ t= α } \cup C) = (unify (C [α := t])) \circ [α :=t]

$s=s$ は同じアドレス
はい。

- ただし、下の2つのケースで t は α を含まない
○ 上記以外は失敗

- Int = Bool や Int = $t \rightarrow t'$ といった形のとき
- $\alpha = t \rightarrow \alpha$ といった形のとき

△ 含んでいない場合は
こう。

let型の

let多相

単純型推論の問題点

解決策：型スキーム

let多相

let多相

単純型推論の問題点

解決策：型スキーム

let多相

単純型推論の問題点

- OCaml のような多相関数が表現できない
 - $\text{fun } x \rightarrow x$ の推論結果は $\alpha \rightarrow \alpha$ だが、 α は未決定な单層型 (OCaml でいう ' $_a$)
- うまくいかない例：
 - let id = fun x -> x in
(id 0, id true)

$\alpha = \text{int}$

$\alpha = \text{bool}$

let多相

単純型推論の問題点

解決策：型スキーム

let多相

解決策

- 型スキームの導入
 - 型スキーム $::= \forall$ 型変数の集合 . 型
 - 式 e が $\forall \alpha_1 \alpha_2 \dots \alpha_n. t$ を持つ



e は任意の型 s_1, \dots, s_n について
 $t [\alpha_1 := s_1, \dots, \alpha_n := s_n]$ という型を持つ

※ 型変数への型スキームの代入は不可

問題点の解決

- id が $\forall \alpha . \alpha \rightarrow \alpha$ を持つとする
 - 使用場所ごとに α を別の型で置き換えられる

($\text{id} \ 0$, $\text{id} \ \text{true}$)

$\alpha = \text{int}$

$\alpha = \text{bool}$

let多相

単純型推論の問題点

解決策：型スキーム

let多相

let多相

- letごとに型スキームに一般化する
 - $\text{let } \text{id} = \text{fun } x \rightarrow x \text{ in } ...$
 - $\text{id}: \alpha \rightarrow \alpha$ を $\text{id}: \forall \alpha. \alpha \rightarrow \alpha$ に置き換える
- 変数の使用ごとに型スキームを型に置換える
 - $(\text{id} 0, \text{id} \text{true})$ 使うタぐで型にする.
 - それぞれの id の出現で、
 $\text{id}: \forall \alpha. \alpha \rightarrow \alpha$ を
 $\text{id}: \beta \rightarrow \beta$ および $\text{id}: \gamma \rightarrow \gamma$ に置き換える

制約の収集 (改)

- 型環境は変数から型スキームへのマップ

(変数から型へのマップをついた)

制約の収集 (改) : let式

subst = (tyvar * ty) list
type-schema = (tyvar list * ty)

○ let $x = e_1 \text{ in } e_2$

- 現在の型環境 Γ で e_1 の型と制約を求める
(型 t_1 、制約 C_1 とする)

- $\sigma = \text{unify } C_1, s_1 = t_1 \sigma, \Delta = \Gamma \sigma$ とする
 - $(\forall \alpha. \alpha \rightarrow \alpha) [\alpha := \text{int}] = \forall \alpha. \alpha \rightarrow \alpha$ に注意
- P を s_1 に現れるが Δ に現れない型変数の集合とする
- 型環境 $\Delta \cup \{ x = \forall P. s_1 \}$ で e_2 の型と制約を求める
(型 t_2 、制約 C_2 とする)
- let式は、型 t_2 、制約 $C_1 \cup C_2$

制約の収集 (改) : 変数

- x
 - 型環境を見ると、 x の型スキームが分かる
(これを $\forall \alpha_1 \dots \alpha_n . t$ とする)
 - 新しい型変数 β_1, \dots, β_n を導入する
 - x の型は $t[\alpha_1 := \beta_1, \dots, \alpha_n := \beta_n]$ 、制約は {}

おまけ：let多相の制限

- 以下は型推論できない

```
(fun f -> (f 0, f true)) (fun x -> x)
```

- let で定義される値は多相にできるが、
関数引数は多相にできない
- 関数引数を多相にするには、
高ランク多相 (rank-2 多相) が必要

例題

理解の確認をするための課題です

課題提出システム上での提出の必要はありません

例題を解きTAに見せることで出席とします

分からぬことがあつたら、積極的に質問しましょう

例題

- 前回の問 2 のインタプリタが扱う型について、その型を表す型 ty を定義せよ
 - 整数型、真偽値型、関数型に加えて、型変数も含めること
 - $ty ::= \text{Int} \mid \text{Bool} \mid ty \rightarrow ty \mid \alpha$
- 型変数を表す型 $tyvar$ も別途定義すると良い

新しい型 ty を
作るかんどうを
作れば? ?

ついでに問 2 も解くことをお勧めします

レポート課題 8

締切：2019/6/25 13:00(JST)

6/25

問 1

○ 次の制約の最汎単一化子は何か
(单一化子が存在しないものもある)

- $\{ \alpha = \text{Int}, \beta = \alpha \rightarrow \alpha \}$
- $\{ \text{Int} = \text{Int} \rightarrow \alpha \}$
- $\{ \text{Int} \rightarrow \text{Int} = \alpha \rightarrow \beta \}$
- $\{ \alpha_1 \rightarrow \alpha_2 = \alpha_2 \rightarrow \alpha_3, \alpha_3 = \beta_1 \rightarrow \beta_2 \}$
- $\{ \alpha \rightarrow \alpha = \beta \rightarrow \gamma, \gamma = \text{Int} \rightarrow \beta \}$

問 2

- 型代入の型を

type subst = (tyvar * ty) list

として定義する。

- 型代入 σ と型 t を受け取り、 $\sigma(t)$ を返す関数

ty_subst : subst -> ty -> ty

を定義せよ

問 3

○代入を合成する関数

compose : subst -> subst -> subst
を定義せよ

compose signal sigma2 は $\sigma_1 \circ \sigma_2$ を返すようにして下さい

問 4

- 単一化を行う関数

ty_unify : (ty * ty) list -> subst
を定義せよ

問 5 (やや難)

- インタプリタを let多相に対応させよ
- 型スキームを表す型 type_schema を定義する
- 次の関数を定義する
 - generalize : type_env -> ty \rightarrow type_schema
↑
型からより型を引いて、
おかれであらわす型を作成する。
 - instantiate : type_schema -> ty \leftarrow 型をいれる。
↑
型へゆくあす
 - get_type_vars : ty -> tyvar list
↑
型変数を集めてくる。
- ※ 上記の方針に従わずともよい

注意

- 問2～4はまとめて提出しても構わない
 - 考察は課題ごとに行うこと
- 最終的に单一化器ができていれば、型定義や補助関数などは問の記述に沿っていなくとも良い

発展 1

- 単一化アルゴリズムの定義における

$$\text{unify } (\{ \alpha=t \} \cup C) = (\text{unify } (C [\alpha:=t])) \circ [\alpha:=t]$$

のケースに付いている条件

※ ただし、 t は α を含まない

を出現検査 (occurrence check) という

- 出現検査はどうして必要か。例を用いて説明せよ。
- 出現検査をしないで单一化することは
どのような型を考えていることに相当するか。

発展 2

- 以下の式の型推論をできるようにせよ

```
let x = fun x -> (x, x) in let2 a0 be3  
let x = fun y -> x (x y) in let1 a4 be3  
let x = fun y -> x (x y) in let1 a5 be3  
let x = fun y -> x (x y) in  
let x = fun y -> x (x y) in  
let x = fun y -> x (x y) in  
  x (fun x -> x)
```

バカテ"かイものと
コレパクトに表現
できよう!...

発展 3

- 講義で紹介した单一化アルゴリズムは、停止性の証明が難しい
 - 構造帰納による定義になつてないため
$$\text{unify } (\{ \alpha=t \} \cup C) = (\text{unify } (C [\alpha:=t])) \circ [\alpha:=t]$$
- 実はデータ型を工夫で構造帰納法にできる
- 下記の論文を読んで、要約し、感想を述べよ
Conor McBride, "First-order unification by structural recursion", *Journal of Functional Programming*, 13(6), 1061–1075, 2003