関数・論理型プログラミング実験
ML演習第6回

松田 一孝
TA: 武田広太郎 寺尾拓

今日の話

- 5/01：簡単な評価器
  - 字句解析・構文解析．簡単な評価器
- 5/13：関数型言語の評価器
  - （高階）関数定義・呼出機構の作成
- 5/20：型システム
  - ML風の型推論の実装
- 5/27：その他拡張
  - 評価規則等

今日の内容

- （単純型の）型推論
  - アルゴリズムW
- 多相型の型推論
  - let多相（ML多相）

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

型システム

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・保証する仕組み
  - （今回は）評価結果の値の分類
    - 整数，真偽値，関数，etc.

- 「式が与えられた型を持つか」の検査を「型検査」と呼ぶ

ML言語の型システム

- 静的（static）
  - 型検査は実行前に行われる
    → 実行時オーバヘッドがない
- 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）
  - 型を明示する必要なし

素朴な試み

- 例について見てみる
  - 定数・組み込み関数
  - letと変数
  - 関数適用と抽象

定数+組み込み関数の場合

- 予め型が決っている
  - 1
    - 1はint型の定数.
      なのでこの式全体はint型
  - true
    - trueはbool型の定数.
      なのでこの式全体はbool型
  - not
    - notはbool -> bool型の定数.
      なのでこの式全体はbool -> bool型

letの場合

- let x = 1 in x + 2
  - 1はint型，なので変数xはint型．
    xをint型とすると，x + 2 はint型．
    よって，let全体もint型

型環境：変数から型へのマッピング

cf. 評価時の環境

変数の型推論

- 型環境をルックアップ
  - {x=int}の下で，xの型はint
  - {x=bool} の下で，xの型はbool
  - {x=bool} の下で，yはエラー

関数の適用

- not true
  - trueはbool型.
    notはbool→bool型.
    なので全体はbool型.

- is_zero 1
  - 1はint型.
    is_zeroは int → bool型.
    なので全体はbool型.

クイズ

- fun x -> x + 1の型は？

問題点：関数抽象

- fun x -> x + 1
  - {x=???}という型環境の下で，
    x+1を型推論するのがよさそう？
  - でも，この時点では???がわからない
  - どうする？

解決

- 型変数の導入と推論の変更
  - 今回紹介する手法
    - 式を走査し，型と型の制約を返す
      - 型変数を適宜導入
      - 例：fun x -> x + 1
        - xの型は型変数α
        - 式全体の結果：型α->intと制約α=int
    - 制約を解き，式の具体的な型を求める

型推論の流れ

- ステップ1：制約の収集
  - 式の型とその型の満たす制約を式の構造にそって求める
    - 例：(fun x -> x + 1)に対し，
      α→int と {α=int} を得る
- ステップ2：制約の解決
  - 制約を解き，式の具体的な型を得る
    - 例：{α=int}を解くと，[α↦int].
      これを α→int に適用し int→int

ステップ1：制約の収集

- 式の構造に従い定義
  - 定数
  - 変数
  - let
  - if
  - 関数抽象
  - 関数適用
  - 再帰関数

制約の収集：定数

- 予め決まった型，空の制約
  - 1
    - int型，制約{}
  - true
    - bool型，制約{}
  - not
    - bool -> bool型，制約{}

制約の収集：変数

- 型環境をルックアップ，空の制約
  - 型環境{x=int}下のx
    - 型int，制約{}
  - 型環境{x=bool}下のx
    - 型bool，制約{}
  - 型環境{x=bool}下のy
    - エラー
      - いわゆる Error: Unbound value y

制約の収集：let式

- let x = e_1 in e_2
  - 現在の型環境でe_1の型と制約を求める
    （それぞれt_1とC_1とする）
  - 現在の型環境にxとt_1の対応を追加し，
    e_2の型と制約を求める
    （それぞれt_2とC_2とする）
  - let式全体の型と制約は
    t_2と$C_1 \cup C_2$

制約の収集：if式

- if e_1 then e_2 else e_3
  - i=1, 2, 3について
    現在の型環境でe_iの型と制約を求める
    （それぞれt_iとC_iとする）
  - if式全体の型と制約は
    t_2と$\{t_1=\mathrm{bool}, t_2=t_3\} \cup C_1 \cup C_2 \cup C_3$

  - 注意：t_1がboolでないからといって
    この時点ではエラーを生じてはダメ
    * fun x -> if x then ... else ...

制約の収集：関数抽象

- fun x -> e
  - 新たな型変数αを導入
  - 現在の型環境にxとαの対応を追加した型環境のもとでeの型と制約を求める（それぞれtとCとする）
  - fun式全体の型と制約は，それぞれ α → t と C

制約の収集：関数適用

- e_1 e_2
  - i=1, 2について
    現在の型環境でe_iの型と制約を求める
    （それぞれt_iとC_iとする）
  - 新たな型変数αを導入する
  - 関数適用式全体の型と制約は
    α と $\{t_1=t_2 \rightarrow \alpha\} \cup C_1 \cup C_2$
  - 注：t_1が$t_2 \rightarrow t$の形か調べるのはNG
    * fun f -> f 1

例

```
fun x -> not x
```

- {x=α}を型環境に追加
  - notの型はbool→bool，制約{}
  - xの型はα，制約{}
  - not xの型はβ，
    制約{(bool→bool)=(α→β)}
- fun x -> not xの
  型はα→β
  制約{(bool→bool)=(α→β)}

制約の収集：再帰関数

- let rec f x = e_1 in e_2
  - 新たな型変数αとβを導入
  - 現在の型環境にfとα→βの対応を追加した型環境をΓとする
  - Γにxとαの対応を追加した型環境の下でe_1の型と制約を求める（それぞれt_1とC_1とする）
  - Γの下でe_2の型と制約を求める（それぞれt_2とC_2とする）
  - 式全体の型と制約は t_2 と $\{t_1 = \beta\} \cup C_1 \cup C_2$

例

```
let rec fact n =
  if n=0 then 1 else n*fact (n-1)
in fact 3
```

- factを$\alpha \to \beta$とする
  - nをαとすると，上のifの部分の型はint，制約は
    $\{\alpha = \text{int},\ \alpha \to \beta = \text{int} \to \gamma,\ \gamma = \text{int}\}$
- fact 3の型はδ，制約は
  $\{\alpha = \text{int},\ \alpha \to \beta = \text{int} \to \gamma,\ \gamma = \text{int},$
  $\alpha \to \beta = \text{int} \to \delta\}$

ステップ2：制約の解決

- 前述のアルゴリズムで求まった
  型と制約に対し，制約を解くことで
  具体的な型を求める
  - 例
    - fun x -> not x に対し
      型 $\alpha \to \beta$ と
      制約{(bool→bool)=($\alpha \to \beta$)}が収集
    - 制約を解くと
      $\alpha = \beta$ =boolとなるので
      fun x -> not xの型は
      bool → bool

単一化（Unification）

- 与えられた等式制約を満たすような変数の置換え方（代入）を求めること
  - ここでの入出力
    - 入力：
      型に関する等式の集合
    - 出力：
      型変数から型へのマッピング（代入）
      - 代入σに対し，σ(x)=xでない要素を並べ，[x1↦σ(x1),…,xn↦σ(xn)]と書く
      - 型tに出現する全ての型変数αをσ(α)で置き換えて得られる型をtσと書く

単一化の例

- unify { α =bool }
  = [$\alpha \mapsto$ bool]
- unify {(bool->bool)=($\alpha \to \beta$) }
  = [$\alpha \mapsto$ bool, $\beta \mapsto$ bool]
- unify { bool = $\alpha \to \beta$ }は失敗
  → 単一化不可
  - 型エラー

単一化アルゴリズム

unify {} = {}
unify ({s=s}∪C) = unify C
unify ({s→t=s'→t'} ∪ C)
 = unify ({s=s', t=t'}∪C)
unify ({α=t}∪C)= unify ({t=α}∪C)
 = unify (C[α↦t])∘[α↦t]
  - ただしtはαを含まない

- 上記以外は失敗
- ∘は代入の合成 (f∘g)x = f (g x)
  - 注意：[α↦t] β = β (if α≠β)

例

- unify $\{(\alpha \to \beta) = (\mathrm{bool} \to \gamma)\}$
  $= \mathrm{unify}\ \{\alpha = \mathrm{bool},\ \beta = \gamma\}$
  $= \mathrm{unify}\ \{\beta = \gamma\} \circ [\alpha \mapsto \mathrm{bool}]$
  $= [\beta \mapsto \gamma] \circ [\alpha \mapsto \mathrm{bool}] = [\alpha \mapsto \mathrm{bool},\ \beta \mapsto \gamma]$

- unify $\{(\alpha \to \beta) = (\mathrm{bool} \to \alpha)\}$
  $= \mathrm{unify}\ \{\alpha = \mathrm{bool},\ \beta = \alpha\}$
  $= \mathrm{unify}\ \{\beta = \mathrm{bool}\} \circ [\alpha \mapsto \mathrm{bool}]$
  $= [\beta \mapsto \mathrm{bool}] \circ [\alpha \mapsto \mathrm{bool}] = [\alpha \mapsto \mathrm{bool},\ \beta \mapsto \mathrm{bool}]$

- unify $\{(\alpha \to \beta) = (\beta \to \mathrm{bool})\}$
  $= \mathrm{unify}\ \{\alpha = \beta,\ \beta = \mathrm{bool}\}$
  $= \mathrm{unify}\ \{\beta = \mathrm{bool}\} \circ [\alpha \mapsto \beta]$
  $= [\beta \mapsto \mathrm{bool}] \circ [\alpha \mapsto \beta] = [\alpha \mapsto \mathrm{bool},\ \beta \mapsto \mathrm{bool}]$

まとめ

- ステップ1：制約の収集
  - 式の型とその型の満たす制約を式の構造にそって求める
    - 例：(fun x -> x + 1)に対し，
      $\alpha \to \mathrm{int}$ と $\{\alpha = \mathrm{int}\}$ を得る

- ステップ2：制約の解決
  - 制約を解き，式の具体的な型を得る
    - 例：$\{\alpha = \mathrm{int}\}$を解くと，$[\alpha \mapsto \mathrm{int}]$.
      これを $\alpha \to \mathrm{int}$ に適用し $\mathrm{int} \to \mathrm{int}$

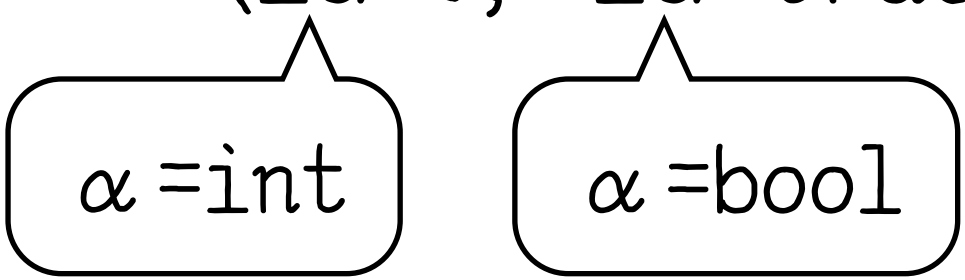
let多相

ここまで推論の問題点

- OCamlのような多相関数を表現不可
  - fun x -> xの推論結果は
    $\alpha \rightarrow \alpha$になるのだが…
    - このαは未決定な単相型（camlでいう'_a）

- うまくいかない例：

```
let id = fun x -> x in
    (id 0, id true)
```

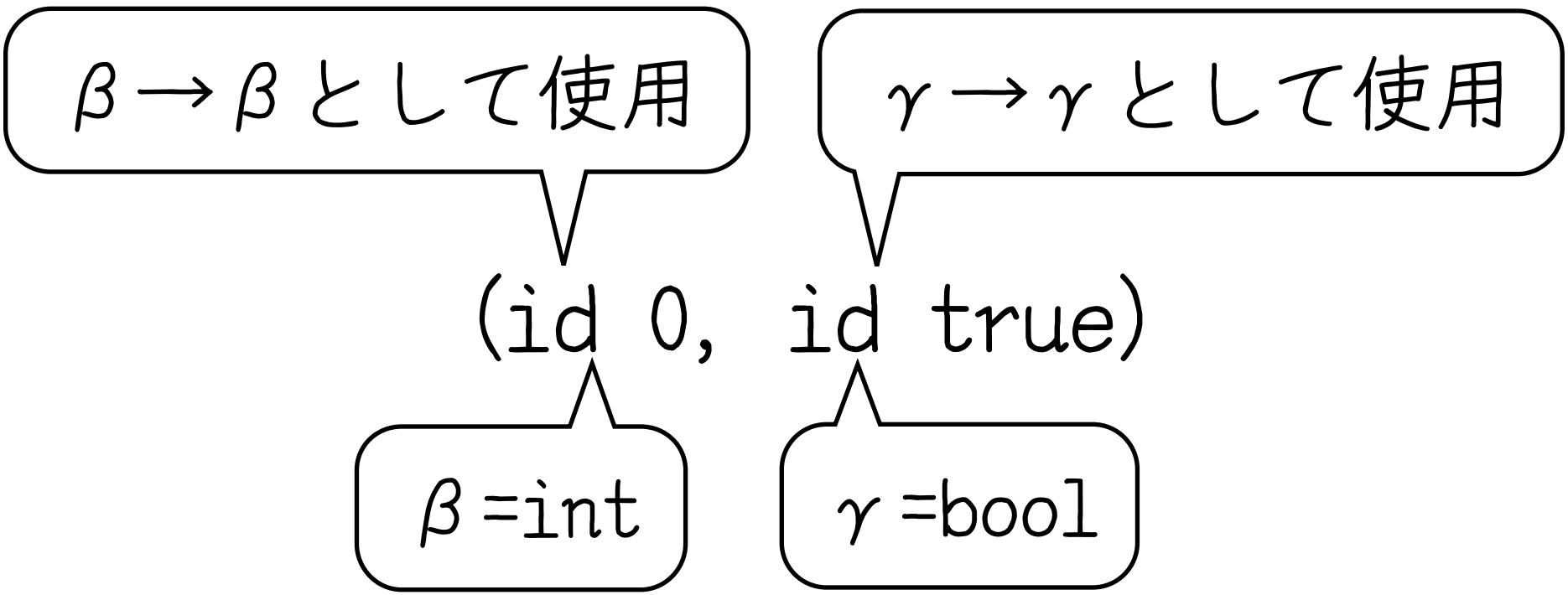


解決案

- 型スキームの導入
  - 型スキーム ::= ∀型変数の集合. 型
  - 「式eが$\forall x_1 \cdots x_n.\ t$を持つ」：
    eは，任意の型$s_1, \cdots, s_n$について，
    $t[x_1 \mapsto s_1, \cdots, x_n \mapsto s_n]$という型を持つ
    - 型変数を型スキームに置き換えることは許されていないことに注意
      - cf. impredicative polymorphism

問題点の解決

- idが$\forall \alpha . \alpha \to \alpha$を持つとする
  - それぞれの使用場所毎に，
    αを新たな型変数に置き換えてよい



let多相

- let毎に型を型スキームに一般化する
  - let id = fun x -> x in ...
    - id: $\alpha \to \alpha$ をid: $\forall \alpha . \alpha \to \alpha$ におきかえる
- そして変数の使用毎に型スキームを型に置き換える
  - (id 0, id true)
  - それぞれのidの出現で,
    id: $\forall \alpha . \alpha \to \alpha$ を
    id: $\beta \to \beta$, id: $\gamma \to \gamma$ に置き換える

制約の収集（改）：let

- let x = e_1 in e_2
  - 現在の型環境Γでe_1の型と制約を求める（それぞれt_1とC_1とする）
  - σ = unify C_1, s1 = $t_1\sigma$ とする
  - $\Delta = \Gamma\ \sigma$
    - $(\forall\alpha.\alpha\to\alpha)[\alpha\mapsto int] = \forall\alpha.\alpha\to\alpha$に注意
  - 型環境 $\Delta \cup \{x = \forall P.s_1\}$の下で，$e_2$の型と制約を求める
    - Pは「s_1に出現する型変数でΔに含まれないもの」
  - この結果が全体の型と制約となる

制約の収集（改）：変数

- x
  - 型環境にxと∀P.tの対応が含まれていたら，xの型と制約は，sと{}
    - ただし，sはt中のそれぞれの型変数のうち，Pに含まれるものを別の新しい型変数に置き換えたもの

let多相の制限

- 以下は型推論できない
  (fun f -> (f 0, f true))
  - 持ちうる型
    - $(\forall \alpha.\, \alpha \to \alpha) \to (\mathrm{int},\ \mathrm{bool})$
    - $\forall \beta.(\forall \alpha.\, \alpha \to \beta) \to (\beta,\ \beta)$
    - …
  - rank-2多相が必要
    - ランク：∀にいたるまでに→の左の子を高々何個たどるか？

letと関数適用・抽象

- let x = e_1 in e_2 と
  (fun x -> e_2) e_1 の違い
  - e_1 ≡ fun y -> y
    e_2 ≡ (x 1, x true)
    とすると
    * 前者はlet多相で型が付く
    * 後者はrank-2多相が必要

第6回レポート課題
締切 6/3 13:00 (JST)

問1

- 前回の課題のインタプリタが扱う値に応じて，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 ::= Int | Bool | ty → ty | α
  - （その後の問の）必要に応じてリストや組の型も定義すること

問2

- 型代入σと型tを受けとり，
  型tσを返す関数ty_substを
  定義せよ
  - ty_subst :
    型代入の型 -> ty -> ty
    - 型代入の型は
      (型変数の型*型を表す型) listでよい
    - 以下の関数を使う？
      - ty_subst_one :
        型変数の型 * ty -> ty -> ty

問3

- 単一化を行う関数ty_unifyを実装せよ
  - ty_unify ：
    型制約の型 -> 型代入の型
    - 型制約の型
      - tyとtyの組のリストでよい

問4

- 前回の課題のインタプリタを拡張し，多相型なしの型推論を実装せよ
  - 以下の関数を実装することになる？
    - gather_constraints :
      型環境の型 -> expr -> ty*型制約の型
    - infer_expr :
      型環境の型 -> expr -> ty
    - infer_cmd :
      型環境の型 -> cmd -> ty*型環境の型

注意

- 再掲：資料で「新たな型変数」と書いてあるところでは，その度ごとに別の型変数を導入するように
  - 副作用を使うと便利か
    - new_ty_var : unit -> 型変数の型
- スコープにも注意
  - (fun x y -> x +
    (fun x -> if x then 1 else 2) y)
    の型はint→bool→int

注意

- 実行例としては
  型推論に成功する例だけでなく，
  しないはずの例についても出すこと
  - （今回の範囲で）型の付かない式の例
    - `fun x -> x x`
    - `fun f -> (f 0 < 1) && f true`

問5

- さらに拡張し，パターンマッチを含む式を型推論をできるようにせよ
  - パターンは値と変数とリストが扱えればよい

制約の収集：match式

- match e with p_1 -> e_1 |…| p_n -> e_n
  - eの型tと制約Cを求める
  - 各iについて
    - p_iの型t_iと制約C_i，追加される型環境Γ_iを求める（後述）
    - 現在の型環境にΓ_iを追加した型環境の下で，e_iの型t_i'と制約C_i'を求める
  - 型変数αを導入
  - match式の型はα，制約は
    $\{t=t_1=\cdots=t_n, \alpha=t_1'=\cdots=t_n'\} \cup C \cup C_1 \cup \cdots \cup C_n \cup C_1' \cup \cdots \cup C_n'$

制約の収集：パターン 1/2

- 型と制約と，追加される型環境を計算
  - 定数パターン 1
    - 型int，制約{}，追加される型環境{}
  - 変数パターン x
    - 型変数αを導入して
    - 型α，制約{}，追加される型環境{x=α}

制約の収集：パターン 2/2

- 型と制約と，追加される型環境を計算
  - ニルパターン []
    - 型変数αを導入
    - 型α list，制約{}，追加される型環境{}
  - コンスパターン p_1 :: p_2
    - p_iの型t_i，制約C_i，追加される型環境Γ_iを求める (i=1, 2)
    - 型変数αを導入
    - 型α list，制約$\{\alpha=t_1, \alpha\ \mathrm{list}=t_2\} \cup C_1 \cup C_2$，追加される型環境$\Gamma_1 \cup \Gamma_2$

ヒント

- 以下の関数を使う？
  - gather_constraints_pattern :
    pattern ->
      ty * 型制約の型 * 型環境の型

発展1

- さらに拡張し，let多相を実現せよ
  - 以下の関数を実装する？
    - generalize :
      型環境の型 -> ty -> ty_scheme
    - instantiate :
      ty_scheme -> ty
      - 副作用を使う
  - 型環境の定義を変更する必要があることに注意
    - 変数から型スキームへのマッピング

発展2（高ランク多相）

- 以下の式が型検査を通るような型システムを実装せよ
  - (fun f -> ((f 1) = 0) && f true)
    (fun x -> x)

    * rank-2多相が必要
      - rank-2多相の型推論は決定可能
      - rank-3以上は一般には決定不能

発展3（再帰型）

- 以下の式が型検査を通るような型システムを実装せよ
  - fun () ->
    (fun x -> x x) (fun x -> x x)
    - 参考：ocamlの-rectypes