計算機科学実験 ソフトウェア 課題3

1029-28-9483 勝田 峻太朗

2018 年 7 月 12 日

目 次

4.2.1[]

textbook を参考にして, 実装した. 型推論の example code の都合上, 課題 2 の実装から, 再帰と複数 let 宣言機能を削除し, 実装を開始した.

4.3.1[]

`ty` を入力とし,`tyvar` の `MySet` を返す関数として実装した. `ty_in` を再帰的に舐めていき,`TyVar(id)` を出力に追加する.

```
(* 与えられた型中の型変数の集合を返す関数 *)
let freevar_ty ty_in =
  let rec loop ty current =
    (match ty with
     | TyVar a -> MySet.insert a current
     | TyFun(a, b) -> MySet.union (loop a current) (loop b current)
     | _ -> current) in
  loop ty_in MySet.empty
```

4.3.2[]

型代入に関する以下の型，関数を typing.ml 中に実装せよ．type subst = (tyvar * ty) list val subst_type : subst -> ty -> ty

Function `subst_type` takes a list of subst `s` and a type `ty` to apply the subst to. This is done by applying `resolve_subst` to every element in `s`, which takes one substitution and applies it to `ty` recursively.

```
type subst = (tyvar * ty) list

(* apply subst:(substutution) to ty:(type) *)
let rec subst_type s ty =
  let rec resolve_subst (subst_tyvar, subst_ty) ty =
    let subst_pair = (subst_tyvar, subst_ty) in
    match ty with
    | TyVar id -> if id = subst_tyvar then subst_ty else TyVar id
    | TyFun(a, b) -> TyFun(resolve_subst subst_pair a, resolve_subst subst_pair b)
    | TyInt -> TyInt
    | TyBool -> TyBool
  in match s with
  | top :: rest ->
    subst_type rest (resolve_subst top ty)
  | [] -> ty
```

4.3.3[]

上の単一化アルゴリズムを `val unify : (ty * ty) list -> subst` として実装せよ．

教科書の資料に従って実装した．

```
(* main unification algorithm *)
let rec unify eqs: (tyvar * ty) list  =
  let rec loop lst current_subst =
    (match lst with
     | (x, y) :: rest ->
       if x = y then loop rest current_subst else
         (match x, y with
          | TyFun(a, b), TyFun(c, d) -> loop ((a, c) :: (b, d) :: rest) current_subst
          | TyVar(id), b ->
            if not (MySet.member id (freevar_ty b)) then
              let mid = unify(subst_eqs (id, b) rest) in
              (id, b):: mid
            else err "unify: could not resolve type"
          | b, TyVar(id) ->
            if not (MySet.member id (freevar_ty b)) then
              let mid = unify(subst_eqs (id, b) rest) in
              (id, b) :: mid
            else err "unify: could not resolve type"
          | _ -> err "unify: could not resolve type"
```

```
        )
    | _ -> current_subst) in
  loop eqs []
```

単一化アルゴリズムの詳細

1. $\mathcal{U}\left(\{(\tau,\tau)\}\cup X'\right)=\mathcal{U}\left(X'\right)$

同じ型がある場合は, 読み飛ばし, 次に進む.

```
if x = y then loop rest current_subst else...
```

2. $U(\{(\tau_{11}\to\tau_{12},\tau_{21}\to\tau_{22})\}\uplus X')=U(\{(\tau_{11},\tau_{21}),(\tau_{12},\tau_{22})\}\uplus X'$

2 つの Fun の入力型と出力型は一致していなければいけない.

```
(match x, y with
 | TyFun(a, b), TyFun(c, d) -> loop ((a, c) :: (b, d) :: rest) current_subst
```

3. $U(\{(\alpha,\tau)\}\cup X')\quad(\text{ if }\tau\neq\alpha)=\begin{cases}U([\alpha\mapsto\tau]X')\circ[\alpha\mapsto\tau] & (\alpha\notin FTV(\tau))\\ error & (\alpha\in FTV(\tau))\end{cases}$

まず,(α,τ) を, 制約を, 残りの型同値 X' に適用し, それを単一化する. その後, 単一化した型代入にこの制約を追加する. $(\alpha\in FTV(\tau)$ の場合にエラーを出力する理由については, 課題 4.3.4 で述べる.

```
(match lst with
 | (x, y) :: rest ->
   if x = y then loop rest current_subst else
     (match x, y with
     ...
        | TyVar(id), b ->
        if not (MySet.member id (freevar_ty b)) then
          let mid = unify(subst_eqs (id, b) rest) in
          (id, b):: mid
        else err "unify: could not resolve type"
        | b, TyVar(id) ->
        if not (MySet.member id (freevar_ty b)) then
          let mid = unify(subst_eqs (id, b) rest) in
          (id, b) :: mid
        else err "unify: could not resolve type"
    ...
```

4.3.4[]

単一化アルゴリズムにおいて，$\alpha\in FTV(\tau)$ という条件はなぜ必要か考察せよ.

fun x -> x x の型推論過程について考えてみる.

まず,ty_exp tyenv FunExp([x], AppExp(x, x)) が実行され, その後,x に新しい TyVar('a とする) を追加した環境 eval_env を用いて, ty_exp eval_env AppExp(x, x) が呼び出される.

ここで,

```
| AppExp(exp1, exp2) ->
    let ty_exp1, tysubst1 = ty_exp tyenv exp1 in
    let ty_exp2, tysubst2 = ty_exp tyenv exp2 in
    (* make new var *)
    let ty_x = TyVar(fresh_tyvar()) in
    let subst_main = unify([ty_exp1, TyFun(ty_exp2, ty_x)] @ eqls_of_subst tysubst1 @ eqls_of_subst tys
    ...
```

以上のコードより,`unify([('a, 'a -> 'b)])` が実行されるが, ここで,$\alpha \in FTV(\tau)$ の条件をチェックしていないと,

```
| TyVar(id), b ->
            if not (MySet.member id (freevar_ty b)) then
              let mid = unify(subst_eqs (id, b) rest) in
              (id, b):: mid
            else err "unify: could not resolve type"
```

このコードの if 分に入ることになるが, これでは,

```
('a, 'a -> 'b) ---> ('a, ('a -> 'b) -> 'b) --->
('a, (('a -> 'b) -> 'b) -> 'b) ---> ('a, ((('a -> 'b) -> 'b) -> 'b) -> 'b)
```

というように無限ループしてしまう.

これは, 型同値のペアの両方に, 同じ型変数が入っていることによるので, これは検出して, 無限ループを防がなければいけない.

4.3.5[]