主専攻実験[S-8] 関数プログラミング

発展課題 7-5

情報科学類 202113564 三村潤之介

関数 tinf1 に、いくつかの式を与え、正しく型推論できていることを確かめよ。

```
let exp = (If(Var("x"), IntLit(2), IntLit(3)))
let exp2 = (If(Var("x"), Var("y"), IntLit(3)))
let exp3 = (If(Var("x"), Var("y"), Var("y")))
let exp4 = (If(Var("x"), Var("x"), Var("y")))
let exp5 = (If(Var("x"), Var("x"), IntLit(3)))
let exp6 = (Plus(Var("x"), IntLit(3)))
let exp7 = (Plus(Var("x"), BoolLit(false)))
```

以上のように、exp 式を定義した。結果は以下のようになった。

```
# tinf1 [] exp;;
- : (string * ty) list * ty = ([("x", TBool)], TInt)
# tinf1 [] exp2;;
- : (string * ty) list * ty = ([("y", TInt); ("x", TBool)], TInt)
# tinf1 [] exp3;;
- : (string * ty) list * ty = ([("y", TVar "'y"); ("x", TBool)],
TVar "'y")
# tinf1 [] exp4;;
- : (string * ty) list * ty = ([("y", TBool); ("x", TBool)],
TBool)
# tinf1 [] exp5;;
Exception: Failure "type error in IF".
# tinf1 [] exp6;;
- : (string * ty) list * ty = ([("x", TInt)], TInt)
# tinf1 [] exp7;;
Exception: Failure "type error in Plus".
```

上記のように substitute を書き換えても、元の subsitute と同じ型と同じ動作をすることを確認しなさい。また、2 つのバージョンで、動作速度はどうなるか(非常に長いリストに対して動かしたり、同じ操作を 1000 回繰返す等をして)調べなさい。

上述の exp 式などに対して実行して、同じ結果が得られることを確認した。
以下のように、1000 個の要素を持つ型環境のダミーを用意した。

```
let rec longenv a b =
  if a > b then []
```

```
  else (string_of_int(a),TVar("'" ^ string_of_int(a))) :: longenv (a+1)
b;;

let thousandtenv = longenv 1 1000
```

また、指定した回数分、substitute を実行する関数を定義した。

```
let rec thousandsub te x =
  if x < 0 then te
  else let tenv = substitute (TVar("'20")) (TBool) (te) in
  thousandsub tenv (x-1)
```

実行速度は以下のような関数を定義して測定する。

```
let time : (unit -> 'a) -> float =
  fun f ->
    let start = Sys.time () in
    let res   = f () in
    let end_  = Sys.time () in
    end_ -. start
```

これらを用いて 1000 個の要素を持つ型環境のダミーに対して、1000 回分の substitute 処理を行った。

List.map を用いた substitute を substitute2 と呼ぶことにする。本来の substitute での実行時間と、substitute2 での実行時間を示す。

```
(substitute)
# let ex1 = time (fun () -> thousandsub thousandtenv 1000);;
val ex1 : float = 0.0716800000000000215

(substitute2)
# let ex1 = time (fun () -> thousandsub thousandtenv 1000);;
val ex1 : float = 0.064728999999999981
```

substitute2 を用いた実行のほうが、実行時間が短くなっていることがわかる。

OCaml の標準ライブラリに含まれる List モジュールには「List.map は tail recursive ではないが、Lisp.rev_map は tail recursive である」と書いてある。「関数が tail recursive である」とはどういう意味であるか調べなさい。

「関数が tail recursive である」とは、その関数は再帰関数であり、かつ、末尾再帰であることを意味する表現である。
再帰関数は、通常、関数の呼び出しの階層が深くなるごとにメモリ領域が消費され、深くなりすぎるとエラー(スタックオーバーフロー)が発生することがある。末尾再帰である関数は、末尾呼び出し最適化により、スタックオーバーフローの発生を防げるという特徴をもつ。これは、内部的に通常のループに置き換えることで実現している。