レポート 3

提出期限:2019/7/19
提出日: 2019 年 7 月 19 日

1029293806　大山 偉永

1　はじめに

このレポートでは計算機科学実験及演習 3 におけるインタプリタ作成実験の型推論実装部分の演習課題に対する説明を与える。セクションは各課題ごとに分かれておりそのセクション内部ではその課題に対する設計方針と実装の詳細/工夫した点の二つの項目に分かれている。

2　Exercise4.2.1

2.1　各プログラムの設計方針

資料のコードをコピーし... の部分を自分で埋める。ty_prim の Mult のパターンなどは Plus のパターンを参考にすれば実装できる。それ以外の Exp の部分も他のすでに埋めてある部分を参考にして同じような共同を示すように埋めていく。

2.2　実装の詳細/工夫した点

ty_exp の IfExp (exp1, exp2, exp3) のパターンの場合はまず exp1 について ty_exp を再帰的に適用してその型を得て、その型が TyBool かどうかで分岐する。もし TyBool でないときはこれは if の文の制約を満たさないのでエラーを返す。exp1 の型が TyBool の時は if A then B else C における B と C の部分の方は一致する必要があるのでそこの型が等しくない時はエラーを返す。等しい時は上の例の式における B の型を返す。
次に LetExp (id, exp1, exp2) の時はまず exp1 の型を今までと同様に ty_prim を再帰的に適用して解析し、id でこの exp1 の型を束縛して tyenv を拡張する。その上で exp2 を ty_prim によって評価する。

3　Exercise4.3.1

3.1　各プログラムの設計方針

図 4.1 中の pp_ty，与えられ た型中の型変数の集合を返す関数で、型は val freevar_ty : ty -> tyvar MySet.t とする関数 freevar_ty を完成させる。freevar_ty は型'a MySet.t は mySet.mli で定義されている'a を要素とする集合を表す型であるのでそれをもとに実装する。

3.2 実装の詳細/工夫した点

pp_ty は string_of_ty を print_string したものなので pp_ty の実装は let pp_ty ty = print_string (string_of_ty ty) とするだけでよい。次に与えられた型中の型変数 (int) のリストを返す関数 freevar_ty についてはその引数がどの ty 型であるかによってパターンマッチを行う。引数が TyFun(a,b) の時は a と b についてそれぞれ freevar_ty を適用して返されたリストを結合する。引数が TyVar a の時はそのまま a をリストに追加して返せばいい。

4 Exercise4.3.2

4.1 各プログラムの設計方針

型代入に関する type subst = (tyvar * ty) list 型、型代入 (のリスト) をひとつの型に適用する val subst_type : subst − > ty -¿ ty 型の関数を typing.ml 中に実装する。

4.2 実装の詳細/工夫した点

型代入を ty に適用する関数を作る。ty の型によってパターンマッチングを行う。ty が TyFun(a,b) の時はこの a と b に再帰的に subst_type を適用しその結果返されたリストを union を用いて結合する。
TyVar theta のときはまず型代入の 1 つ目 (型代入を hd::tl としたときに hd の部分) の型変数 alpha がこの thata と一致するときはこの TyVar theta を型代入の一つ目の型変数 alpha に対応する型で置き換えてその TyVar (theta を alpha に対応する型で置き換えた型) に再び型代入の残りのリストを subst_type を用いて再帰的にこの関数を適用する。この theta が型代入の一つ目の型変数 alpha と異なる場合はその型に残りの型代入 tl を再帰的に適用する。
パターンマッチにおける型が TyFun でも TyVar でもない時は型代入を適用する部分はないのでそのまま与えられた型を返すように実装する。

5 Exercise4.3.3

5.1 各プログラムの設計方針

資料内の単一化に関する部分を読み実装する。そのアルゴリズムについては詳細がすでに書かれているので基本的にはその方針に沿って実装していけばいい。単一化の関数の型は val unify : (ty

* ty) list -> subst である。

5.2 実装の詳細/工夫した点

まず単一化を適用したい型の等式集合でパターンマッチを行いその型の等式集合が空リストの場合はそのまま空リストを返す。つぎにその型の等式集合が (ty1,ty2)::tl のようにかける時、つまり型の等式集合の一つ目の要素が (ty1,ty2) で残りの要素の集合が tl の時は以下の手順を実行する。まず ty1 と ty2 が既に等しい時は何もする必要がないので残りの型の等式集合の部分 tl に再び再帰的にこの単一化の関数 unify を適用する。そうでないときはこの (ty1,ty2) でパターンマッチを行う。

- (ty1,ty2) が (TyFun(ty1,ty2),TyFun(ty3,ty4)) の時はこのファンクションの引数の部分と式部分の型がそれぞれ一致する必要があるので (ty1,ty3) と (ty2,ty4) という等式集合を残りの等式集合 tl に追加して再びこれに unify を適用する。
- (ty1,ty2) が ((TyVar a),ty) の時は ty に含まれる型変数にこの a が含まれているときはエラーを返し、含まれていないときは残りの型の等式集合 tl にこの型代代入 (a,ty) を適用してその等式集合に再びこの関数を適用する。ここで、型代入（のリスト）を一つの型に適用する関数 subst_type は既に定義したが今回は一つの型代入を型の等式集合に適用したいので新しく subst_unify という関数を定義する。これは今書いたように一つの型代入を型の等式集合に適用する関数であり subst_type を使うように工夫することで簡単に定義できる。このように型代入を残りの等式集合 tl に適用したものに再び unify を適用した結果返ってくる型代入（のリスト）に今の型代入 (a,ty) を追加する。
- (ty1,ty2) が (ty,TyVar a) の時は上の場合と逆のことを行えばいい。
- (ty1,ty2) が上記のいずれでもないときはエラーを返す。

以上が単一化のアルゴリズムの実装方式である。

6 Exercise4.3.4

単一化アルゴリズムにおいて，α !∈ FTV(τ) という条件の必要性を考える。たとえば型の等式集合の要素として (α, α-> α) のようなものがあった際これは$\alpha \in$ FTV(τ) であり、$\alpha = \alpha$-> α = (α-> α)->(α-> α)=$\cdots$ のようにこの型の等式集合を解こうとすると無限ループしてしまう。したがってこのようにτに型変数αは含まれないという制約を付ける必要がある。

7 Exercise4.3.5

7.1 各プログラムの設計方針

まずは各型付け規則の型推論手続きを与える。

- 変数 x に対して型付け規則 T-VAR の型推論の手続きを行う。
  1. Γ,x を入力として型推論を行い S と，τ を得る。
  2. S と τ を出力として返す。

- 整数 n に対して型付け規則 T-INT の型推論の手続きを行う。
  1. Γ,n を入力として型推論を行い空の型代入と，TyInt を得る。
  2. $[\,]$ と TyInt を出力として返す。

- 真偽値 b に対して型付け規則 T-BOOL の型推論の手続きを行う。
  1. Γ,b を入力として型推論を行い S_1と，τ を得る。
  2. 型代入 S_1 を $\alpha = \tau$ という形の方程式の集まりとみなして，$S_1 \cup \{(\tau, TyBool)\}$ を単一化し，型代入 S_2 を得る
  3. S_2 と TyBool を出力として返す。

- 式 $e_1 * e_2$ に対して型付け規則 T-MULT の型推論の手続きを行う。
  1. Γ,e_1を入力として型推論を行い S_1と，τ_1 を得る。
  2. Γ,e_2を入力として型推論を行い S_2と，τ_2 を得る。
  3. 型代入 S_1, S_2 を $\alpha = \tau$ という形の方程式の集まりとみなして，$S_1 \cup S_2 \cup \{(\tau_1, TyInt), (\tau_2, TyInt)\}$ を単一化し，型代入 S_3 を得る
  4. S_3 と TyInt を出力として返す。

- 式 $e_1 < e_2$ に対して型付け規則 T-LT の型推論の手続きを行う。
  1. Γ,e_1を入力として型推論を行い S_1と，τ_1 を得る。
  2. Γ,e_2を入力として型推論を行い S_2と，τ_2 を得る。
  3. 型代入 S_1, S_2 を $\alpha = \tau$ という形の方程式の集まりとみなして，$S_1 \cup S_2 \cup \{(\tau_1, TyInt), (\tau_2, TyInt)\}$ を単一化し，型代入 S_3 を得る
  4. S_3 と TyBool を出力として返す。

- if 文 $if\ e_1\ then\ e_2\ else\ e_3$ に対して 型付け規則 T-IF の型推論の手続きを行う。
  1. Γ,e_1を入力として型推論を行い S_1と，τ_1 を得る。
  2. Γ,e_2を入力として型推論を行い S_2と，τ_2 を得る。
  3. Γ,e_3を入力として型推論を行い S_3と，τ_3 を得る。
  4. 型代入 S_1, S_2, S_3 を $\alpha = \tau$ という形の方程式の集まりとみなして，$S_1 \cup S_2 \cup S_3 \cup \{($

$\tau_1, TyBool), (\tau_2, \tau_3)\}$ を単一化し，型代入 S_3 を得る

5. S_3と τ_2 を出力として返す。

- let 文 $let\ x\ =\ e_1\ in\ e_2$ に対して型付け規則 T-LET の型推論の手続きを行う。
  1. Γ,e_1を入力として型推論を行い S_1と，τ_1 を得る。
  2. Γ,x:τ_1,e_2を入力として型推論を行い S_2と，τ_2 を得る。
  3. 型代入 S_1, S_2 を $\alpha\ =\ \tau$ という形の方程式の集まりとみなして，$S_1 \cup\ S_2$を単一化し，型代入 S_3 を得る。
  4. S_3と τ_2 を出力として返す。

- function 式 $fun\ x\ ->\ e$ に対して型付け規則 T-FUN の型推論の手続きを行う。
  1. x の型を τ_1 とする。
  2. Γ,x:τ_1,e を入力として型推論を行い S と，τ_2 を得る。
  3. S と $TyFun(\tau_1, \tau_2)$ を出力として返す。

- 関数の適用式 $e_1\ e_2$ に対して型付け規則 T-APP の型推論の手続きを行う。
  1. Γ,e_1を入力として型推論を行い S_1と，τ_1 を得る。
  2. Γ,e_2を入力として型推論を行い S_2と，τ_2 を得る。
  3. e_1の型を $TyFun(\tau_3, \tau_4)$ とする。
  4. 型代入 S_1, S_2 を $\alpha\ =\ \tau$ という形の方程式の集まりとみなして，$S_1 \cup\ S_2 \cup\ \{(\tau_1, TyFun(\tau_3, \tau_4)), (\tau_2, \tau_3)\}$ を単一化し，型代入 S_3 を得る。
  5. S_3と τ_3 を出力として返す。

以上の型推論の手続きを実際に実装していく。実装については T-PLUS,T-FUN,T-VAR はすでに実装してあるので残りの部分を例に倣って埋めていく方針で行う。

7.2 実装の詳細/工夫した点

- T-MULT,T-LT, は T-PLUS の場合とほぼ同じで ty_prim 関数でその制約と型を返す。その制約の内容は乗算、比較ともにその二項の型は TyInt であるという型代入が必要で返す型は乗算の場合は TyInt、比較の場合は TyBool を返す。
- T-IF の場合は ty_exp 関数の引数が IfExp の時の箇所で実装する。上で記述した型推論の手続きと同様に if 文 if exp1 then exp2 else exp3 における exp1,exp2,exp3 の部分をまず ty_exp で再帰的に評価しその結果返ってきた型 ty1,ty2,ty3 について上と同様の制約を型代入に追加し unify で単一化し、その型代入と型 ty2 を返す。
- T-LET の場合も ty_exp 関数の引数が LetExp の時の箇所で実装する。上で記述した型推論の手続きと同様に let 文 let id = exp1 in exp2 における exp1 部分をまず ty_exp で再帰的に評価しその結果返ってきた型 ty1 で型環境 tyenv を拡張する。また同時に返ってきた型代入 s1 とその拡張した型環境で exp2 を評価しその結果返ってきた型代入 s2 を結合し単一

化を行い exp2 を評価した型 ty2 にこの型代入を適用してその型代入と型を返す。

- T-APP の場合も ty_exp 関数の引数が AppExp の時の箇所で実装する。関数適用式 exp1 exp2 においてまず exp1,exp2 の型を ty_exp で評価し (s1,ty1) と (s2,ty2) を得る。ここでこの ty1 が TyFun 型であるという制約を与えるために fresh_tyvar を用いて新たな型 domty1.domty2 を作り domty1 と ty2 が等しいという型代入と ty1 と TyFun(domty1,domty2) が等しいという型代入を s1 と s2 を結合したものに追加して単一化しその型代入と、domoty2 にその型代入を適用した型 subst_type s4 domty2 を返す。

8 Exercise4.3.6

8.1 各プログラムの設計方針

資料に与えられている let rec 式の型付け基礎に基づいてその型推論部分を実装する。今まで同様にこの型付け規則を忠実に表現する実装を行えば再現できる。

8.2 実装の詳細/工夫した点

let rec 式 let rec id = fun exp1 -> exp2 においてまず id の型に対応する関数の型で型環境を拡張する必要があるがこの時点でこの関数の型の内容は分からないので自分で新たに fresh_tyvar を用いて型変数を二つ用意しその型 domty1,domty2 を一度その id に対する型を TyFun(domty1,domty2) として拡張しその環境を newenv1 とする。また para に関しても同様に今作った型 domty2 で型環境 newenv1 を拡張しこの型環境を newenv2 とする。この newenv1 で exp2 を、newenv2 で exp1 を ty_exp で評価し (s1, ty1) と (s2, ty2) を得る。この s1 と s2 を結合しこれに domty2 と ty1 が等しいという制約を加え単一化しこの型代入と、この型代入を ty2 に適用した型を返す。

9 Exercise4.4.1

9.1 各プログラムの設計方針

多相的 let 式・宣言ともに扱える型推論アル ゴリズムの実装を完成させる。資料内で実装済みの部分はそこを写経する。資料中に作るべき関数などは書いてあるのでまずはその関数を作りそこから対応部分を変えていく方針で実装した。

9.2 実装の詳細/工夫した点

まず関数 freevar_ty の拡張で、型スキーム σ を受け取り、σ に自由に出現する（ただし型環境には出現しない）型変数 の集合を計算する関数である freevar_tysc を実装する。まず型スキームの情報の中にその型を束縛した変数の情報がありその自由変数の集合を lst とする。次にその型スキームに含まれるの型の情報を ty1 として freevar_ty を用いて ty1 に含まれる自由変数を求めその集合を tyvarlst とする。よって求める集合はこの tyvarlst の集合から lst を引いたものである。ここで lst は list 型で求める集合は MySet 型なので MySet.from_list を用いて一度 lst の型を変更しそこでさ集合を求める。

この関数を用いて freevar_tyenv を定義する。この関数は与えられた型環境の中にある、型スキーム内の束縛されていない自由変数の集合を返すものである。型環境をまず list 型に変換するために Environment.ml でこの env 型から list 型に変換する to_list を定義しこの引数の tyenv をリストに変換する。次にそのリストについてパターンマッチを行う。空リストの場合は MySet 型の空リストを返す。型環境が (_,tysc)::tl と表せるときはまずその tysc について先ほど定義した freevar_tysc を用いてその自由変数を求め残りの tl の部分についても同様にこの関数 freevar_tyenv を再帰的に適用する。ここで残りの tl について再帰的にこの関数を適用する際にこの関数の引数は env 型であるためこの list 型の tl は引数にとれないためここで再びこの tl を env 型に変換する関数.from_list を定義した。以上で与えられた定義すべき関数の実装は終わりである。

残りは ty_exp の環境を拡張する部分を型スキームで拡張するように変更する。 LetExp (id, exp1, exp2) の部分は今まで exp1 の型で id を束縛して型環境を拡張していたのを、closure を用いて exp1 の型 τ と型環境 tyenv と exp1 の評価で得た型代入 s1 から、条件「α 1,..., α n は τ に自由に出現する型変数で s1 Γ には自由に出現しない」を満たす型スキーム $\forall\alpha$ 1.… . $\forall\alpha$ n. τ を求めその型スキームで id を束縛し環境を拡張するように変更する。FunExp (id, exp) については関数の自分で作った引数の型 domty1 で id を束縛して型環境を拡張していたが、これを型スキーム TyScheme([],domty) で束縛するようにして拡張するように変更した。以上でこの問の実装は終わりで LetRecExp の多相型に対応する型推論の実装は次の演習問題で述べる。

10 Exercise4.4.2

10.1 各プログラムの設計方針

再帰関数を多層型で型推論できるように拡張する。問題文で型付け規則は与えられており、これにそって型推論できるように実装していく。

10.2　実装の詳細/工夫した点

まず上の問題と同様に環境の拡張を型で拡張するのではなくその型スキームで行うようにする。以下具体的に述べる。LetRecExp(id, para, exp1, exp2) において para の型を fresh_var を用いて domty1 とし同様にして exp1 の型を domty2 とする。このとき多層型での型推論を実装する前はまず id を TyFun(domty1,domty2) で束縛して環境を拡張していたのを、TyScheme([],(TyFun(domty1,domty2)) で束縛して拡張して newenv1 を定義する。para について束縛の拡張も同様で para を domty1 で拡張していたのを TyScheme([],domty1) で束縛して newenv1 を拡張するようにする。この環境を newenv2 とする。

次にこの newenv2 環境下で exp1 を評価し (s1,ty1) を得る。またここで domty2 と ty1 は等しいという制約があるためここで一度これと制約 s1 を結合し単一化し型代入 s5 を得る。次に exp2 の評価のために id を多相的な TyFun(domty1,ty1) で束縛して拡張した環境で評価する必要があるのでこの TyFun(domty1,ty1) に s5 を適用しこの型、型環境 tyenv、型代入 s5 を引数に closure を用いて型スキームを得る。この型スキームで id を束縛し tyenv を拡張しえた newenv3 で exp2 を評価する。こうして得た型代入 s2 を今までの型の制約に追加し単一化し最後に ty2 にこの型代入を適用して得られた型とその型代入を返す。

11　感想

レポート 2 までの課題に依存する課題が多かったためレポート 2 で実装していない部分についての拡張が行えずあまり必須課題以外をすることができなかった。これでしんどかった前期の計算機科学実験が終わると思うと感慨深いものがある。実際 cpu 実験もインタプリタ作成実験も非常に自分のためになりいい経験になったと思うが、自分の勉強時間が奪われたためもう少し分量が少なくてもいいかなとは思った。今実験ではまわりの友達に助けてもらうことが多く、自分を高めてくれる友達の存在には感謝せざる負えない。