計算機科学実験演習３　ソフトウェア　レポート３

氏名：吉村仁志

提出日：７月１９日（金）

第1章　ML^2インタプリタ型推論

1.1　Exercise　4.2.1

chap04-1.pdf で解説されていない部分について解説する。まず、ty_prim についてだが、Plus の時と同様に、受け取った２つの ty に関して op に対して望まれる型であるときに、その op の結果として来るべきである型を返せば良く、そうでない時にはエラーを発生させれば良い。Mult は TyInt と TyInt を受け取って TyInt を返し、Lt は TyInt と TyInt を受け取って TyBool を返す。といった具合である。AAND、OOR は引数も結果もすべて TyBool である。次に、ty_exp について説明する。IfExp についてだが、型の条件としては、exp1 の型が TyBool であり、exp2 と exp1 の型が等しいということである。そこで、まず tyarg1 に ty_exp を再帰呼出しして exp1 の型を束縛し、match 文でそれが TyBool 出ない時はエラーを発生させる。TyBool であるもとで、exp2 と exp1 の型を評価し、それが等しいかどうかを match 文で評価し、そうでない時はエラーを発生させる。等しいというもとで、最終的には exp2 もしくは exp3 の型が結果になるので、それを返すようにする。LetInExp は、まず id を exp1 の型の評価結果で束縛し、それで現在の環境を拡張した新たな環境 tynewenv を作成し、そのもとで exp2 の型を評価すれば良い。

第2章　ML^3インタプリタ型推論

2.1　Exercise　4.3.1

まず pp_ty の実装についてだが、これは string_of_ty を呼び出すだけと定義したので、string_of_ty について解説する。これは、ty 型の値を受け取ってそれを文字列にするような関数である。match 構文を用いて型によって場合分けする。まず TyInt、TyBool のときはそれぞれ'int'、'bool' をそのまま返せば良い。TyFun(ty1, ty2) のときは、ty1、ty2 をそれぞれ再び評価しなければならなく、'TyFun(' (string_of_ty ty1) (string_of_ty ty) ')'といったように中身について string_of_ty を再帰呼出しをする。TyList ty のときは、ty を再び評価するので、(string_of_ty ty) 'list' というようにする。TyVar num のときは、その num の中身を探しに行くことになる。そこで、すでに定義された TyVar num を格納しておくものが必要であり、syntax.ml にグローバル変数としてリストの参照 tyvarlist * tyvar * id を定義しておく。これは、tyvar とそれを表す文字を対応させている。そこで、TyVar num のときは、tyvarlist を走査して num に対応する文字を返すようにする。また、リストを走査し、num を発見するための関数 researchlist を定義してそれを利用した。
次に fresh_tyvar の実装だが、まず型変数が現れうるのは、TyVar num、TyFun、TyList である。そこで、match 構文を利用してそれぞれの型に関して処理を行う。TyVar num のときは num は欲しいものそのものであるから、MySet.ml で定義されている関数 insert を利用して、insert num MySet.empty とした MySet 型の集合を作り、それを返す。TyFun(ty1, ty2) のときは、ty1、ty2 のそれぞれを評価する必要があるので、ty1、ty2 を再帰的に評価しその結果を統合すれば良い。よって、MySet.ml で定義されている関数 union を利用して、union (fresh_tyvar ty1) (fresh_tyvar ty2) を返せば良い。TyList ty のときは ty を評価する必要があり、fresh_tyvar ty を返す。それ以外の型のときは MySet.empty を返す。

2.2　Exercise　4.3.2

subst_type は、(tyvar * ty) list と ty を受け取って (tyvar * ty) list を ty に適用するのだが、ここでそのための補助関数として、sub_subst を定義する。これは、(tyvar * ty) と ty を受け取って (tyvar * ty) を ty に適用しその

結果を返すような関数である。これは match 構文を利用して、それぞれの型に関して処理を行う。TyInt、TyBool のときはそのまま TyInt、TyBool を返せば良い。TyFun(ty1, ty2) 、TyList ty のときは前者は ty1、ty2 に関して、後者は ty に関して再帰呼出しする必要があるので、TyFun(sub_subst ... ty1, sub_subst ... ty2) 、TyList sub_subst ... ty をそれぞれ返せば良い。TyVar num のときは、num が引数として受け取った tyvar と一致するときには型を置き換える必要がある。sub_subst の引数を (numm, tyy) ty とすると、num = numm のときは、tyy を返し、そうでないときはそのまま Tyvar num を返すといったようにする。これで sub_subst の実装は終了する。subst_type であるが、これは受け取った第一引数である、(tyvar * ty) list の一要素毎に sub_subst を呼びだせば良い。これで subst_type の実装は終了する。

2.3 Exercise 4.3.3

まず、型 ty の中に TyVar num が現れるかどうかを判定するための関数 reserarch_ftv を定義する。これは num と ty を受け取って、bool 型の値を返すような関数である。存在するとき true を返し、そうでないとき false を返す。受け取る引数を、numm ty とし、match 構文で ty の型によってそれぞれ処理を行う。TyVar num のときは、numm = num を返せば良い。TvFun(ty1, ty2) のときは、ty1、ty2 のそれぞれを評価する必要があり、ty1、ty2 のどちらかが true のときには全体の結果も true になるので、(research_ftv ty1) || (research_ftv ty2) を返せば良い。TyList ty のときは、ty を評価すれば良く、research_ftv ty を返せば良い。それ以外のときは false を返す。これで research_ftv の実装は終了する。次に、型代入のリスト numtyl と、ty のタプルのリスト tytyl を受け取って、後者に前者を適用するような関数 map_subst を定義する。これは、tytyl が (ty1, ty2) :: rest という形になるとき、ty1、ty2 のそれぞれに型代入を適用すればよく、このとき ((subst_type numtyl ty1), (subst_type numtyl ty2)) :: (map_subst rest) を返すようにする。tytyl が空リストのときは空リストを返す。次に unify について解説する。受け取る引数は tylist であるとする。解説にあるとおり、tylist の型によって場合分けをする。tylist が空リストのときは空リストを返せば良い。そうでない時、すなわち (t1 , t2) :: rest にマッチしたとする。t1 = t2 のときは条件は満たされており、unify rest を返せば良い。以下そうでない時について考える。t1、t2 がそれぞれ TyFun(tyy11, tyy12)、TyFun(tyy21, tyy22) のとき、条件は tyy11 = tyy21、tyy12 = tyy 22 であるので、解説にあるとおり、unify ((tyy11, tyy21) :: (tyy12, tyy22) :: rest) を返す。t1、t2 がそれぞれ TyVar num 、tyy のとき、research_ftv num tyy を評価して、解説にあるとおり、これが true のときはエラーを発生させる。false のとき、解説にあるとおり、 (num, tyy) は答えの型代入の一要素になり、また残りにその型代入

を行ったリストを再帰的に評価するので、先ほど定義した map_subst を利用して、(num, tyy) :: (unify (map_subst [(num, tyy)] rest)) を返す。t1、t2 がそれぞれ tyy、TyVar num のときも同様である。t1、t2 がそれぞれ TyList ty1、TyList ty2 のときは、ty1 と ty2 が等しい必要があるので、それを rest に含めたリストを再帰的に評価すればよく、unify ((ty1, ty2) :: rest) を返せば良い。これで unify の実装は完了する。

2.4 Exercise 4.3.4

unify の第一引数のリストに含まれる (TyVar num, tyy) について、num が tyy に真に含まれるとき、まずこれが必ず等しくならないことは明らかである。(方程式 a = a + b (ただし、b != 0) が成り立たないのと原理的に同じである。) よって、型代入に (num, tyy) を含めてしまうと、tyy の中の num を tyy で置き換える、という操作を型代入が起きるたびに繰り返され、また型代入ではエラーが起きることはないので、間違った結果が返ってくる。

2.5 Exercise 4.3.5

IfExp(exp1, exp2, exp3) について型推論の手続きを与えると以下のようになる。

1. Γ、$exp1$ を入力として型推論を行い、$S1$、$\tau 1$ を得る。 (2.1)
2. Γ、$exp2$ を入力として型推論を行い、$S2$、$\tau 2$ を得る。 (2.2)
3. Γ、$exp3$ を入力として型推論を行い、$S3$、$\tau 3$ を得る。 (2.3)
4. 型代入 $S1, S2, S3$ を$\alpha = \tau$という方程式の集まりとみなして、
   $S1 \cup S2 \cup S3 \cup (\tau 1, bool) \cup (\tau 2, \tau 3)$ を単一化し、型代入 $S4$ を得る。 (2.4)
5. $S4$ と$\tau 2$ を出力として返す。 (2.5)

次に LetInExp(id, exp1, exp2) についての型推論の手続きを与えると以下のようになる。

1. $\Gamma, exp1$ を入力として型推論を行い、$S1, \tau 1$ を得、
   id を$\tau 1$ でΓを拡張した新しい環境 $N\Gamma$を作る。 (2.6)
2. $N\Gamma, exp2$ を入力として型推論を行い、$S2, \tau 2$ を得る。 (2.7)
3. 型代入 $S1, S2,$ を$\alpha = \tau$という方程式の集まりとみなして、
   $S1 \cup S2 \cup$ を単一化し、型代入 $S3$ を得る。 (2.8)

4. $S3$ と$\tau 2$ を出力として返す。 (2.9)

次に FunExp(id, exp) についての型推論の手続きを与えると以下のようになる。

1. Γ を入力とし、新しい型変数$\tau 1$ を作り、Γを、
   $\tau 1$ に束縛された id で拡張した新しい環境 $N\Gamma$を作る。 (2.10)

2. $N\Gamma$、exp を入力とし、型推論を行い、$S1$, $\tau 2$ を得る。 (2.11)

3. $\tau 1$ に $S1$ の型代入を適用してできた新たな型、$\tau 3$ を得た後、
   $S1, TyFun(\tau 3, \tau 2)$ を返す。 (2.12)

次に、AppExp(exp1, exp2) についての型推論の手続きを与えると以下のようになる。

1. $\Gamma, exp1$ を入力とし、型推論を行い、$S1$, $\tau 1$ を得る。 (2.13)

2. $\Gamma, exp2$ を入力とし、型推論を行い、$S2, \tau 2$ を得る。 (2.14)

3. $\tau 1$ の型が $TyFun(tyy1, tyy2)$ のとき、手続き 4 へ、
   $TyVar\ num$ のときは手続き 5 へ
   どちらでもない時はエラー発生 (2.15)

4. $S1 \cup S2 \cup (tyy1, \tau 2)$ を単一化し、型代入 $S3$ を得、
   $S3, tyy2$ を出力として返す。 (2.16)

5. 新しい型変数$\tau 3$ を得、$S1 \cup S2 \cup (TyVar\ num,\ TyFun(\tau 2, \tau 3))$
   を単一化し、型代入 $S4$ を得、$S4, \tau 3$ を出力として返す。 (2.17)

次に、補助関数について解説する。ひとつ目は、型代入を型の等式集合に変換する関数 eqs_of_subst である。これは、リストを走査しながら (num, ty) を (Tyvar num, ty) に全てを変換すれば良いだけである。また、解説 pdf では補助関数 subst_eqs を定義しているが、これは map_subst そのものである。次に、先ほどの型推論の実装であるが、基本的には手続き通りに実装すれば良い。ただし、単一化をする際の入力となるリストは、先ほどの補助関数 eqs_of_subst を利用して型代入のリストの型を変換してから unify に渡す必要があることに注意する。AppExp に関して少し補足すると、ty1 の形として許されるのは、TyFun、TyVar num の 2 つである。TyVar num についてだが、ty1 はこの段階で初めて関数型であることが決まり、最終的に返す型はまだ確定していないので、新しい型変数を作る必要がある。

2.6 Exercise 4.3.6

let rec 式の型推論の手続きを与えると以下のようになる。
LetRecExp(id, para, exp1, exp2) を処理することを考える。

1. 新しい型変数 $\tau 1$、$\tau 2$ を得、新しい型変数$\tau 3$ を $TyFun(\tau 1, \tau 2)$ で束縛する。 (2.18)

2. Γ を、$\tau 1$ で束縛された $para$、$TyFun(\tau 1, \tau 2)$ で束縛された id で拡張した新しい環境 $N\Gamma$を得る。 (2.19)

3. $N\Gamma$、$exp1$ を入力として、型推論を行い、$S1$、$\tau 4$ を得る。 (2.20)

4. $N\Gamma$、$exp2$ を入力として、型推論を行い、$S2$、$\tau 5$ を得る。 (2.21)

5. $S1 \cup S2 \cup (\tau 2, \tau 4)$ を単一化し、型代入 $S3$ を得る。 (2.22)

6. $S3$、$exp2$ を出力として返す。 (2.23)

実装は手続き通りに行えばよく、特に注意するべきこともない。

2.7 Exercise 4.3.7

リストの型推論の手続きを与えると以下のようになる。ListExp(exp1, exp2) を処理することを考える。

1. Γ、$exp1$ を入力として、型推論を行い、$S1$、$\tau 1$ を得る。 (2.24)

2. Γ、$exp2$ を入力として、型推論を行い、$S2$、$\tau 2$ を得る。 (2.25)

3. $\tau 2$ が $TyList\ ty1$　のとき手続き 4 へ、
そうでないときは手続き 5 へ (2.26)

4. $S1 \cup S2 \cup (ty1, \tau 1)$ を単一化し、型代入 $S3$ を得、
$S3$、$ty2$ を出力として返す。 (2.27)

5. $S1 \cup S2 \cup (TyList\ \tau 1, \tau 2)$ を単一化し、型代入 $S4$ を得、
$S4$、$TyList\ \tau 1$ を出力として返す。 (2.28)

手続き 3 の場合分けについてだが、リストの後続がリスト型か、型変数であるかで場合分けしている。型変数である場合は、この段階でリスト型に決定している。また、場合分けの後者は、型変数、リスト型以外の場合でもマッチするが、その場合は全て unify error が発生する。

マッチ構文の型推論の手続きを与えると以下のようになる。
MatchExp(exp1, exp2, id1, id2, exp3) を処理することを考える。

1. Γ、$exp1$ を入力として、型推論を行い、$S1$、$\tau 1$ を得る。 (2.29)

2. Γ、$exp2$ を入力として、型推論を行い、$S2$、$\tau 2$ を得る。 (2.30)

3. $\tau 1$ が $TyList\ ty$ であるとき手続き 4 へ
   $\tau 1$ が $TyVar\ num$ であるとき手続き 7 へ
   そうでないときはエラー発生 (2.31)

4. Γを ty で束縛された $id1$、$TyList\ ty$ で束縛された $id2$ で拡張した新しい環境 $N\Gamma$を得る。手続き 5 へ (2.32)

5. $N\Gamma$、$exp3$ を入力として、型推論を行い、$S3$、$\tau 3$ を得る。手続き 6 へ (2.33)

6. $S1 \cup S2 \cup S3 \cup (\tau 2, \tau 3)$ を単一化し、型代入 $S4$ を得、$S4$、$\tau 2$ を出力として返す。手続きを終了する。 (2.34)

7. 新しい型変数 $domty1$ を得、Γを $domty1$ で束縛された $id1$、$TyList\ domty1$ で束縛された $id2$ で拡張した環境 $N\Gamma$を得る。手続き 8 へ (2.35)

8. $N\Gamma$、$exp3$ を入力として、型推論を行い、$S5$、$\tau 3$ を得る。手続き 9 へ (2.36)

9. $S1 \cup S2 \cup S5 \cup (\tau 2, \tau 3) \cup (TyVar\ num, TyList\ domty1)$ を単一化し、型代入 $S6$ を得、$S6$、$\tau 2$ を返す。手続きを終了する。 (2.37)

手続き 4 での場合分けであるが、これもリストの型推論と同様で、リスト型か型変数であるかで場合分けしている。また後者の場合、リストの型がまだわからないので、新しい型変数 domty1 を作っている。また TyVar num が domty1 のリスト型であるとして整合性が取れるかを判断するため、単一化の条件に (TyVar num, TyList domty1) を加えている。

第3章　多相的letの型推論

3.1　Exercise 4.4.1

まず解説されていない全ての補助関数を定義する。ひとつ目は freevar_tysc を定義する。これは型スキーム σ を受け取り、σ の中の自由変数の集合をリストとして返す関数である。受け取る型スキームを TyScheme(tyvarlist ,ty) とする。match 構文を使い、この ty の形で場合分けする。TyInt、TyBool のときは空リストを返す。TyFun(ty1, ty2)、TyList ty1 のときは、それぞれフィールド値に自由変数が含まれる場合があるので、それぞれ、
(freevar_tysc (TyScheme(tyvarlist, ty1))) @ (freevar_tysc (TyScheme(tyvarlist, ty2)))、
freevar_tysc(TyScheme(tyvarlist , ty1)) を返せば良い。
TyVar num のときは、これが自由変数であれば [num] を返さなければならなく、そうでない時、すなわち tyvarlist に含まれるときには空リストを返せば良い。num が tyvarlist に含まれるかどうかの判定には List.mem 関数を利用する。これで実装は完了する。
ふたつ目は freevar_tyenv を定義する。これは現在の型環境 Γ から、Γ に含まれる全ての自由変数の集合を MySet 型として返すような関数である。そこでまず、Environmemt.ml に新しい関数 getschemelist を定義する。これは、タプルを要素とするような環境から、全ての要素の第二要素をリスト型として返すような関数である。すなわち、getschemelist に型環境 Γ をわたすと、Γ に含まれる全ての型スキームの集合をかえす。freevar_tyenv では、まず getschemelist を使って型スキームのリスト集合を schemelist に格納し、freevar_tyenv の中で再帰関数 freevar_tyenvrec を定義する。これは、型スキームのリスト集合を受け取って、そこに含まれる全ての自由変数の集合を MySet 型として返す。slist を受け取るとすると、match 構文で slist を一要素ずつ処理する。型スキームから自由変数の集合を取り出すのに freevar_tysc 関数を、リストを統合して MySet 型を返すのに MySet.insertlist を利用する。これで freevar_tyenvrec を定義でき、最終的に freevar_tyenvrec に schemelist を渡せば、freevar_tyenv の実装が完了する。また、freevar_tyenv 関数は、closure 関数の中で SΓ に出現する全ての自由変数の集合を求める際に利用する。

次に、let 式の型推論の手続きを与えると以下のようになる。

LetInExp(id, exp1, exp2) を処理することを考える。

1. Γ、$exp1$ を入力として型推論を行い、$S1$、$\tau 1$ を得る。 (3.1)

2. Γ、$\tau 1$、$S1$ を入力として、$closure$ 関数を呼び出し、型スキームσを得る。 (3.2)

3. Γをσで拡張した環境 $N\Gamma$を得る。 (3.3)

4. $N\Gamma$、$exp2$ を入力として型推論を行い、$S2$、$\tau 2$ を得る。 (3.4)

5. $S1 \cup S2$ を単一化し、型代入 $S3$ を得る。 (3.5)

6. Γ、$S3$、$\tau 2$ を出力として返す。 (3.6)

次に let 宣言の型推論の手続きを与えると以下のようになる。Decl(id, exp) を処理することを考える。

1. Γ、exp を入力として型推論を行い、$S1$、$\tau 1$ を得る。 (3.7)

2. Γ、$S1$、$\tau 1$ を入力として、$closure$ 関数を呼び出し、型スキーム $\sigma 1$ を得る。 (3.8)

3. Γを$\sigma 1$ で束縛された id で拡張した環境、$S1$、$\tau 1$ を出力として返す。 (3.9)

3.2 Exercise 4.4.2

まず簡単な補助関数 unionscheme を定義する。これは、型スキームを２つ引数で受け取り、それぞれの多相的な型変数を統合したリストを返すような関数である。実装は簡単なので説明は省略する。let rec 式の型推論の手続きを与えると以下のようになる。LetRecExp(id, para, exp1, exp2) を処理することを考える。

1. 新しい型変数 $domty1$、$domty2$ を作る。 (3.10)

2. Γを、$TyScheme([], domty1)$ で束縛された $para$、$TyScheme([], TyFun(domty1, domty2))$ で束縛された id で拡張した環境 $N\Gamma$を得る。 (3.11)

3. $N\Gamma$、$exp1$ を入力として型推論を行い、$S1$、$\tau 1$ を得る。 (3.12)

4. $S1 \cup (domty2, \tau 1)$ を単一化し、型代入 $S2$ を得る。 (3.13)

5. $domty1$、$domty2$ それぞれに型代入を適用して、$ty1$、$ty2$ を得る。 (3.14)

6. Γ、$ty1$、$S2$ を入力として $closure$ 関数を呼び出し、型スキーム$\sigma 1$ を得る。 (3.15)

7. Γ、$ty2$、$S2$ を入力として $closure$ 関数を呼び出し、型スキーム$\sigma 2$ を得る。 (3.16)

8. Γを $TyScheme(unionscheme\ \sigma 1\ \sigma 2,\ TyFun(ty1, ty2))$ で束縛された id で拡張した環境 $N\Gamma 2$ を得る。 (3.17)

9. Γ、$exp2$ を入力として型推論を行い、$S4$、$\tau 2$ を得る。 (3.18)

10. $S2$、$S4$ を単一化し、型代入 $S5$ を得る。 (3.19)

11. Γ、$S5$、$\tau 2$ を出力として返す。 (3.20)

手続きは基本的に今まで通りだが、手続き 5、6 の説明を補足する。ここでは、closure に domty を渡さずに、一旦型代入した ty を渡している。これは、closure の中で、受け取った引数の型に型代入を行っていないからである。(closure 関数の中で型代入を行っても良いが、どの道 ty1、ty2 は必要になる。)
次に、let rec 宣言の型推論の手続きを与えると以下のようになる。RecDecl(id, para, exp1) を処理することを考える。

1. $LetRecExp$ の手続きの 1〜7 を行う。 (3.21)

2.Γを $(TyScheme(unionscheme\ \sigma 1\ \sigma 2,\ TyFun(ty1, ty2))$ で束縛された id で拡張した環境、$S2$、$TyFun(ty1, ty2)$ を返す。 (3.22)

LetRecExp の処理とほぼ同じであり、説明は特にない。

第4章　その他

この章では、chap03-4 までの問題で追加で実装した問題や、他に自分で追加した機能について解説する。

4.1　Exercise 3.6.1

まず、字句解析に MATCH、WITH、PAIPU を追加する。次に構文解析に規則 MatchExpr を追加する。これはそのまま macth 構文にマッチするようにすればよく、以下のようになる。

$$\begin{aligned} &MATCH\ e1 = Expr\ WITH\ MDRPAREN\ MDLPAREN \\ &RARROW\ e2 = Expr\ PAIPU\ id1 = ID\ COROCORO\ id2 = ID \\ &RARROW\ e3 = Expr\ \{\ MatchExp(e1,\ e2,\ id1,\ id2,\ e3)\ \} \end{aligned} \tag{4.1}$$

この還元時アクションになっている、MatchExp の定義は、
MatchExp * exp * exp * id * id * exp となっている。次に、これの eval での評価について解説する。
MatchExp(exp1, exp2, id1, id2, exp3) を評価するとする。まず、exp1 の評価結果を e1 に束縛し、それを match 構文で場合分けする。まず e1 の型として許されるのは、NilV、Cons(e11, e12) のいずれかの場合である。どちらでもない時にはエラーを発生させる。e1 が NilV のときは、評価結果は exp2 を評価した結果を返せば良い。
e1 が Cons(e11, e12) のときは、id1 が e11、id2 が e12 になるので、その 2 つの変数で拡張した新しい環境のもとで、exp3 を評価し、その結果を返すようにすれば良い。これで実装は完了する。

4.2　その他の追加機能

まず、評価結果を出力することを考える。そのために、typing.ml に空の環境の参照 idenv を定義する。これは、一度の評価が終わった時点で新しく定義された全ての変数と型が格納される。すなわち、ty_decl の評価結果を返す前に、

$$idenv\ :\ =\ Environment.extend\ id\ ty\ !tyenv; \tag{4.2}$$

という一文を入れる (Exp e を除く)。これにより、新しく評価された変数とその型を格納できる。次に、cui.ml で結果を出力するための関数 print_localenv を説明する。これは、引数として tmp_localenv というものを持つ。これは、eval で新しく宣言された全ての変数とその評価結果のタプルのリストになる。また、eval では localenv という変数と評価結果のタプルのリストをグローバルとして定義している。print_localenv は、tmp_localenv の中身を match 構文を使って一要素ずつ評価する。そこで、型を出力する際に先ほどの idenv の中を Environment.lookup 関数を使って探しに行く。見つからない場合は式の型を返すので、ty をそのまま返す。最後に idenv に空の環境を破壊的代入をすることで、次の評価に備えて初期化する。RecAndLet の型推論の手続きを与えると以下のようになる。RecAndLet(id, para, exp1, exp2) を処理することを考える。

$$1.\quad \mathit{RecDecl}\ \text{の型推論の手続きの 1 を行う。} \tag{4.3}$$

$$2.\quad \Gamma\text{を、}\mathit{TyScheme}(\mathit{unionscheme}\ \sigma 1\ \sigma 2,\ \mathit{TyFun}(ty1, ty2))\ \text{で束縛した}\ id\ \text{で拡張した環境を}\ N\Gamma\text{とする。} \tag{4.4}$$

$$3.\quad \mathit{ty\_decl}\ N\Gamma\ \mathit{exp}2\ \text{を返す。} \tag{4.5}$$

基本的には、RecDecl とほぼ同じである。しかし、このままでは相互再帰が定義できない。例えば、以下のような式を宣言したいとする。

$$let\ f\ =\ fun\ n\ \rightarrow\ g\ n\ and\ g\ =\ fun\ n \rightarrow\ f\ n \tag{4.6}$$

これを構文解析すると、以下のようになる。

$$\begin{aligned} &\mathit{AndLet}(f,\ \mathit{FunExp}(n,\ \mathit{AppExp}(\mathit{Var}\ g,\ \mathit{Var}\ n)),\\ &\qquad\qquad \mathit{Decl}(g,\ \mathit{FunExp}(n,\ \mathit{AppExp}(\mathit{Var}\ f,\ \mathit{Var}\ n)))) \end{aligned} \tag{4.7}$$

これを評価していくと、一番目の AppExp で g が見つからないと怒られる。そこで、補助関数をいくつか実装する。まず、eval.ml に getVar 関数を定義する。これは、exp 型の値を受け取って、その中に現れる全ての変数をリストとして返すような関数である。実装は単純であるので、説明は省略する。次に、assigntyvar 関数を定義する。これは、id 型のリストと型環境を受け取って、まだ環境に入っていない id に対して、新しく型変数を作ってその id に対応させ、それらの変数で拡張された型環境を返すような関数である。実装は単純であるので、説明は省略する。次に、AppExp の処理を変更する。すなわち、初めの exp1、exp2 の処理を try 構文で囲み、Environment.Not_bound のエラーが発生したときに、assigintyvar 関数に tyenv と (Eval.getVar exp1) を渡して返ってきた環境のもとで、exp1 を再び評価する。これにより、まだ定義されていない変数の型をとりあえず型変数で置くことができる。exp2 でも同様である。これにより、相互再帰関数も実装できる。

第5章　終わりに

5.1　工夫した点

工夫した点として、初めの方に型を出力するように cui.ml を変更したことである。これによって型が予期せぬ型になっているときにデバッグをするのが楽になった。また、型推論のパートでは型推論を行わなければならない exp 型 の評価が比較的少なかったが、eval.ml で定義されている exp 型 の評価を書いていないと、宣言できないものが出てくるので、eval.ml で評価した exp 型、program 型は typing.ml でも全て評価するようにした。これによって定義したものはしっかり宣言、評価ができるようになった。

5.2　感想

全体としては、比較的楽しく実装できた。しかし LR 構文解析を完全に理解ができておらず、パーズできない理由などがわからずに苦労した時があったので、構文解析の勉強が必要だと思った。また、型推論パートで難しかった点としては、let 多相の型推論の実装で、補助関数などの機能を、全体的に繋げるのに少し時間がかかった。また、let rec 多相では、型代入を一部行っていないなどで、かなりデバッグに時間がかかったのはきつかった。インタプリタ実験は CPU 実験より 100 倍楽しかったので、次はコンパイラを作ったりもしてみたいと思った。