レポート3(型推論器)

1029289895 尾崎翔太

2018/7/20

1　はじめに

ソースコード単位で解説していくのではなく、課題単位で解説していく。また、説明は現在のソースコードに合わせたり合わせなかったりしている。さらに、最後の方で微妙に時間が余ったので、課題に関係ないものを実装してあるが、あまり気にしないでいただきたい。

2　各課題の説明

2.1　Exercise 4.2.1

2.1.1　設計方針

教科書に載っている部分はそのまま利用した。教科書に載っていない if 式と let 式については教科書の説明を読んで、二項演算の部分からの類推も合わせて自然に思いつくような設計にした。

2.1.2　実装の詳細

今のソースコードにこの頃の原型はほとんど残っていないので述べることはない。let 多相以前の型推論については Exercise 4.3.5 でまとめて述べる。

2.2　Exercise 4.3.1

2.2.1　設計方針

いろいろ実装した今の状態のものに関して述べる。pp_ty 関数は、pp_ty 関数の中で string を作るのではなくて、まず ty を string に変換する関数を作って、そこに受け取った型を渡して、得られたものを表示することにした。なぜなら、関数型などは再帰的な作業を行わないと string 型に変換できず、pp_ty の中ではするのは大変だと思ったからである。また、型変数については'a,...,'z,'a1,... となるようにしたかったので、表示しようとしている型から型変数を取り出して、表示する際に左に現れるものから順に'a,'b,... と割り当てることにした。当然同じ型変数には同じアルファベットが割り当てられる。freevar_ty 関数は単純にリストでするところを MySet にしただけで実装できた。

2.2.2 実装の詳細

pp_ty 関数に関する部分は syntax.ml の 99〜155 行目である。各関数の意味は関数名とコメントからわかってもらえると思っている。型中に現れる型変数を左から順に 0,1,... と対応付けて、それを 26 で割った時の商と余りを用いて'a,'b,... といったアルファベットを生成する。型変数以外の部分は単純に再帰的に string に変換している。freevar_ty 関数は 167〜174 行目にある。特に特筆すべきことはない。

2.3 Exercise 4.3.2

2.3.1 設計方針

subst_type 関数は subst の中身を順番に型に適用していく部分と、各適用に関して、型の隅々まで適用する部分の二つのループが必要である。そこで、subst_type 関数の内部に、もう一つ let rec で定義された関数を用意することにした。

2.3.2 実装の詳細

typing.ml の 10〜20 行目に subst_type 関数がある。単純に再帰的に subst を適用していくだけで、特筆すべきことはない。

2.4 Exercise 4.3.3

2.4.1 設計方針

教科書に載っているアルゴリズムを素直に実装した。FTV についても、単純に freevar_ty 関数を用いた。

2.4.2 実装の詳細

関係している部分は typing.ml の 27〜45 行目である。subst_eqs 関数は、eqs 中の各型に subst を適用する関数である。また、リストに関しては、何のリストか、という部分が等しいという制約を新たに追加している。

2.5 Exercise 4.3.4

もし $\alpha \in FTV(\tau)$ だったとする。例として $\tau = \alpha \to \alpha$ とする。これは、意味としては $\alpha = \alpha \to \alpha$ という制約が存在するということである。すると、$\tau = (\alpha \to \alpha) \to (\alpha \to \alpha)$ となって、さらに α に $\alpha \to \alpha$ を代入して、、、となる。つまり、無限に再帰が起こって、τ の型が定まらない。すなわち、エラーということである。よって、$\alpha \notin FTV(\tau)$ が必要となる。

2.6 Exercise 4.3.5

2.6.1 型推論の手続き

T-VAR 1. Γ, x を入力として型推論を行い、$\emptyset, \tau$ を得る。

2. $\emptyset$ と τ を出力として返す。

T-INT 1. Γ, n を入力として型推論を行い、$\emptyset$,int を得る。

2. $\emptyset$ と int を出力として返す。

T-BOOL 1. Γ, n を入力として型推論を行い、$\emptyset$,bool を得る。

2. $\emptyset$ と bool を出力として返す。

T-MULT 1. Γ, e_1 を入力として型推論を行い、$\mathcal{S}_1, \tau_1$ を得る。

2. Γ, e_2 を入力として型推論を行い、$\mathcal{S}_2, \tau_2$ を得る。

3. 型代入 $\mathcal{S}_1, \mathcal{S}_2$ を $\alpha = \tau$ という形の方程式の集まりとみなして、$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \{(\tau_1,\text{int}), (\tau_2,\text{int})\}$ を単一化し、型代入 $\mathcal{S}_3$ を得る。

4. $\mathcal{S}_3$ と int を出力として返す。

T-LT 1. Γ, e_1 を入力として型推論を行い、$\mathcal{S}_1, \tau_1$ を得る。

2. Γ, e_2 を入力として型推論を行い、$\mathcal{S}_2, \tau_2$ を得る。

3. 型代入 $\mathcal{S}_1, \mathcal{S}_2$ を $\alpha = \tau$ という形の方程式の集まりとみなして、$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \{(\tau_1,\text{int}), (\tau_2,\text{int})\}$ を単一化し、型代入 $\mathcal{S}_3$ を得る。

4. $\mathcal{S}_3$ と bool を出力として返す。

T-IF 1. Γ, e_1 を入力として型推論を行い、$\mathcal{S}_1, \tau_1$ を得る。

2. Γ, e_2 を入力として型推論を行い、$\mathcal{S}_2, \tau_2$ を得る。

3. Γ, e_3 を入力として型推論を行い、$\mathcal{S}_3, \tau_3$ を得る。

4. 型代入 $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ を $\alpha = \tau$ という形の方程式の集まりとみなして、$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \{(\tau_1,\text{bool}), (\tau_2, \tau_3)\}$ を単一化し、型代入 $\mathcal{S}_4$ を得る。

5. $\mathcal{S}_4$ と $\mathcal{S}_4(\tau_2)(= \mathcal{S}_4(\tau_3))$ を出力として返す。

T-LET
1. Γ, e_1 を入力として型推論を行い、$\mathcal{S}_1, \tau_1$ を得る。
2. $\Gamma, x : \tau_1, e_2$ を入力として型推論を行い、$\mathcal{S}_2, \tau_2$ を得る。
3. 型代入 $\mathcal{S}_1, \mathcal{S}_2$ を $\alpha = \tau$ という形の方程式の集まりとみなして、$\mathcal{S}_1 \cup \mathcal{S}_2$ を単一化し、型代入 $\mathcal{S}_3$ を得る。
4. $\mathcal{S}_3$ と $\mathcal{S}_3(\tau_2)$ を出力として返す。

T-FUN
1. $\Gamma, x : \tau_1, e$ を入力として型推論を行い、$\mathcal{S}, \tau_2$ を得る。
2. $\mathcal{S}$ と $\mathcal{S}(\tau_1 \to \tau_2)$ を出力として返す。

T-APP
1. Γ, e_1 を入力として型推論を行い、$\mathcal{S}_1, \tau_1$ を得る。
2. Γ, e_2 を入力として型推論を行い、$\mathcal{S}_2, \tau_2$ を得る。
3. 型代入 $\mathcal{S}_1, \mathcal{S}_2$ を $\alpha = \tau$ という形の方程式の集まりとみなして、$\mathcal{S}_1 \cup \mathcal{S}_2 \cup \{(\tau_1, \tau_2 \to \tau_3)\}$ を単一化し、型代入 $\mathcal{S}_3$ を得る。
4. $\mathcal{S}_3$ と $\mathcal{S}_3(\tau_3)$ を出力として返す。

2.6.2 設計方針

上記したアルゴリズムに従って実装した。

2.6.3 実装の詳細

教科書の図 4.4 に倣って実装した。IfExp は単純に実装できたが、LetExp は課題 2 でいろいろと拡張していたので少し面倒だったが、eval と同じように処理することにした。AppExp は e_1 の型が関数型なら単に定義域の型と e_2 の型が等しいという制約を加えて値域の型を返すだけで良かったが、型変数なら、新たに型変数を二つ作ってそれぞれ定義域と値域として、e_1 の型に関する制約も付け加えなければならなかった。他の部分は教科書そのままである。

2.7 Exercise 4.3.6

2.7.1 設計方針

バックパッチではなく、単に正しい値の束縛を追加することでダミーの束縛を隠すことにした。

2.7.2 実装の詳細

関係する部分は typing.ml の 394〜435 行目である。let rec 式も課題 2 の拡張のせいで少し複雑になっている。まず、適当に関数型を作って変数がそれに束縛されているようにする。その状態で本当に束縛すべき値を評価して等式集合を得る。次にそれを単一化して、環境からダミーを取り出して型代入を適用して、再度環境に追加する。そして、その環境下で最後の式を評価する。気をつけるべきことは、e_1 の型を得るときに x を domty で拡張することと、e_1 の型と ranty の型が等しいという制約を追加することである。

2.8 Exercise 4.3.7

2.8.1 設計方針

基本的には eval の時と同じような流れで行う。大きく違うのはパターンマッチの部分で、eval の時は束縛のリストを返すだけで良かったが、今回はそれに加えて、型変数に関する情報として型代入の集合も返すようになっている。こうすることで、各パターン列が異なる型を持っていた場合に型エラーを検出できる。

2.8.2 実装の詳細

主に match 式について述べる。関係する部分は typing.ml の 114〜154、452〜509 行目である。match 式も課題 2 の拡張のせいでとても複雑になっている。しかし、設計方針で述べたように基本的には eval と同じである。ただ、単一化すべき等式集合が多くてややこしくなっている。詳細はプログラムのコメントを見ていただきたい。

2.9 Exercise 4.4.1

2.9.1 設計方針

図 4.5,4.6 に載っている部分はそのまま実装した。let 式、宣言は、環境を拡張する部分が型から型スキームに変えることで実装した。

2.9.2 実装の詳細

最初は単純に環境を型ではなく型スキームで拡張するようにしたのだが、これだとこの課題のテストは通ったが、前の課題のテストが通らなくなっていた。要するに型スキームにすべき範囲が広すぎたのだ。結果として、closure

関数の第三引数を利用して環境に型代入を適用することで解決した。他の部分については特に述べることはない。

2.10 Exercise 4.4.2

2.10.1 設計方針

let 式、宣言と同じである。

2.10.2 実装の詳細

変更点は let 式、宣言と同じなので特筆すべきことはない。

2.11 Exercise 4.4.3

2.11.1 設計方針

まず、id や exp と型の組を新たな型として定義する。そして、それが評価されて、型代入と型を返す前に、その付与された型との整合性もチェックするようにする。また、型注釈に型変数が使用されている場合は、その型変数は string 型で表現されているので通常の型を表す型とは別の型を用意する必要がある。

2.11.2 実装の詳細

まず、syntax.ml に新たに attached_ty や typedId や typedExp といった型を追加する。意味は名前の通りである。そして、exp の各コンストラクタにぶらさがっている id と exp を typedId と typedExp に変更する。attached_ty の Ranty は、let f x : int = x のような場合の int は f の型そのものではなく、f の値域の型であるから、それを表したものである。TransformedTyvar は中間表現で、Tyvar を TyVar に変換するためのものである。次に、lexer.mll に：や型名に関するルールを追加する。型変数の場合は'(アルファベット)(アルファベット or 数字 or　アンダースコア)*である。次に parser.mly も大きく変更された。まず、IDt という非終端記号を追加した。これは、単なる ID か型注釈付きの ID のどちらかを表す。nonempty_list を適用するために一つの非終端記号にまとめる必要があったのである。exp については、基本的に AExpr で型注釈を追加するようにした。それ以外では単純に exp を返していたところを (exp, []) を返すようにした。最後に、typing.ml もかなり変更された。まず、与えられた式、宣言中の Tyvar を TransformedTyvar にする必要がある。そのために get_attached_ty_list 関数で式中の型注釈を全て集めて、

get_attached_tyvar_list 関数を用いながら make_Tyvar_to_TyVar_list 関数で Tyvar と TyVar の対応表を作る。その対応表の実体は、string と int の組である。その対応表を用いて transform 関数で式中の Tyvar を TransformedTyvar に変換する。その後、式を評価していき、今まで返していた型代入 $\mathcal{S}$ と型 τ を求めたあとに、make_eqs_about_att_ty 関数で、型注釈に関する等式集合を得て、それも含めて新たに単一化をして型代入 $\mathcal{S}'$ を得て、$\mathcal{S}'$ と $\mathcal{S}'(\tau)$ を返す。また、make_eqs_about_att_ty 関数には、attached_ty を ty に変換する ty_of_attached_ty 関数を用いている。最後に、ocaml に合わせるために、LetRecExp は id と FunExp を引数に取るようにした。

3 感想

全体的に楽しかった。ただ、やっぱり選択課題をすべてやるには時間が足りなさすぎると思った。また、テストケースがなぜ通らないのかがわからない時が辛かった。型推論は、アルゴリズムが教科書に載っているのでそこまで難しくはなかった。実験 4 のコンパイラもやろうかなと思った。