型推論機能を持つ意味解析器の生成系

佐藤 宇洋† 中井 央† 天野 勝利† 佐藤 聡†
†筑波大学

1 はじめに

コンパイラの実装の負担を削減するために構文解析・字句解析では Yacc や Lex といった自動生成系が広く使われている。しかし、意味解析において型推論器を生成する生成系は無いため、型推論器を作成するには、型推論規則に基づいて開発者が手書きで実装するのが一般的であった。

本論文では、入力された型推論規則から型推論器を生成する生成系について述べる。生成系は関数型言語 OCaml を用いて実装され、生成されるコードも OCaml である。生成系が生成する型推論器の設計には [1] と [2] のソースプログラムを参考にした。

2 システムの設計と実装

型推論器の作成は容易ではない。型推論器の実装を手作業で行うには、複雑なコードを実装する作業が必要である。本研究では、この作業を行わなくても、型推論器を構成する要素の作成に必要な最低限の情報から型推論器を作成できることを目的とする。この章では、生成系の設計方針と生成系に与える記述、本生成系を用いたコンパイラの実装について述べる。

2.1 生成系の設計方針

型推論を行う処理系は、抽象構文木を出力する構文解析器と出力される抽象構文木を走査する型推論器から構成される。このような処理系を具体的に実装するには、抽象構文木、その言語が持つ型、型推論をしていく際の環境についてデータ表現を定義する必要がある。そして、それらに基づいて実際に型推論を行っていくための単一化を行う関数と各抽象構文に対する型付けを行う関数を実装する必要がある。

これらの要素を機械的に作成するために以下のことを考えた。単一化関数は基本的には言語の型の構造に従って型が等しいかどうか再帰的に確認していく関数なので、言語の型情報に基づいて生成することができる。型のためのデータ型の定義も同様に言語の型情報に基づいて生成することができる。型推論規則には構文と構文に対する型付けの形式的規則が表現されている。構文の表現には抽象構文を表すための情報が含まれているので、ここから抽象構文木のためのデータ型の定義を生成することができる。また、型付けの形式的規則に基づいて型付けを行う関数を生成することができる。その他の要素は生成系が予め用意することで、型推論器を構成することができる。これらに基づいて、生成系への入力を次節に述べるように設計した。

2.2 生成系に与える記述

図 1 に生成系に与える記述例を示す。生成系に与える記述は%%に区切られた型宣言部と型推論規則部から構成される。型宣言部では `%type` を用いた宣言により、型推論規則中に使用する型を記述する。型推論規則部には、各構文要素に対する型推論規則を記述する。型推論規則の記法については後述する。

型宣言部には型を表記する型式を記述する。ここには、基本型と型構成子を宣言することができる。型式は基本型、あるいは型式に型構成子を用いたものからなる。型構成子の定義は図 1 の 3 行目のような形で行う。この例では、`ty` を型式とすると、`TyFun(ty, ty)` は型式であり、要素の型 `ty` を 2 つ持つ型を表す。

型推論規則部での型推論規則の形式的な記述を示す。

```
begin {(* 上段規則部 *)}{(* 下段規則部 *)} end
```

上段規則部が規則の前提で、下段規則部が規則の結論である。上段規則部には 0 個以上の規則式を記述し、下段規則部には 1 個の型判断を記述する。型判断とは、ある前提からある式 e が型 ty を持つことが示される形式的記述のことである。前提は識別子と型の対の列によって構成される環境に対応する。

<環境> $\vdash e : ty$

<環境>には環境の表現を記述し、 e には式の表現、ty には型式を記述する。生成系は 2 つの環境を生成する。1 つは変数と型の対の列で構成される値環境 (value environment) ともう 1 つは型識別子と型の対の列で構成される型環境 (type environment) である。入力記述では、値環境を `venv` 、 型環境を `tyenv` と表現する。また、これら両方を含む環境を `env` と表現する。

ある識別子 `x` は型 `ty1` であるという情報を環境に加えた下で型判断を行うには次のように記述する。

<環境>, `x : ty1` $\vdash e : ty$

この場合、識別子 `x` の束縛の有効範囲は式 `e` の中だけである。<環境>の部分には、拡張する環境を値環境

```
1  (* 型宣言部 *)
2  %type TyInt
3  %type TyFun(ty, ty)
4  %%
5  (* 型推論規則部 *)
6  begin (* T-FunExp *)
7  {venv, id : ty1 |- exp : ty2;}
8  {env |- FunExp(id, exp) : TyFun(ty1, ty2);}
9  end
10 ...
11 ;;
```

図 1: 入力記述例

```
let main venv tyenv =
  try
    print_string "# ";
    flush stdout;
    let tree = Parser.toplevel Lexer.main
      (Lexing.from_channel stdin) in
    let ty = typing tyenv venv tree in
    Printf.printf "%s" (pp_ty ty)
    print_newline();
  with
    Tg_type_error(ty1, ty2, s) -> ...
```

図 2: コンパイラを構成するプログラム

venv か型環境 tyenv のどちらかに指定する必要がある。また、両方拡張したい場合は次のように記述する。

venv, x1 : ty1 tyenv, x2 : ty2 $\vdash e : ty$

下段規則部の型判断の式の部分には、次の形で抽象構文木の要素を表す記述をする必要がある。

<構文名>(<式$_1$>, ..., <式$_n$>)

2.3 生成系の構成

生成系は、入力記述中の型宣言部を処理する型宣言処理部、型推論規則部を処理する型推論規則処理部から構成される。

型宣言処理部は型宣言部から、%type 節に記されている型情報を収集する。そして、その情報を基に型を表すためのデータ型を定義し、型の単一化を行う関数と、型変数による循環参照を検査する関数を生成する。

型推論規則処理部は各型推論規則の構文要素を収集し、抽象構文のためのデータ型を定義する。構文要素の処理が終わると再び各型推論規則を処理し、各構文に対して型付けを行う関数を作成する。型推論規則の処理は各規則式を処理する間、リストに型を登録・参照しながらコードを構成する。

2.4 コンパイラ作成

この章では、字句解析器生成系 ocamllex と構文解析器生成系 ocamlyacc および本生成系を用いた型推論を行うコンパイラの作成について述べる。コンパイラの作成には、ocamllex, ocamlyacc への入力記述と、型推論に必要なプログラムを自動生成するための本システムへの入力記述、それらを動かすためのメインプログラムが必要である。図 2 にメインプログラムの例を示す。ここでは、構文解析器が生成した抽象構文木を型推論器の入力とし、型推論の結果と型エラーの表示を行う。これらのファイルをコンパイル、リンクさせて作成した実行ファイルがコンパイラとなる。

3 適用例

本システムを用いて miniML [2] インタプリタと、TL [3] コンパイラの実装を行った。miniML 言語は関数型言語で、ML に似た構文を持つ。型は基本型のほかに関数型、リスト型を持つ。TL 言語は手続き型言語で、C 言語に似た構文を持つ。基本型のほかに配列型、ポインタ型、レコード型を持つ。また、TL 言語ではユーザが型に型名をつけることができる。これらの言語処理系の実装方針は、意味解析において型推論を行い、コード生成は行わないものとした。

本生成系を用いることで、ocamllex・ocamlyacc・本生成系に与える記述とメインプログラムを作成するだけでコンパイラを作成することができた。

4 おわりに

本生成系により、容易に型推論機能を持つ意味解析器を実装できるようになった。しかし、多相型の型推論やオブジェクト指向言語が持つ型に関してはほとんど考慮されておらず、これらは今後の課題である。

参考文献

[1] 住井英二郎. MinCaml コンパイラ. 日本ソフトウェア科学会:コンピュータソフトウェア, Vol. 25, No. 2, pp. 28–38, 2008.

[2] 五十嵐淳. 型推論によるプログラム解析. http://www.sato.kuis.kyoto-u.ac.jp/~igarashi/class/isle4/text/index.html. 2008-5-19 アクセス.

[3] Andrew W. Appel. *Modern Compiler Implementation in ML*. The Press Syndicate of The University of Cambridge, 1997.