プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 7: 型システムその 2

目次

先週のまとめ

	C/C++	Lisp	ML,Haskell	Java	Ruby,JavaScript
静的/動的	静的	動的	静的	静的	動的
検査/推論	型検査	—	型推論	型検査	—

- 静的型付けと動的型付け
  - 静的: 実行前に検査/推論
  - 動的: 実行時に検査/推論
- 型検査と型推論
  - 型検査: 変数などの型は宣言済み⇒型の整合性を検査
  - 型推論: 変数などの型は未知⇒型を推論しつつ型の整合性を検査

目次

型システムの例

整数、足し算、比較、if 式だけがある言語 L_1：

$$e ::= i \mid e + e \mid e > e \mid \texttt{if } e \texttt{ then } e \texttt{ else } e$$
$$t ::= \texttt{bool} \mid \texttt{int}$$

型付け $e : t$: 「式 e が型 t を持つ」こと

$$(4 + 5) + 6 > 3 \ : \ \texttt{bool}$$
$$\texttt{if } (5 + 6 > 3) \texttt{ then } (1 + 2) \texttt{ else } (3 + 4) \ : \ \texttt{int}$$
$$\texttt{if } (5 + 6 > 3) \texttt{ then } 1 \texttt{ else } (3 > 4) \quad \text{型エラー}$$

型検査

型検査 (type checking) 問題:

- 入力: 式 e と 型 t
- 出力: Yes か No
- 仕様: $e : t$ のとき Yes、$e : t$ でないとき No

型検査の例:

$$(4 + 5) + 6 > 3 \ : \ \texttt{bool} \quad \to \text{Yes}$$
$$(4 + 5) + 6 > 3 \ : \ \texttt{int} \quad \to \text{No}$$
$$\texttt{if } (5 + 6 > 3) \texttt{ then } 1 \texttt{ else } (3 > 4) \ : \ \texttt{int} \quad \to \text{No}$$

言語の拡張

前の言語に変数と let 式を追加した言語 L_2:

$$e ::= \cdots \mid x \mid \texttt{let } x : t = e \texttt{ in } e$$
$$t ::= \text{(前と同じ)}$$

型付けの例:

$$\texttt{let } x : \texttt{int} = 1 + 2 \texttt{ in } x > 3 \ : \ \texttt{int}$$
$$\texttt{let } x : \texttt{int} = 1 > 2 \texttt{ in if} x 3 4 \ : \ \texttt{int}$$

型システム

$e : t$ だけでは情報が足りない。

$$e_1 = \qquad x + 10 : \texttt{int}$$
$$e_2 = \qquad \texttt{if } x \texttt{ then } 10 \texttt{ else } 20 \ : \texttt{int}$$
$$e_1 + e_2 \quad \text{型エラー}$$

e_1 と e_2 はそれぞれ単独では型がつくが、1 つのプログラムで両方を使うと型がつかない。

- $x : \texttt{int} \vdash e_1 : \texttt{int}$
- $x : \texttt{bool} \vdash e_2 : \texttt{int}$
- $x : t_1 \vdash e_3 : t_2$ となる型 t_1, t_2 はない。

型判定

型判定 (Typing Judgement) $\Gamma \vdash e : t$

- $\Gamma = x_1 : t_1,\ x_2, t_2, \cdots, x_n : t_n\ (n \geq 0)$
- Γ (ガンマの大文字) は型文脈、e に含まれる自由変数たちの型付け

例:

$$x : \mathtt{bool}, y : \mathtt{int} \vdash x \ : \ \mathtt{bool}$$
$$x : \mathtt{int}, y : \mathtt{int} \vdash \mathtt{if}\ x > 3\ \mathtt{then}\ 1\ \mathtt{else}\ x + y \ : \ \mathtt{int}$$
$$x : \mathtt{int} \vdash \mathtt{if}\ x > 3\ \mathtt{then}\ 1\ \mathtt{else}\ x + y \quad \text{エラー}$$
$$x : \mathtt{int}, y : \mathtt{bool} \vdash \mathtt{if}\ x > 3\ \mathtt{then}\ 1\ \mathtt{else}\ x + y \quad \text{エラー}$$

型システムの規則その 1

L_1 に対応する規則たち:

$$\frac{}{\Gamma \vdash i : \mathtt{int}}\ i \text{ が整数定数のとき}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{int} \quad \Gamma \vdash e_2 : \mathtt{int}}{\Gamma \vdash e_1 + e_2 : \mathtt{int}}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{int} \quad \Gamma \vdash e_2 : \mathtt{int}}{\Gamma \vdash e_1 > e_2 : \mathtt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{bool} \quad \Gamma \vdash e_2 : t \quad \Gamma \vdash e_3 : t}{\Gamma \vdash \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : t}$$

L_1 での型付けの例

$$\dfrac{\dfrac{\dfrac{}{\vdash 3 : \mathtt{int}} \quad \dfrac{}{\vdash 5 : \mathtt{int}}}{\vdash 3 > 5 : \mathtt{bool}} \quad \dfrac{}{\vdash 7 : \mathtt{int}} \quad \dfrac{\dfrac{}{\vdash 5 : \mathtt{int}} \quad \dfrac{}{\vdash 8 : \mathtt{int}}}{\vdash 5 + 8 : \mathtt{int}}}{\vdash \mathtt{if}\ 3 > 5\ \mathtt{then}\ 7\ \mathtt{else}\ 5 + 8 : \mathtt{int}}$$

型システムの規則その 2

L_2 に対応する規則たち:

$$\frac{}{\Gamma \vdash x : t}\ (\mathrm{x{:}t}) \in \Gamma \ \text{のとき}$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma, x : t_1 \vdash e_2 : t_2}{\Gamma \vdash \mathtt{let}\ x : t_1 = e_1\ \mathtt{in}\ e_2 : t_2}$$

L_2 での型付けの例

$$\dfrac{\dfrac{}{x : \mathtt{int} \vdash x : \mathtt{int}} \quad \dfrac{\text{(省略)}}{x : \mathtt{int}, y : \mathtt{int} \vdash y > 10 : \mathtt{bool}}}{x : \mathtt{int} \vdash \mathtt{let}\ y : \mathtt{int} = x + 1\ \mathtt{in}\ y > 10 : \mathtt{bool}}$$

型検査問題

型検査 (type checking) 問題

- 入力: 型文脈 Γ と式 e と 型 t
- 出力: Yes か No
- 仕様: $\Gamma \vdash e : t$ が上述の規則から導けるとき Yes、そうでないとき No

型検査アルゴリズム

L_3 に対する 型検査問題を解くアルゴリズム (あるいは計算機プログラム) が存在する。

課題: L_2 に対する型検査アルゴリズムを設計・実装せよ。

型を明示しない言語

L_2 において、let 式の変数に型をつけない言語 L_3:

$$e ::= \cdots \mid \mathtt{let}\ x = e\ \mathtt{in}\ e$$
$$t ::= \text{(前と同じ)}$$

型付けの例:

$$\vdash \mathtt{let}\ x = 1 + 2\ \mathtt{in}\ \mathtt{let}\ y = x > 5\ \mathtt{in}\ \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1\ :\ \mathtt{int}$$

型システムの規則その 3

L_3 に対応する規則たち:

$$\dfrac{\Gamma \vdash e_1 : t_1 \quad \Gamma, x : t_1 \vdash e_2 : t_2}{\Gamma \vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : t_2}$$

束縛変数 x に対する型 t_1 を書かない。

L_3 での型付けの例

$$\dfrac{\dfrac{\overline{x : \mathtt{int} \vdash x : \mathtt{int}}}{x : \mathtt{int} \vdash x + 1 : \mathtt{int}} \quad \dfrac{\text{省略}}{x : \mathtt{int}, y : \mathtt{int} \vdash y > 10 : \mathtt{bool}}}{x : \mathtt{int} \vdash \mathtt{let}\ y = x + 1\ \mathtt{in}\ y > 10 : \mathtt{bool}}$$

型推論問題

型推論 (type inference) 問題

- 入力: 型文脈 Γ と式 e
- 出力: (Yes, t) または (No, -)
- 仕様: Yes のときは、$\Gamma \vdash e : t$ が上述の規則から導ける。No のときはどんな t に対しても導けない。

型推論問題のバリエーション

- 入力: 式 e のみ
- 出力: (Yes, t, Γ) または (No, -)
- 仕様: Yes のときは、$\Gamma \vdash e : t$ が上述の規則から導ける。No のときはどんな t に対しても導けない。

型推論アルゴリズム

L_3 に対する 型推論問題を解くアルゴリズム (あるいは計算機プログラム) が存在する。

練習問題: 以下の式の型付けを行ってみなさい。

- $\mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ y\ \mathtt{else}\ z$
- $\mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ x\ \mathtt{else}\ x$
- $\mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{let}\ y = 3 + 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ y\ \mathtt{else}\ y + 5$

解答 (の一例):

- $y : \mathtt{int}, z : \mathtt{int} \vdash \mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ y\ \mathtt{else}\ z : \mathtt{int}$ (これら 3 つの int を bool に変更したものでもよい。)
- $\mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ x\ \mathtt{else}\ x$ は型付けできない。(x を bool 型としても int 型としても矛盾する。)
- $y : \mathtt{int} \vdash \mathtt{let}\ x = 3 < 5\ \mathtt{in}\ \mathtt{let}\ y = 3 + 5\ \mathtt{in}\ \mathtt{if}\ x\ \mathtt{then}\ y\ \mathtt{else}\ y + 5 : \mathtt{int}$

関数型の型付け

言語 L_4 の構文:

$$e := \ldots \mid \lambda x.e \mid e\ e$$
$$t := \ldots \mid t \to t$$

L_4 に対応する規則たち:

$$\dfrac{\Gamma, x : t_1 \vdash e\ :\ t_2}{\Gamma \vdash \lambda x.e\ :\ t_1 \to t_2}$$

$$\dfrac{\Gamma \vdash e_1\ :\ t_1 \to t_2 \quad \Gamma \vdash e_2\ :\ t_1}{\Gamma \vdash e_1\ e_2\ :\ t_2}$$

関数型の型付け例その 1

$\lambda f.\ (f\ 3) + 5$ の型付け:

$$\dfrac{\dfrac{\overline{f : T \vdash f\ :\ \mathtt{int} \to \mathtt{int}}\quad \overline{f : T \vdash 3\ :\ \mathtt{int}}}{f : T \vdash f\ 3\ :\ \mathtt{int}}\quad \overline{f : T \vdash 5\ :\ \mathtt{int}}}{\dfrac{f : T \vdash (f\ 3) + 5\ :\ \mathtt{int}}{\vdash \lambda f.\ (f\ 3) + 5\ :\ (\mathtt{int} \to \mathtt{int}) \to \mathtt{int}}}$$

ここで、型 T は、$\mathtt{int} \to \mathtt{int}$ をあらわす。

関数型の型付け例その 2

$\lambda f.\lambda x.\ f\ (f\ x)$ の型付け:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f\ :\ \alpha \to \alpha}\quad \dfrac{\overline{\Gamma \vdash f\ :\ \alpha \to \alpha}\quad \overline{\Gamma \vdash x\ :\ \alpha}}{\Gamma \vdash f\ x\ :\ \alpha}}{\Gamma \vdash f\ (f\ x)\ :\ \alpha}}{\dfrac{f : \alpha \to \alpha \vdash \lambda x.\ f\ (f\ x)\ :\ \alpha \to \alpha}{\vdash \lambda f.\lambda x.\ f\ (f\ x)\ :\ (\alpha \to \alpha) \to (\alpha \to \alpha)}}$$

ただし、$\Gamma = f : \alpha \to \alpha,\ x : \alpha$

ML 言語の多相型 (polymorphic type)

map の型:

$(\alpha \to \beta) \to (\alpha\ \mathtt{list} \to \beta\ \mathtt{list})$

```
let inc x = x + 1;;
map inc [1; 2; 3];;
=> [2; 3; 4]
```

(map の型は、(int -> int) -> (int list -> int list))

```
let add1 x = x ^ "1";;
map add1 ["kameyama"; "yukiyoshi"];;
=> ["kameyama1"; "yukiyoshi1"]
```

(map の型は、
(string -> string) -> (string list -> string list))

ML 言語の多相型 (polymorphic type)

ユーザ定義関数における多相型; let で導入される。

```
let f (x,y) = (y,x);;
==> f : α * β → β * α
let g x = x in ((g 10), (g "abc"))
==> g : α → α
```

ちなみに、以下の式は ML では、多相型と見なされない。

```
(fun h →  ((h 10), (h "abc"))) (fun x → x)
==> type error
```

ML 言語の多相型 (polymorphic type)

ユーザ定義関数における多相型; let で導入される。

```
let g f = f f;;
==> type error

let f x = x in
    (f f) 10 ;;
==> OK
```

f の型

- let で定義した f ... 'a -> 'a
- 1 つ目の f ... (int -> int) -> (int -> int)
- 2 つ目の f ... int -> int

多相型の利点

もし、map 関数を C 言語で書くとしたら‥。

- 方法 1. int 型に対する map, string 型に対する map などを別々に定義する。
- 方法 2. 「void 型に対する map」を定義して、使うときに各型に cast する。
- 方法 3. C++ の template を使う。「T 型に対する map」を定義して、この関数を使うときに T を具体化する。

方法 1 は、コード量が多くなる、同じコードを何度も書くため保守性が悪い、等のデメリットがある。
方法 2 は、型の検査を素通りするため、型に関する間違いのチェックができなくなる等のデメリットがある。
方法 3 は、多相型と基本的に同じ効用がある。ただし、C/C++言語自体に組み込まれた機能ではないので、型エラーが起きたときに原因となるコードを発見しにくい等のデメリットがある。

多相型

パラメータ多相 (Parametric polymorphism): ML 言語のほか、Haskell などの関数型言語で利用可能。
cf. ほかに、オブジェクト指向言語の subtyping polymorphism、Haskell を含む一部の言語の ad hoc polymorphism がある。

型推論アルゴリズム

ML 言語 (多相型を含む) に対する 型推論問題を解くアルゴリズム (あるいは計算機プログラム) が存在する。

まとめ

型が整合していたからといって、プログラムが「正しい」わけではない。しかし、型が整合しているかどうかの検査をやるだけでも、(人間がよくやる) 多くの間違いを早期に発見できることが多い。[経験的事実]

- 型システム
- 静的型付け vs 動的型付け
- 多相型

積み残し: オブジェクト指向言語 (特に Java) の型システム (Generics, subtyping polymorphism, ...)

議論

動的型付けの方が静的型付け有利であるという主張がある。どういう点で動的型付けの方が有利なことがあり得るか、想像して書いてください。

解答の一例：より柔軟なプログラミングを行なう場合に有利なことがある。

- 型が「邪魔」になる場合、たとえば、
  - 「異なる型の要素を並べたリスト」を処理したい。
  - 「外部からやってくる (型のわからない) データ」を処理したい。
- 拡張性・再利用性
  - 将来、(設計時には思ってもいなかった) 拡張を行なう際の容易さ。
  - ある型のプログラムの一部を、別の型で使いたい。
- メタ・プログラミング (プログラムを生成するプログラム作り)

動的型のメリットについての参考図書: 「コードの世界」、まつもとゆきひろ (Ruby の設計者)、日経 BP 社, 2009 年。

目次

ML 言語の多相型 (復習)

map 関数の型:

```
map : (α → β) → (α list → β list)

map (fun x → x + 1) [1; 2; 3];;
=> [2; 3; 4]
map (fun x → x ^ ".pdf") ["a"; "b"; "c"];;
=> ["a.pdf"; "b.pdf"; "c.pdf"]
```


ML 言語の多相型

多相型 (polymorphic type): $\forall\alpha.\ t$

$$\mathrm{map} : \forall\alpha.\forall\beta.((\alpha \to \beta) \to (\alpha\ \mathrm{list} \to \beta\ \mathrm{list}))$$

α と β は型変数; どんな型をいれてもよい。

- $\alpha = \beta = \mathtt{int}$ とすると、$\mathrm{map} : (\mathtt{int} \to \mathtt{int}) \to (\mathtt{int\ list} \to \mathtt{int\ list})$
- $\alpha = \beta = \mathtt{string}$ とすると、
  $\mathrm{map} : (\mathtt{string} \to \mathtt{string}) \to (\mathtt{string\ list} \to \mathtt{string\ list})$
- $\alpha = \mathtt{int}$ および $\beta = \mathtt{string}$ とすると、
  $\mathrm{map} : (\mathtt{int} \to \mathtt{string}) \to (\mathtt{int\ list} \to \mathtt{string\ list})$

```
string_of_int 256 ;;
==> "256"
map string_of_int [10; 20; 30] ;;
==> ["10"; "20"; "30"]
```

ML 言語の多相型

ML 言語では、多相型は let 式で (のみ) 導入される。

```
let foo x = x ;;
==> foo : α → α
let goo x = [x; x; x] ;;
==> goo : α → α list
let hoo f x = f (f (f x)) ;;
==> hoo : (α → α) → (α → α)
```


let 多相の型システム

型の世界を拡張 (α は型変数をあらわす):

$$t ::= \texttt{bool} \mid \texttt{int} \mid t \to t \mid \alpha$$

多相型の導入 (let 式の規則を変更):

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma,\ x : (\forall\alpha_1.\cdots\forall\alpha_n.\ t_1) \vdash e_2 : t_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : t_2}$$

ただし、$\alpha_1, \cdots, \alpha_n$ は、t_1 に含まれる型変数のうち、Γ に (自由には) 現れない型変数を並べたリスト。

多相型の使用 (変数の規則を変更):

$$\frac{}{\Gamma \vdash x : t'}\ (x : \forall\alpha_1.\cdots\forall\alpha_n.\ t) \in \Gamma \quad \text{のとき}$$

ただし t' は t の中の α_i たちを (多相でない) 型 t_i たちで置きかえたもの

let 多相の型付け例

$$\frac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x.\ x : \alpha \to \alpha}$$

$$\frac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x.\ x : \alpha \to \alpha} \quad \dfrac{\dfrac{}{f : \forall\alpha.(\alpha \to \alpha) \vdash f : \texttt{int} \to \texttt{int}} \quad \dfrac{}{\cdots \vdash 5 : \texttt{int}}}{f : \forall\alpha.(\alpha \to \alpha) \vdash f\ 5 : \texttt{int}}}{\vdash \texttt{let}\ f = \lambda x.\ x\ \texttt{in}\ f\ 5 : \texttt{int}}$$

$$\frac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}}{\vdash \lambda x.\ x : \alpha \to \alpha} \quad \dfrac{\dfrac{}{f : \forall\alpha.(\alpha \to \alpha) \vdash f : \texttt{bool} \to \texttt{bool}} \quad \dfrac{}{\cdots \vdash 5 > 3 : \texttt{bool}}}{f : \forall\alpha.(\alpha \to \alpha) \vdash f\ (5 > 3) : \texttt{bool}}}{\vdash \texttt{let}\ f = \lambda x.\ x\ \texttt{in}\ f\ (5 > 3) : \texttt{bool}}$$

let 多相の演習問題

以下の関数に (できるだけ多相となる) 型を付けよ。

- `let foo x y = x`
- `let goo f x = f x`
- `let hoo f g x = g (f x)`

以下の最初の式は型が付くが、2 つ目は型が付かない。これを OCaml で試してみよ。また、この理由を説明せよ。

- `let f = fun x -> x in (f f) 10`
- `(fun f -> (f f) 10) (fun x -> x)`

ML 言語の多相型のまとめ

ML 言語の型システム (Hindley-Milner Type System):

- let 多相: let 束縛された変数に対して多相型を導入
- $\forall$ は型の一番外側のみ。($(\forall\alpha.\alpha) \to$ `int` などは考えない。)
- 式の型は一意的とは限らない

ML 言語における主型の存在

式 e と型文脈 Γ が与えられると、$\Gamma \vdash e : t$ となる型が存在する場合は、Principal Type (主型, そのような t のうち最も一般的な型) が存在する。

ML 言語の型推論アルゴリズム [Hindley-Milner]

式 e と型文脈 Γ が与えられたとき、$\Gamma \vdash e : t$ となる型 t が存在するかどうかを判定し、存在する場合は主型を返すアルゴリズムが存在する。

目次

発展的な話題

Sestoft 教科書 7 章の型推論の実装は、chap7.ml として coins マシン上にある。

Sestoft 教科書 (6.4.3):

ML の型推論の計算量

コア ML の型推論の計算量 (最悪計算量) は、与えられた式に対して指数時間 (EXPTIME) である。

```
let id x = x
let pair x y p = p x y
let p1 p = pair id id p
let p2 p = pair p1 p1 p
let p3 p = pair p2 p2 p
...
```