プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 7a: 型システム補足

型システムの例

整数、足し算、かけ算、条件式、等式だけがある言語：

$$e ::= i \mid e + e \mid e * e \mid e > e \mid \texttt{if } e \texttt{ then } e \texttt{ else } e$$

型:

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

型付けの例:

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

型検査 問題 (type checking)

入力: 式 e と 型 t

出力: Yes か No

入出力関係: $e : t$ 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 (変数と let 式)

前の言語に変数と let 式を追加した言語:

$$e ::= \cdots \mid x \mid \texttt{let } x = e \texttt{ in } e$$

型: 前と同じ

型付けの例:

$$(x+5)*6>y \ : \ \texttt{bool}$$
$$\texttt{if } x*6>y \texttt{ then } x+1 \texttt{ else } y+4 \ : \ \texttt{int}$$
$$\texttt{if } x*6>y \texttt{ then } x>1 \texttt{ else } y+4 \quad \text{型エラー}$$
$$\texttt{let } x=1+2 \texttt{ in let } y=x>5 \texttt{ in if } y \texttt{ then } x \texttt{ else } x+1 \ : \ \texttt{Int}$$

型システム

$e : t$ だけでは情報が足りない。

$$e_1 = x + 10 : \texttt{int}$$
$$e_2 = \texttt{if } x \texttt{ then } 10 \texttt{ else } 20 \ : \texttt{int}$$
$$e_3 = e_1 + e_2 \quad \text{型エラー}$$

e_1 と e_2 は単独では型がつくが、1 つのプログラムで両者を使うと型がつかなくなる。

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

型判定 $\Gamma \vdash e : t$

- $\Gamma = x_1 : t_1,\ x_2, t_2, \cdots, x_n : t_n\ (n \geq 0)$
- Γ は型文脈、e に含まれる「自由変数」たちの型付け

型システムの規則

$$\frac{(x{:}t) \in \Gamma}{\Gamma \vdash x : t}$$

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

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

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

型付けの推論

$$\dfrac{\dfrac{\dfrac{}{x : \texttt{int} \vdash x : \texttt{int}}}{x : \texttt{int} \vdash x + 1 : \texttt{int}} \quad \dfrac{\cdots}{x : \texttt{int}, y : \texttt{int} \vdash y > 10 : \texttt{bool}}}{x : \texttt{int} \vdash \texttt{let } y = x + 1 \texttt{ in } y > 10 : \texttt{bool}}$$

型検査と型推論

型検査:
型判定 $\Gamma \vdash e : t$ が成立するかどうかを判定する。

型推論:
型判定 $\Gamma \vdash e : t$ となる t があるかどうかを判定する。
あるいは、型判定 $\Gamma \vdash e : t$ となる t と Γ があるかどうかを判定する。

問題: $\texttt{let } x = 1 + 2 \texttt{ in let } y = x > 5 \texttt{ in if } y \texttt{ then } x \texttt{ else } x + 1$ を型推論せよ。

OCaml の型システム

型の世界がとても広がる

$$t ::= \texttt{bool} \mid \texttt{int} \mid t \to t \mid t * t \mid t\ \mathit{list} \mid \cdots$$

現代プログラミング言語の設計では、型システムの設計が極めて重要。

- 静的に安全性を保証できる (コンパイル時に型がつけば、実行中に型エラーを起こさない、つまり、整数を文字列に足したりしないこと保証される。)
- 型による抽象化が可能。
- 型システムを強化すれば更に強い信頼性の保証が得られることもある。

Ruby などは静的な型システムは持たないが、実行時には型の整合性を検査する。(整数を文字列に足するプログラムは、実行時に発見される。)