プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 10: プログラム言語と型システム

目次

1. 様々なプログラム言語の型システム
2. 静的型付け vs 動的型付け
3. 型検査 vs 型推論

型システムの分類のポイント

この資料では、以下の 2 つのポイントで比較する。

- 静的型付け vs 動的型付け vs 型付けなし
- 型検査 vs 型推論

目次

1. 様々なプログラム言語の型システム
2. 静的型付け vs 動的型付け
3. 型検査 vs 型推論

型付けなし 静的型付け vs 動的型付け

静的型付け: C,C++,ML,Haskell,Java,Scala,...

- プログラム実行前に (静的に)、型の整合性を検査
- 保守的近似で型を定める: (`if true then 1 else "a"`は型エラー)
- 高い安全性 (実行時エラーの削減), 型情報による最適化が可能

動的型付け: Python,Perl,Ruby,JavaScript,Lisp,...

- プログラム実行中に (動的に)、型の整合性を検査
- 実際に演算を行う際に型が整合していればよい:
- 安全性や最適化については静的型付けに劣ることが多い

型付けなし: 機械語など

静的型付けにおける基本定理

型安全性 (型システムの健全性) (Type Soundness, Subject Reduction)

- 「プログラム実行前に型が整合したら、実行中は、常に、型が整合する」という性質
- $\Gamma \vdash M : T$ が導出可能で、$M \hookrightarrow^* N$ ならば、$\Gamma \vdash N : T$ が導出可能。
- 成立しない場合、実行時にも型検査が必要 (動的型検査 $\Rightarrow$ 実行性能を落とす)
- 静的型付け言語すべてに対して成立することが期待される。

現実のプログラム言語:

- 広く使われているプログラム言語においては (機能がどんどん進化する等の理由により) 言語全体に対する型安全性の証明は研究課題。
- Standard ML, OCaml 等の中核部分については、型安全性は証明されている.

(補足) 静的型付けにおける基本定理

$$\frac{\Gamma \vdash \lambda x.M : T_1 \to T_2 \quad \Gamma \vdash N : T_1}{\Gamma \vdash (\lambda x.M)\ N : T_2} \hookrightarrow \Gamma \vdash M\{x := N\} : T_2$$

目次

1. 様々なプログラム言語の型システム
2. 静的型付け vs 動的型付け
3. 型検査 vs 型推論

明示的型付け (Church style) の体系

変数束縛において、その変数の型を明示するスタイル:
例. C 言語のプログラム

```
void foo (int x) {
  int i, j;
  float y;
  ...
}
```

Church style のプログラム言語:

- C, C++, Java, Scala, etc.

暗黙的型付け (Curry style) の体系

変数の束縛においてその変数の型を明示しないスタイル:
例. OCaml 言語のプログラム

```
let foo x =
  let i = 0 in
  let y = 3.14 in
  ...
```

Curry style のプログラム言語:

- ML 系言語, Haskell, etc.

Church style vs Curry style

比較:

- Church style では、プログラマの負担が大きいが、プログラマの意図が明示される。
- Church style では、システムの負担は小さい。
- Curry style では、プログラマの負担は小さいが、プログラマの意図は暗黙。
- Curry style では、システムの負担は大きい。

型検査と型推論

型検査 (type checking)

- 原則として Church style の体系に対するもの。
- 与えられたラムダ項と型に対して、型の整合性をチェックすること。
- 型文脈 Γ と項 M と型 T が与えられたとき、$\Gamma \vdash M : T$ が導けるかどうか？ (YES/NO 問題)

型推論 (type inference)

- 原則として Curry style の体系に対するもの。
- 与えられたラムダ項に対して、その型を推論しつつ型の整合性をチェックすること。
- 項 M が与えられたとき、$\Gamma \vdash M : T$ が導ける Γ と T があるか、また、ある場合はそれは何か？ (探索問題)

プログラム言語の型システム

型付け:

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

多相型:

	ML	Haskell	Java
パラメータ多相	let 多相	高階多相	Generics
サブタイピング多相	なし (*)	なし	あり
アドホック多相	なし	type class	overloading

(*) OCaml は Row 多相を持つ。