プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 1

目次

授業計画

- No.1 プログラム言語論について
- No.2 プログラム言語 OCaml 入門 [演習]
- No.3 ラムダ計算
- No.4 関数プログラミング [演習]
- No.5 単純型つきラムダ計算，型検査と型推論
- No.6 型推論アルゴリズム [演習]
- No.7 多相型
- No.8 多相型に対する型推論 [演習]
- No.9 オブジェクト指向，部分型
- No.10 発展的な話題，授業のまとめ [最終レポート]

目次

プログラム言語論とは？

内容: 種々のプログラム言語に共通する概念について考える.

- 構文と意味
- インタプリタとコンパイラ
- 抽象機械 (abstract machine)
- 型システム (type system)
- ...

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.
- I は解釈系 (実行系); `I(p,in) -> o`

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.
- I は解釈系 (実行系); `I(p,in) -> o`
- C は，プログラムを受け取って,「それを実行して答えを返すプログラム」を返すプログラム.

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.
- I は解釈系 (実行系); `I(p,in) -> o`
- C は，プログラムを受け取って,「それを実行して答えを返すプログラム」を返すプログラム.
- C は翻訳系; `C(p) -> q`; `q(in) -> o`

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.
- I は解釈系 (実行系); `I(p,in) -> o`
- C は，プログラムを受け取って,「それを実行して答えを返すプログラム」を返すプログラム.
- C は翻訳系; `C(p) -> q`; `q(in) -> o`

コンパイラ はそれだけ？

質問

「インタープリタ (I) とコンパイラ (C) はどう違うか？」

- I は，プログラムを受け取って，それを実行して答えを返すプログラム.
- I は解釈系 (実行系); `I(p,in) -> o`
- C は，プログラムを受け取って,「それを実行して答えを返すプログラム」を返すプログラム.
- C は翻訳系; `C(p) -> q`; `q(in) -> o`

コンパイラ はそれだけ？

- 型の整合性の検査を行う言語・処理系もある.

目次

質問

以下のプログラムを走らせるとどうなるか？

```
int f (int x) {
  if (x == 0)
    return 0;
  else if (x > 0)
    return f(x-1) + 2;
  else
    return "Error! negative number";
}
int main () {
  f(10);
}
```

質問

以下のプログラムを走らせるとどうなるか？

```
int f (int x) {
  if (x == 0)
    return 0;
  else if (x > 0)
    return f(x-1) + 2;
  else
    return "Error! negative number";
}
int main () {
  f(10);
}
```

型の整合性の警告

test1.c:9:12: warning: returning ‘ char * ’ from a function with return type ‘ int ’ makes integer from pointer without a cast [-Wint-conversion]

型ってなんだろう？

機械語:

- 「文字列の参照 (ポインタ)」だと思っていたデータを，整数と誤解したら，とんでもないエラーになる可能性がある．

型ってなんだろう？

機械語:

- 「文字列の参照 (ポインタ)」だと思っていたデータを，整数と誤解したら，とんでもないエラーになる可能性がある.

Ruby, Python, Perl, Lisp などの言語:

- プログラム中に `5+"abc"` とあっても，実行されない限りエラーでない.
- 実行されるとエラー (実行時エラー).

型ってなんだろう？

機械語:

- 「文字列の参照 (ポインタ)」だと思っていたデータを，整数と誤解したら，とんでもないエラーになる可能性がある.

Ruby, Python, Perl, Lisp などの言語:

- プログラム中に 5+"abc"とあっても，実行されない限りエラーでない.
- 実行されるとエラー (実行時エラー).

C, Java, ML, Haskell などの言語:

- 5+"abc"は型エラー (コンパイル時のエラー).
- プログラムを実行する事なく，潜在的なエラーの場所を指摘.

型システムの目的

型システムが実現したいこと:

型システムの目的

型システムが実現したいこと:
目的 1. 実行時エラー (システムがいきなり落ちる等) を未然に防止

- `5+"abc"` のような危ない演算の前に危ないかどうか検査.
- Ruby 等の処理系は実行時にこの検査をやる. (動的型付け)

型システムの目的

型システムが実現したいこと:
目的 1. 実行時エラー (システムがいきなり落ちる等) を未然に防止

- 5+"abc" のような危ない演算の前に危ないかどうか検査.
- Ruby 等の処理系は実行時にこの検査をやる. (動的型付け)

目的 2. 危険かもしれない部分を含むプログラムを実行前に検査.

- 5+"abc"のような危ない演算を含むプログラムを検査.
- C 言語等の処理系は実行前にこの検査をやる. (静的型付け)

型システムの目的(つづき)

目的 3. 関数やデータに情報を追加する.

- exch=「ドルを円に変換する関数」とする.
- exch(x)=x*137 (2022.7.11 現在)
- exch(exch(100)) は無意味なのでエラーにしたい.
- 「ドル表示の整数」と「円表示の整数」を違う型として定義すればよい.

型システムの目的(つづき)

目的 3. 関数やデータに情報を追加する.

- exch=「ドルを円に変換する関数」とする.
- exch(x)=x*137 (2022.7.11 現在)
- exch(exch(100)) は無意味なのでエラーにしたい.
- 「ドル表示の整数」と「円表示の整数」を違う型として定義すればよい.

目的 4. プログラムの「抽象化」の手段.

- 抽象化 (ここでの意味)==インタフェースと実装の分離
- 抽象データ型 [Liskov; Turing Award (2008)]

型システムとは？

B. Pierce, “Types and Programming Languages”, MIT Press, 2002. (型理論の著名な教科書，TaPL と略記)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

型システムとは？

B. Pierce, “Types and Programming Languages”, MIT Press, 2002. (型理論の著名な教科書，TaPL と略記)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

- 「プログラムの (良くない) 振舞いが発生しない」ことを証明する.

型システムとは？

B. Pierce, “Types and Programming Languages”, MIT Press, 2002. (型理論の著名な教科書，TaPL と略記)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

- 「プログラムの (良くない) 振舞いが発生しない」ことを証明する.
- 手に負える構文的な方法である

型システムとは？

B. Pierce, “Types and Programming Languages”, MIT Press, 2002. (型理論の著名な教科書，TaPL と略記)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

- 「プログラムの (良くない) 振舞いが発生しない」ことを証明する.
- 手に負える構文的な方法である
- (プログラムの) フレーズを分類することにより,

型システムとは？

B. Pierce, “Types and Programming Languages”, MIT Press, 2002. (型理論の著名な教科書，TaPL と略記)

> *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.*

- 「プログラムの (良くない) 振舞いが発生しない」ことを証明する.
- 手に負える構文的な方法である
- (プログラムの) フレーズを分類することにより,
- フレーズたちが計算する値の種類に従って

型と論理

「ある種の論理の世界と型の世界は同型である.」(Curry-Howard の対応)

型と論理

「ある種の論理の世界と型の世界は同型である.」(Curry-Howard の対応)

- 論理式 $A \supset A$ は証明可能.
- 型 $A \to A$ は要素を持つ.
  - 恒等関数.

型と論理

「ある種の論理の世界と型の世界は同型である.」(Curry-Howard の対応)

- 論理式 $A \supset A$ は証明可能.
- 型 $A \to A$ は要素を持つ.
  - 恒等関数.
- 論理式 $(A \wedge B) \supset (B \wedge A)$ は証明可能.
- 型 $(A \times B) \to (B \times A)$ は要素を持つ
  - 引数の左右を逆にしたものを返す関数.

型と論理

「ある種の論理の世界と型の世界は同型である.」(Curry-Howard の対応)

- 論理式 $A \supset A$ は証明可能.
- 型 $A \to A$ は要素を持つ.
  - 恒等関数.
- 論理式 $(A \wedge B) \supset (B \wedge A)$ は証明可能.
- 型 $(A \times B) \to (B \times A)$ は要素を持つ
  - 引数の左右を逆にしたものを返す関数.
- 論理式 $(A \supset B) \supset (B \supset C) \supset (A \supset C)$ は証明可能.
- 型 $(A \to B) \to (B \to C) \to (A \to C)$ は要素を持つ.,
  - 関数を 2 つもらって，それらの合成関数を返す高階関数.

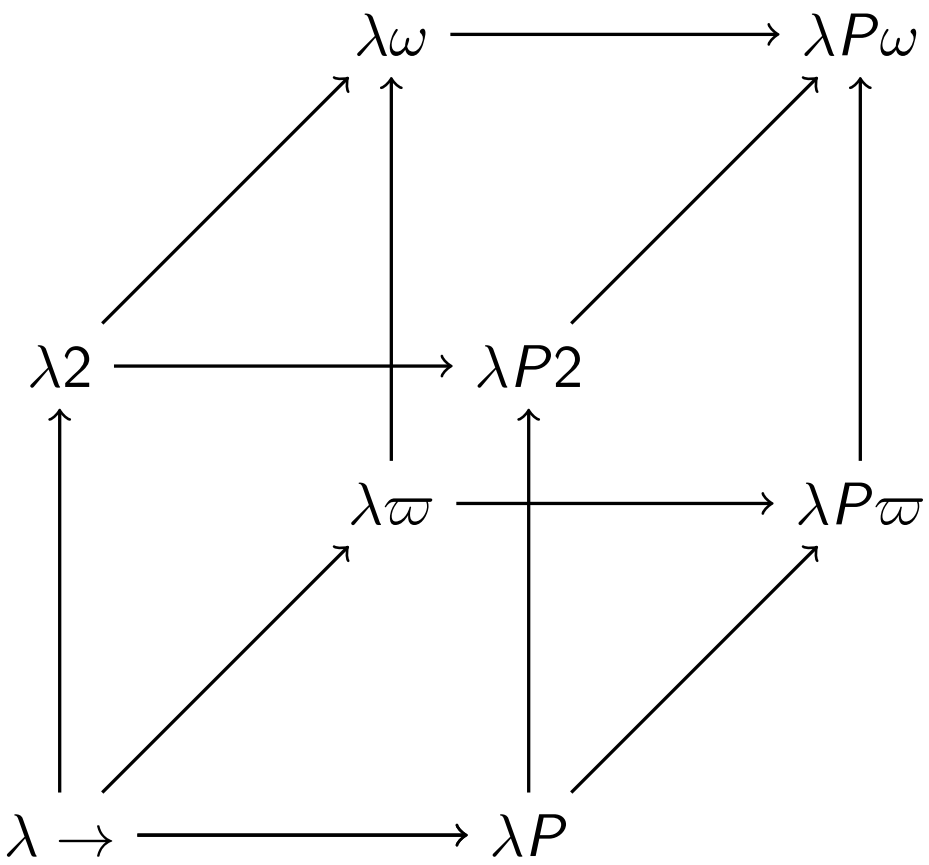
型と論理

「ある種の論理の世界と型の世界は同型である.」(Curry-Howard の対応)

- 論理式 $A \supset A$ は証明可能.
- 型 $A \to A$ は要素を持つ.
  - 恒等関数.
- 論理式 $(A \wedge B) \supset (B \wedge A)$ は証明可能.
- 型 $(A \times B) \to (B \times A)$ は要素を持つ
  - 引数の左右を逆にしたものを返す関数.
- 論理式 $(A \supset B) \supset (B \supset C) \supset (A \supset C)$ は証明可能.
- 型 $(A \to B) \to (B \to C) \to (A \to C)$ は要素を持つ.,
  - 関数を 2 つもらって，それらの合成関数を返す高階関数.

Curry-Howard の対応は，プログラミング言語の型システムの設計に多大な影響を与えてきた.

ラムダ・キューブ


$\lambda\omega$
$\lambda P\omega$
$\lambda 2$
$\lambda P2$
$\lambda\underline{\omega}$
$\lambda P\underline{\omega}$
$\lambda\to$
λP

まとめ

この授業では，プログラム言語の型システムを理解してそれを通して様々なプログラム言語について語るための概念を修得する.

- Java,C,TypeScript,ML,Haskell, ...

まとめ

この授業では，プログラム言語の型システムを理解してそれを通して様々なプログラム言語について語るための概念を修得する．

- Java,C,TypeScript,ML,Haskell, ...

演習では，特定のプログラム言語だけを一貫して使いたいので，型について議論しやすい言語を採用する．

まとめ

この授業では，プログラム言語の型システムを理解してそれを通して様々なプログラム言語について語るための概念を修得する．

- Java,C,TypeScript,ML,Haskell, ...

演習では，特定のプログラム言語だけを一貫して使いたいので，型について議論しやすい言語を採用する．

- OCaml (関数型プログラミング言語 ML の一族のひとつ)