計算論理：導入

亀山幸義 (kam@cs.tsukuba.ac.jp)
筑波大学
システム情報工学研究科コンピュータサイエンス専攻

講義の目的

- ソフトウェア科学
  ＝ソフトウェアを利用して何かをする
  のではなく、ソフトウェア自身が考察の対象
- プログラム理論
  ＝プログラムに対する基礎理論
- この授業
  - プログラム言語の基礎理論
  - 「型」が keyword

型システムとはなにか
型があると何がよいのか
型と論理の関係は？
型システムの応用は？

講義計画

1. 概要
2. 基礎編
   1. 現実のプログラム言語における型システム
   2. 型つきラムダ計算
   3. 型検査と型推論
   4. 型の健全性
3. 応用編
   1. 多相型
   2. オブジェクト指向
   3. コンパイラとCPS変換
4. まとめ

講義の題材

- プログラム言語
  - 例題を書くときは、C, Java, ML を使う。
  - 考察の対象とするのは、MLの小さなサブセット。
- λ計算（ラムダ計算）
  - 関数型言語のおもちゃ
  - Schemeに対応
  - 型がない！
- 型付きラムダ計算
  - これを使おう！

perl言語

```
#/usr/bin/perl
$c = "123";
$c = c + 1;
print " The first answer is $c \n";
#/usr/bin/perl
$c = "123";
$c = $c + 1;
print " The second answer is $c \n";
```

文字列型のデータを整数型に自動的に変換(cast)している。
→プログラムを書きやすい。
→プログラムの保守がしにくくなる。

scheme言語 (Lispの一種)

```
(let ((c 123))
   (set! c (+ c 1))
   c)
```

→ 124

```
(let ((c "123"))
   (set! c (+ c 1))
   c)
```

→ illegal argument...
型の整合性を動的に検査している

```
(define (f x) x)
((f f) 1)
```

→ 1

しかし、型の構造がどうなっているかわからない

Prolog言語

```
p(X,Y) ← father(X,F), brother(F,B), father(Y,B).
p(X,Y)←mother(X,M), brother(M,B), father(Y,B)
p(X,Y)←father(X,F),sister(F,S),mother(Y,B).
p(X,Y)←mother(X,M),sister(M,S),mother(Y,S).
father(mike, icihro).
mother(tama, hanako).
brother(hanako, ichiro).
p(mike,tama)←.
```

型の概念がない

C言語

- 豊富な基本型(basic type)を持っている
  - int, float, double, char
  - short X, long X (例 long int), signed X, unsigned X
  - enum X
- 型構成子 (type constructor)
  - 配列(array)
  - 構造体 (struct)
  - 共用体 (union)
  - ポインタ (pointer)
  - 関数
- 特徴的なこと
  - 型に名前が付けられる
  - 型の整合性をコンパイル時にチェックする
  - 型変換が可能　char *x;　(int *) x= ...

C言語

```
char daytable[2][13] = {
    {0,31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0,31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
}
char *s;
...
s = daytable[1];
...
t = (int *) s;
...
```

配列型とポインタ型
型変換

C言語その2

```
strcut tnode {
   char *word;
   struct tnode *left;
   struct tnode *right;
}

struct tnode *root;
root = NULL;
while (...) {
   root = addtree(root, word);
}
```

構造体 （名前つきの直積、レコード型）
自己参照型のポインタ

JAVA言語

```
public class rabbit extends turtle {
   static final short rabbitfig = ...;
   void setrabbit() {
     kame = rabbitfig; ...
   };
   public rabbit() {
    super();
   setrabbit();
   };
   ..
}
```

クラスを型と見なすと、型が完全に一致していなくても(継承されたクラスなら)代入ができる　－＞　subtype　の必要性

「現実」のプログラム言語

- C
  - オーソドックスな言語
  - 「細かい操作が何でもできる」が、かえって危ない(ポインタ操作など)
- Java
  - オブジェクト指向
  - 型に関して「安全」な言語
  - 細かい操作(危ない)操作はあまりできない
  - 機械語レベルでの安全性検証つき
- ML, Haskell etc.
  - 関数型
  - 高度な型システム
  - 通常の言語と同様な代入文等も書ける
- そのほか
  - Scheme/Lisp; 関数の概念はあるのである程度、型を考えることは可能
  - perl、PHP　などのスクリプト言語; 問題外

型(の整合性)と言語

- 強く型付けされた言語
  - ML, Java [型の整合性を保証]
  - C [いい加減な型も書ける]
- 弱く型付けされた言語
  - Scheme, Lisp
- 型の整合性の概念がない言語
  - (多くの)機械語、perlなどのスクリプト言語、
  - (普通の)Prolog

プログラム言語の抽象化(理想化)

- この授業での考察対象は、現実のプログラム言語ではなく、抽象化したプログラム言語
- 利点
  - 個別のプログラム言語の些細な違いにこだわることなく、本質的な点のみに集中することができる
- 欠点
  - 抽象化したプログラム言語での理屈(理論)を現実のプログラム言語にそのまま適用することはできない
- この授業では、「型付きλ計算」を採用

応用編の話題

- 多相型
  - 型変数を持つ型
  - map: $(\alpha \to \beta) \to (\mathrm{list}(\alpha) \to \mathrm{list}(\beta))$
  - sort: $(\alpha \to \alpha \to \mathrm{Bool}) \to (\mathrm{list}(\alpha) \to \mathrm{list}(\alpha))$
- オブジェクトの型(クラスに対応する型)
  - クラスがあれば型はいらない?
  - クラスの継承関係の表現
  - サブタイピング (部分型)
- CPS変換
  - CPS = Continuation Passing Style とはなにか?
  - コンパイラとの関係
  - 型システムとどう関係するのか?

スケジュール

- 講義:12/4, 11, 18(演習の予定)、
  1/8, 15, 22(演習の予定)、
  1/29, 2/5(一部演習の予定)、
  2/12, 19(期末試験)
  3/5 は何もやりません!
  途中1回前後休講の可能性あり
- 評価: 演習+期末試験(+出席)
  2回までの欠席はノーカウント、それを超えると減点。

参考書

- 一番参考になるもの
  - B. Pierce, "Types and Programming Languages", MIT Press, 2002.
- そのほか
  - 大堀淳、「プログラム言語の基礎理論」、共立出版 情報数学講座9
  - 萩谷昌己「ソフトウェア科学のための論理学」, 岩波講座ソフトウェア科学11, 岩波書店

講義に関する情報源

web page
http://logic.cs.tsukuba.ac.jp/~kam/complogic2008/

問合せ
kam @ cs . tsukuba . ac . jp
総合研究棟B棟1008, 1027

大事なこと

- 講義に欠席したときの内容については、自力で補ってください。
  - 大事なこと（レポート出題等）だからといって2回も3回もいわないことがあります。
- メールによるレポート提出等の際は、必ず、返事をします。
  - 5日程度以内に返事が来なければもう一度確認のメールを送ってください。
  - 確認を怠って、（メールがなくなったために）単位が取れない場合、本人の責任とします。