コンパイラ演習
第 7 回
(2011/11/24)

中村 晃一　野瀬 貴史　前田 俊行
秋山 茂樹　池尻 拓朗
鈴木 友博　渡邊 裕貴
潮田 資秀
小酒井 隆広
山下 諒蔵　佐藤 春旗
大山 恵弘　佐藤 秀明
住井 英二郎

今日の内容

- Type Polymorphism (型多相性)の実現について

Polymorphism の種類

- 大きく分けて 3 種類ある
  - Parametric polymorphism
  - Subtyping polymorphism
  - Ad-hoc polymorphism (overloading)

Parametric Polymorphism の例: OCaml

```
# let f x = x;;
val f : 'a -> 'a = <fun>
# f 3;;
- : int = 3
# f 3.14;;
- : float = 3.14
```

型変数

'a を int で置換

'a を float で置換

Parametric Polymorphism の例: C++

```
template <class T>
class stack {
  T *v, *p; int sz;
public:
  stack(int s) { v = p = new T[sz=s]; }
  void push(T a) { *p++ = a; }
  T pop() { return *--p; }
  int size() const { return p - v; }
};
```

型変数

Parametric Polymorphism を実現する上での問題

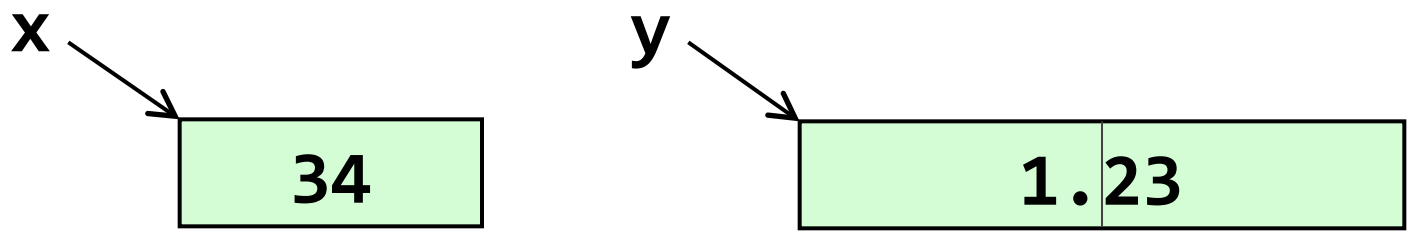
- Polymorphic な変数のサイズを
  コンパイル時に決定できない場合がある
- 例: `let f x y = (x, y)`
  - x と y は 16 bit? 32 bit? 64 bit? 128 bit?
  - x と y はどのレジスタで渡される?
    - 整数レジスタ? 浮動小数点数レジスタ?
  - 作られるタプルのサイズは?

Parametric Polymorphism の実装法

- ここでは 3 通り説明する
  - Boxing
  - Expansion (Function Cloning)
  - Type-passing

Boxing

- 全ての変数が参照を保持するようにする
  - よって全ての変数は同じサイズ (例: 32 bit)
- データは参照の先の “box” の中に置く



- 単純な実装ではオーバヘッドが大きい
  - 「関数をまたがない範囲で受け渡される値には box を作らない」などの最適化が考えられる

Expansion (Function Cloning)

- 関数の呼出し箇所における実引数の型に応じて関数の複製を作るようにする

```
let f x = x
let t = (f 2, f 3.4)
```

```
let f_int (x : int) = x
let f_float (x : float) = x
let t = (f_int 2, f_float 3.4)
```

Type-passing

- 型情報を実行時に受け渡すようにする

```
let f x = [|x|]
let t = (f 2, f 3.4)
```

```
let f {T} (x : T) =
  if {T} = {int} then ...
  else if {T} = {float} then ...
  else ...
let t = (f {int} 2, f {float} 3.4)
```

変数 x の型を表す変数

型 int を表す値

型 float を表す値

Subtyping Polymorphism の例: OCaml

```
# let f p = p#get_x + 1;;
val f : < get_x : int; .. > -> int
      = <fun>
# let p = object
    method get_x = 0
    method get_y = 0
  end;;
val p : < get_x : int; get_y : int >
      = <obj>
# f p;;
- : int  = 1
```

※より厳密には OCaml では subtyping polymorphism ではなく row polymorphism

Subtyping Polymorphism の例: C++

```
void print_xy(Point *p) {
  cout << p->x << endl;
  cout << p->y << endl;
}

int main(void) {
  Point *p = new Point(12, 34);
  ColorPoint *cp = new ColorPoint(7, 8, 9);
  print_xy(p);
  print_xy(cp);
  return 0;
}
```

ここで ColorPoint は Point の subtype とする

Point * を受け取る関数を ColorPoint * で呼び出せる

Subtyping Polymorphism の実装

- 簡単にやるには、subtype 関係で型が変わるところに値を変換するコードを挿入すればよい

```
void print(Point *p);
ColorPoint *cp = new ColorPoint(...);
print(cp);
```

```
print(convert(cp));
```

- もっと真面目に考えたいときは、たとえば
  Garrigue, J. “Programming with polymorphic variants” (ML Workshop 1998) などを参照

Ad-hoc Polymorphism の例: C, C++

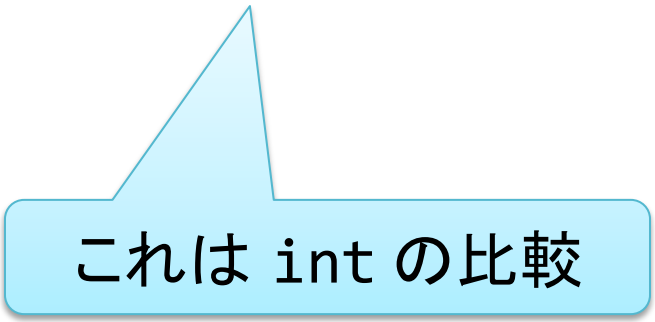
```
int plus_i(int x, int y) {
  return x + y;
}

double plus_f(double x, double y) {
  return x + y;
}
```

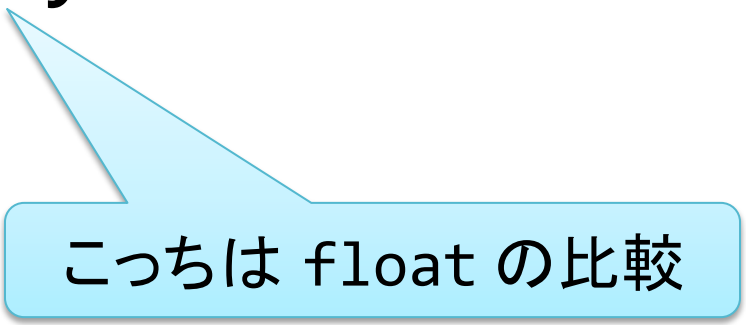
これは int の加算

こっちは double の加算

Ad-hoc Polymorphism の例: OCaml

```
let f x y =
    1 = x && 1.23 = y
```





Ad-hoc Polymorphism の実装

- 簡単にやるには
  expansion のようなことをすればよい
  - 型が分かっていれば
    どの関数を呼べばよいか静的に分かる
- もっと真面目にやる場合
  - Haskell (次ページ以降で簡単に紹介)
    - P. Wadler and S. Blott.
      “How to make ad-hoc polymorphism less ad hoc.” (POPL ‘89) など
  - G'Caml
    - http://web.yl.is.s.u-tokyo.ac.jp/~furuse/gcaml/
  - $'Caml
    - http://jun.furuse.info/hacks/discaml

Haskell での ad-hoc polymorphism: type class

```
class Num a where
   (+), (*) :: a -> a -> a
   negate   :: a -> a

instance Num Int where
   (+) = addInt
   (*) = mulInt
   negate = negInt

instance Num Float where
   (+) = addFloat
   (*) = mulFloat
   negate = negFloat

square   :: Num a => a -> a
square x =  x * x
```

これが type class:
(+) と (*) と negate が適用可能な型の集まりを表している

Type class の instantiation:
型「**Int**」を type class「**Num**」に属す型であると宣言している

Type class の instantiation:
型「**float**」を type class「**Num**」に属す型であると宣言している

型「**Int**」と型「**float**」で異なる関数を用いて instantiate できる
⇒ Ad-hoc polymorphism

関数**square**は型クラス「**Num**」に属す任意の型を引数にとることを表している

型**a**が**Int**なら**mulInt**が、
floatなら**mulFloat**が呼ばれる

Type class の簡単なコンパイル方法

```
class Num a where
   (+), (*) :: a -> a -> a
   negate   :: a -> a

instance Num Int where
   (+) = addInt
   (*) = mulInt
   negate = negInt

instance Num Float where
   (+) = addFloat
   (*) = mulFloat
   negate = negFloat

square   :: Num a => a -> a
square x =  x * x
```


```
type 'a numd = NumD of
   ('a -> 'a -> 'a) *
   ('a -> 'a -> 'a) *
   ('a -> 'a)

let add (NumD (a, m, n)) = a
let mul (NumD (a, m, n)) = m
let neg (NumD (a, m, n)) = n

let numDInt   = NumD
   ((+) , ( * ), (~-) )
let numDFloat = NumD
   ((+.), ( *.), (~-.))

let square numDa x =
   mul numDa x x
```

共通課題

- 今回は全 2 問のうちどちらかを解けばよい
  - もちろん両方解いてもよい

共通課題 1

- OCaml, SML, Haskell, C++, Java などの
  既存の処理系を二つ選び
  それぞれの処理系において
  parametric polymorphism が
  どのように実装されているか説明せよ
  - コンパイルなどの実験の結果をもとに説明しても、文書やコンパイラのソースコードを読んでの理解をもとに説明してもよい

共通課題 2

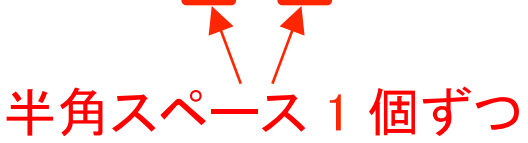
- MinCaml を改造して
  overload された演算子を一つ以上導入せよ
  - typing.ml の型推論結果を利用してよい
  - 必要があれば字句・構文解析器も拡張
  - 改造したソースコードを送ってください
  - たとえば…
    - 整数と浮動小数点数の両方に使える加算演算子
    - 符号反転にも真偽値の反転にも使える演算子
    - 整数乗算にも整数配列の内積にも使える演算子

コンパイラ係向け課題

- Parametric polymorphism を
  自作コンパイラまたは MinCaml に実装せよ

課題の提出先と締め切り

- 提出先: compiler-report-2011@yl.is.s.u-tokyo.ac.jp
- 締め切り: 2 週間後 (12/08) の午後 1 時
  - コンパイラ係用課題の締め切り: 2012 年 2月 27日
- Subject: **Report␣7␣**<学籍番号:5 桁>



  - 例: **Report 7 11099**
- 本文にも氏名と学籍番号を明記のこと
  - 質問は compiler-query-2011@yl.is.s.u-tokyo.ac.jp まで