プログラミング言語設計論

2014年度

第7回: 型システム

担当: 増原英彦

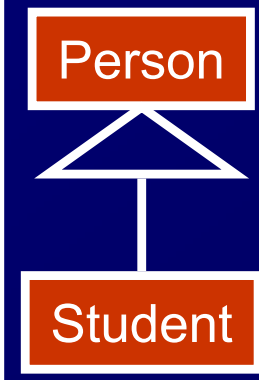
問題1 (15分)

1. 「型」を定義せよ
2. 「(静的)型検査」を定義せよ
3. 「型安全性」を定義せよ
4. 型検査に失敗するプログラムの例を任意の言語で1つ示せ
5. 型安全性が保証されていない言語を1つ挙げ、その根拠となるプログラムを1つ示せ

問題2 (5分)

1. コンパイルするとどうなるか?
2. (コンパイルできた場合)実行するとどうなるか?

```
// Java
void setPerson(Person[] pp, int i, Person p) {
  pp[i]=p;
}
void main(){
  Student[] ss = new Student[1];
  setPerson(ss, 0, new Student(“Alice”));
}
```



用語

- 型: プログラムが扱うデータの集合
  - 例: 整数、実数、Personクラス...
- 型誤り(型エラー): ある操作に与えられるデータが、扱えない型に入っていること
  - 例: 1 + “hello”, sqrt(“hello”), 123.getName()
- (静的)型検査: プログラムを実行せずに型誤りの有無を確認すること
- 型安全性: 静的型検査に合格したプログラムは、実行したときに型誤りが絶対に起こさないこと

古典的な言語の型システム

- 静的な検査をしない
  - 「動的型付言語」と呼ばれる
  - Lisp, Perl, Ruby, Python, Smalltalk, Javascript...
  - データに型を付けて実行中に検査
- 不完全ながら静的検査をする
  - COBOL, FORTRAN, Algol, Pascal, C, ...
  - 不完全な部分
    - ◆キャスト: float f = 123.45; int *i = (int*)&f;
    - ◆配列境界: float f[1]; int i[1]; f[2]=123.45;
    - ◆併合型: union { int i; float f; } u; u.f=123.45;

なぜ不完全?
•難しい
•あえて許す
•遅くなる

古典的な言語の型検査

ここでは式と文を
あわせて「式」という

- 型検査規則
  - 規則: 式の種類ごとに部分式および結果の許される型の組み合わせ
    - 例: if e1 then e2 else e3 ➔ e1の型はbool, e2とe3と結果の型は同一
  - 検査方法: 式に型付けをして、規則を使って確認
- 型付け
  - 変数・関数の型はプログラムに宣言してある
  - 型規則を「部分式の型から結果の型を求める」と読みかえる if e1 then e2 else e3 ➔ 結果の型はe2の型

型安全な言語設計の難しさ

- 仮想言語TypeSafeの仕様
  - データ型: int, double, string, boolean, array of 型, record 名前 of { ラベル: 型; ラベル: 型; ... }
    - 例えば "array of int" や"record pt of { x: int; y: int }"
  - 文法:
    定義 ::= 型 関数名(型: 変数;...) { 文; 文;...}
    文 ::= 代入|変数宣言|関数呼出|if文|while文|return文|式文
- このような言語の型安全を保証することは難しくない
- このような言語は何が不足しているのか?

難しさ: 複数の型を扱うコード (型多相性)

- 手続き
  - 「整数の配列でも文字列の配列でも内容をランダムに並べかえる手続き」
  - 「比較ができる型だけの配列を整列する手続き」
  - Personクラスを受け取るメソッドにStudentを渡す
- 抽象データ型
  - 「整数だけのスタックと文字列だけのスタック」

これらを型安全にする検査方法は単純でない

難しさ: 複数の型を扱うコード (型多相性)

◆例: if e1 then e2 else e3 ➔ e1の型はbool, e2とe3と結果の型は同一

➢ 検査方法: 式に型付けをして、規則を使って確認

■ 型付け

➢ 変数・関数の型はプログラムに宣言してある

➢ 型規則を「部分式の型から結果の型を求める」と読みかえる if e1 then e2 else e3 ➔ 結果の型はe2の型

➢「整数だけのスタックと文字列だけのスタック」

これらを型安全にする検査方法は単純でない

多相性へのアプローチ: 部分型と型検査

- 部分型: 値の集合が型Aの部分集合になっているような型を型Aの部分型という
  - 部分型による多相性: ある型に対して作られたコードは、部分型の値についても使える
- 型検査
  - ある型を期待している変数に、その部分型の値を渡すことを許す (例:Personを印刷する手続きにStudentを渡す)
  - 異なる型の値を混ぜる結果ができる場合は、それらの型の最小上界の型にする (例: if … then new Student() else new Teacher()の結果はPerson型)

部分型と複合型

- 複合型(compound type):
  別の型を組み合わせて作られる型
  - 例:「整数型を要素する配列型」「Personを要素とする集合」「数を受け取りPersonを返す関数」
- 複合型の部分型関係: 自明でない
  - 例: Student <: Personのとき
    - Student[] <: Person[] — covariant という
    - Student[] :> Person[] — contravariant
    - Student[] ≠ Person[] — invariant

部分型と代入

- class Cell { //Personをしまうセル
  Person p;
  void set(Person p) { this.p = p; }
  Person get() { return this.p; }
  }
- これをTeacherをしまうセルとして使えるか?

```
Cell teacherCell = new Cell();
teacherCell.set(new Teacher());
teacherCell.get().teach(aClass);
```

Personには teachがない!

- (古いJavaの)解決: get()の返値をTeacherにキャスト → 型安全でない

Javaの配列: covariantだが安全でない

- 下のプログラムは型検査に合格する
  - void set(Person[] cell, Person p) {
    cell[0]=p;
    }
  - Teacher[] tCell = new Teacher[1];
    set(tCell, new Student());
  - 理由: JavaではT <: P ならば T[] <: P[]
- 起きること: 実行するとここで「Teacher配列にStudentを代入する」例外
- 何故安全でない仕様になっているのか?

解決: テンプレート

- テンプレート: 型を適用するとクラス定義や関数定義を作るような定義 (C++)
  - 例:

```
template <typename T>
class Stack {
   T[] elements;
   void push(T elm) { … }
   T pop() { … } };
```

  - Stack<Student> s = new Stack<Student>();
- 型ごとにコードができる→部分型なしの多相性

テンプレートの問題

このコードの問題を見つけよ

```
class Person {};
int main () {
  std::vector<int> is = { 3,1,4,1,5,9,2 };
  std::sort(is.begin(), is.begin()+7);

  std::vector<Person> pp = { Person(), Person() };
  std::sort(pp.begin(), pp.begin()+1);
}
```

```
In file included from /usr/include/c++/4.6/algorithm:63:0,
                 from v.cpp:2:
/usr/include/c++/4.6/bits/stl_algo.h: In function 'void std::__insertion_sort(_RandomAccessIterator,
_RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<Person*,
std::vector<Person> >]':
/usr/include/c++/4.6/bits/stl_algo.h:2181:4:   instantiated from 'void
std::__final_insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator =
__gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
/usr/include/c++/4.6/bits/stl_algo.h:5409:4:   instantiated from 'void std::sort(_RAIter, _RAIter) [with _RAIter =
__gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
v.cpp:8:37:   instantiated from here
/usr/include/c++/4.6/bits/stl_algo.h:2107:4: error: no match for 'operator<' in '__i.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with
_Iterator = Person*, _Container = std::vector<Person>, __gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]() <
__first.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = Person*, _Container = std::vector<Person>,
__gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]()'
/usr/include/c++/4.6/bits/stl_algo.h:2107:4: note: candidates are:
/usr/include/c++/4.6/bits/stl_pair.h:207:5: note: template<class _T1, class _T2> constexpr bool std::operator<(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
/usr/include/c++/4.6/bits/stl_iterator.h:291:5: note: template<class _Iterator> bool std::operator<(const std::reverse_iterator<_Iterator>&, const
std::reverse_iterator<_Iterator>&)
/usr/include/c++/4.6/bits/stl_iterator.h:341:5: note: template<class _IteratorL, class _IteratorR> bool std::operator<(const std::reverse_iterator<_IteratorL>&, const
std::reverse_iterator<_IteratorR>&)
/usr/include/c++/4.6/bits/stl_iterator.h:1049:5: note: template<class _IteratorL, class _IteratorR> bool std::operator<(const std::move_iterator<_IteratorL>&, const std::move_iterator<_IteratorR>&)
/usr/include/c++/4.6/bits/stl_iterator.h:1055:5: note: template<class _Iterator> bool std::operator<(const std::move_iterator<_Iterator>&, const std::move_iterator<_Iterator>&)
/usr/include/c++/4.6/bits/stl_vector.h:1290:5: note: template<class _Tp, class _Alloc> bool std::operator<(const std::vector<_Tp, _Alloc>&, const std::vector<_Tp, _Alloc>&)
/usr/include/c++/4.6/bits/basic_string.h:2510:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const std::basic_string<_CharT, _Traits, _Alloc>&)
/usr/include/c++/4.6/bits/basic_string.h:2522:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const _CharT*)
/usr/include/c++/4.6/bits/basic_string.h:2534:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const _CharT*, const std::basic_string<_CharT, _Traits, _Alloc>&)
/usr/include/c++/4.6/tuple:586:5: note: template<class ... _TElements, class ... _UElements> bool std::operator<(const std::tuple<_TElements ...>&, const std::tuple<_Elements ...>&)
/usr/include/c++/4.6/bits/stl_algo.h: In function 'void std::__heap_select(_RandomAccessIterator, _RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]':
/usr/include/c++/4.6/bits/stl_algo.h:5254:7:   instantiated from 'void std::__partial_sort(_RAIter, _RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
/usr/include/c++/4.6/bits/stl_algo.h:2279:8:   instantiated from 'void std::__introsort_loop(_RandomAccessIterator, _RandomAccessIterator, _Size) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >, _Size = int]'
/usr/include/c++/4.6/bits/stl_algo.h:5407:4:   instantiated from 'void std::sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
v.cpp:8:37:   instantiated from here
/usr/include/c++/4.6/bits/stl_algo.h:1905:2: error: no match for 'operator<' in '__i.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = Person*, _Container = std::vector<Person>, __gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]() <
__first.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = Person*, _Container = std::vector<Person>, __gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]()'
/usr/include/c++/4.6/bits/stl_algo.h:1905:2: note: candidates are:
/usr/include/c++/4.6/bits/stl_pair.h:207:5: note: template<class _T1, class _T2> constexpr bool std::operator<(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
/usr/include/c++/4.6/bits/stl_iterator.h:291:5: note: template<class _Iterator> bool std::operator<(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_Iterator>&)
/usr/include/c++/4.6/bits/stl_iterator.h:341:5: note: template<class _IteratorL, class _IteratorR> bool std::operator<(const std::reverse_iterator<_IteratorL>&, const std::reverse_iterator<_IteratorR>&)
/usr/include/c++/4.6/bits/stl_iterator.h:1049:5: note: template<class _IteratorL, class _IteratorR> bool std::operator<(const std::move_iterator<_IteratorL>&, const std::move_iterator<_IteratorR>&)
/usr/include/c++/4.6/bits/stl_iterator.h:1055:5: note: template<class _Iterator> bool std::operator<(const std::move_iterator<_Iterator>&, const std::move_iterator<_Iterator>&)
/usr/include/c++/4.6/bits/stl_vector.h:1290:5: note: template<class _Tp, class _Alloc> bool std::operator<(const std::vector<_Tp, _Alloc>&, const std::vector<_Tp, _Alloc>&)
/usr/include/c++/4.6/bits/basic_string.h:2510:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const std::basic_string<_CharT, _Traits, _Alloc>&)
/usr/include/c++/4.6/bits/basic_string.h:2522:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const std::basic_string<_CharT, _Traits, _Alloc>&, const _CharT*)
/usr/include/c++/4.6/bits/basic_string.h:2534:5: note: template<class _CharT, class _Traits, class _Alloc> bool std::operator<(const _CharT*, const std::basic_string<_CharT, _Traits, _Alloc>&)
/usr/include/c++/4.6/tuple:586:5: note: template<class ... _TElements, class ... _UElements> bool std::operator<(const std::tuple<_TElements ...>&, const std::tuple<_Elements ...>&)
/usr/include/c++/4.6/bits/stl_algo.h: In function 'void std::__move_median_first(_Iterator, _Iterator, _Iterator) [with _Iterator = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]':
/usr/include/c++/4.6/bits/stl_algo.h:2252:7:   instantiated from '_RandomAccessIterator std::__unguarded_partition_pivot(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
/usr/include/c++/4.6/bits/stl_algo.h:2284:54:   instantiated from 'void std::__introsort_loop(_RandomAccessIterator, _RandomAccessIterator, _Size) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >, _Size = int]'
/usr/include/c++/4.6/bits/stl_algo.h:5407:4:   instantiated from 'void std::sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<Person*, std::vector<Person> >]'
v.cpp:8:37:   instantiated from here
/usr/include/c++/4.6/bits/stl_algo.h:86:7: error: no match for 'operator<' in '__a.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = Person*, _Container = std::vector<Person>, __gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]() <
__b.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = Person*, _Container = std::vector<Person>, __gnu_cxx::__normal_iterator<_Iterator, _Container>::reference = Person&]()'
/usr/include/c++/4.6/bits/stl_algo.h:86:7: note: candidates are:
```

テンプレートの問題

このコードの問題を見つけよ

```
class Person {};
int main () {
  std::vector<int> is = { 3,1,4,1,5,9,2 };
  std::sort(is.begin(), is.begin()+7);

  std::vector<Person> pp = { Person(), Person() };
  std::sort(pp.begin(), pp.begin()+1);
}
```

sortは要素に<が定義されていることを仮定
sortの型では分からない

```
template <class RandomAccessIterator>
  void sort (RandomAccessIterator first, RandomAccessIterator last);
```

解決: 総称(generic)型

- 総称型: 型変数を含んだ合成型 (Java)
  - 例:

```
class Stack<T> {
   T[] elements;
   void push(T elm) { … }
   T pop() { … } };
```

  - Stack<Student> s = new Stack<Student>();
- テンプレートとの違い
  - Stackのコンパイルは1回だけ (cf.テンプレートは使用する型ごとに展開)
  - Stackの定義と、Stackの使用を別々に検査できる (cf.テンプレートは展開してから誤りが判明)

総称型の型制約

- 型に制約を付けられる
  - 例

```
interface Comparable { boolean compare( ... ) ; }
class Algorithm<T extends Comparable> {
  void sort(Collection<T> lst) { ... } }
```

  - class Person {}

```
Algorithm.sort(new Vector<Person>());
```

T <: Comparableを要求
→Tにはcompareがあることを要求

Person <: Comparable でないのでエラー
→compareがないことを確認できる

総称型にも部分型がほしい

- 例
  - class Cell<T> { //Tをしまうセル
    T p;
    void set(T p) { this.p = p; }
    T get() { return this.p; }
    }
  - void getAndPrint(Cell<Person> cellP) {
    print(cellP.get().getName()); }
  - Cell<Student> c = new Cell<Student>();
    c.set(new Student());
    getAndPrint(c);
- このようなことを許したいか?
- 許すためにはどんな部分型の規則が必要?
- その規則に問題はないか?

総称型をcovarintにしたら?

- 例
  - class Cell<T> { //Tをしまうセル
    T p;
    void set(T p) { this.p = p; }
    T get() { return this.p; }
    }
  - void work(Cell<Person> cellP) {
    print(cellP.get().getName()); }
  - Cell<Student> c = new Cell<Student>();
    c.set(new Student());
    work(c);
- covariant — S <: P ならば Cell<S> <: Cell<P>
  上の例は型検査を通過
- workの定義を以下にすると･･･?
  void work(Cell<Person> cellP) {
  cellP.set(new Teacher()); }

Javaの解決: ワイルドカード

- 例

```
void getAndPrint(Cell<? extends Person> cellP) {
    print(cellP.get().getName()); }
```

- 意味: Cellの要素はPersonの部分型の1つ
  - cellP.get()の結果の型は確定できないが、Personの部分型であることは分かる
    →.getName()は正しい
  - cellP.set(new Student());としたら?
    →cellP.setの引数はPersonの部分型の1つ
    →Studentとは限らないのでエラー

Javaの解決: ワイルドカード

- 同様に次のような制約もある

```
void work(Cell<Student super ?> cellP) {
  cellP.set(new Student());
}
```

- 意味は?
  - StudentがCellの要素型の部分型になっている(contravariant)
- 使い途は?
  - Cell<Person> cp = new Cell<Person>();
    work(cp);
  - cellPの要素型Tは確定できないが、Studentはその部分型
  - cellP.setの引数はT→Studentを渡すのは正しい!

Javaの総称型の型検査

- 難しい
  - F-bounded多相性を使った「存在型」に対応

    Igarashi, Atsushi, and Mirko Viroli. "Variant parametric types: A flexible subtyping scheme for generics." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 28.5 (2006): 795-847.