CSC 548: Lecture 5

Overview

OO languages

Single vs multiple inheritance

Casting / typecase

Generics

OO languages

What features make a language OO?

What is the difference between subtyping and subclassing?

Subtyping in Hobbes

We extend Hobbes with support for subtyping:

```
class Foo { method foo () {} }
class Bar extends Foo { method bar () {} }
```

Single inheritance

What is single inheritance? Does Java have single inheritance? C++? Hobbes?

What is a vtable? For languages with single inheritance, how is the vtable laid out?

Hobbes has static and dynamic method dispatch... What difference does this make to the machine code generated?

Multiple inheritance

What is multiple inheritance? Does Java have multiple inheritance? C++? Hobbes?

How does multiple inheritance affect vtable layout?

Possible solutions to vtable layout:

1: Global graph coloring.

2: Hashing.

3: C++ vtable layout (not discussed by Appel!).

4: Sparse arrays (also not discussed by Appel!).

What are the tradeoffs?

Implementing single inheritance in Hobbes

This is the homework!

What changes need to be made to the Hobbes compiler?

Casting / typecase

What is casting / typecase?

How can we implement casting / typecase?

Generics

What are generics? Does Java have generics? C++?

One approach to generics is *parametric polymorphism*: what is this? What are other approaches?

Generics

Imagine Hobbes had generics:

```
interface List<e> {
  method head () : e;
  method tail () : List<e>;
  method size () : Integer;
  method cons (hd : e) : List<e>;
}
class Empty<e> implements List<e> { ... }
class Cons<e> implements List<e> { ... }
object empty<e> : Empty<e> { }
```

so we can write:

```
let x : List<String> = empty<String>.cons ("hello").cons ("world");
let y : List<Integer> = empty<Integer>.cons (1).cons (2);
```

What needs changed in Hobbes?

Generics

What about base types? Should we be allowed List<Integer>?

What about List<Double>?

Generics

Possible representations at run-time:

1. Expansion.

2. Full boxing/tagging.

3. Coercions.

4. Type-passing.

How do these work? What are the tradeoffs?

Summary

OO languages support subtyping, subclassing and dynamic method dispatch.

Implementing single inheritance is not too bad. Multiple inheritance is difficult!

Generics in OO languages are supported by parametric polymorphism: this has interesting interactions with the rest of the language!

Next week: Midterm

Week after: Functional SSA form