Polymorphic Types

Lucy Wilcox and Xiaofan Wu

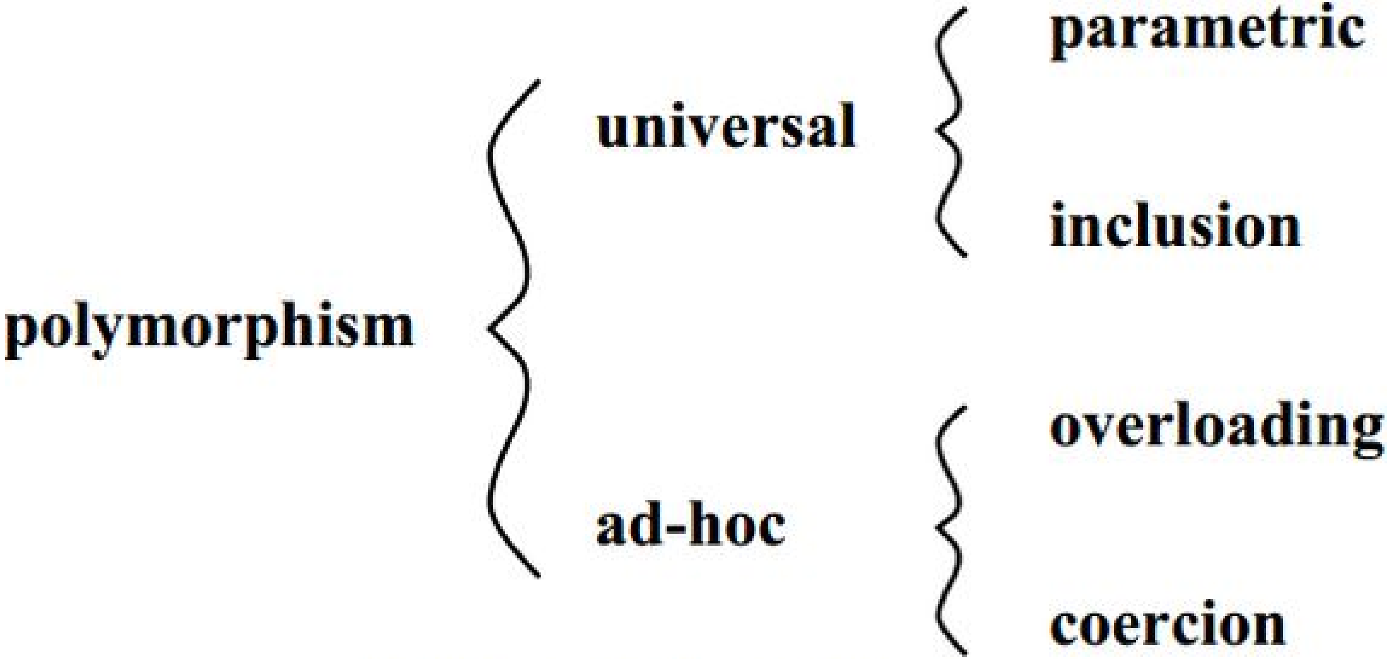
What are polymorphic types?

A polymorphic programing language means that you can write a function which can operate on many types.

There are several different types of polymorphism that a language can use.

Example: A map function that can take a list of any type and a function that operates on the type of the contents of the list.

Four types of polymorphism



polymorphism

universal

parametric

inclusion

ad-hoc

overloading

coercion

Varieties of polymorphism

Parametric

Allows a function or a data type to be written generically. *Eg: Java Generics, C++ Templates*

I can write the following code:

```
class Node<T> {
      T elem;
      Node<T> next;
}
```

I can use the Node class to make a new Node<String> or new Node<Integer> and so on…

Inclusion Polymorphism or Subtyping

Suppose S is subtype of T, if a function expects something of type T, S can be substituted. This is used in many programing languages, like Python, Java etc.Example:

```
function calculateArea(<Shape> shape){

        ...
}
```

Can be used with anything which is a subtype of shape:

```
calculateArea(circle)
calculateArea(square)
```

Ad-Hoc: Overloading

Overloading permits the use of the same operator or method name to denote multiple, distinct program meanings.

+ Operator can both add integer or concatenate strings.

Thus the operator is called *overloaded* because it rely on the compiler to select the appropriate functionality based on program context.

Ad-Hoc: Coercion

Coercion represents implicit parameter type conversion to the type expected by a method or an operator, thereby avoiding type errors.

Ex:

2.0 + 2.0 → 4.0 because double + double

2.0 + 2→ double + int, implicitly convert int to double

Ad-Hoc Differences

Overloading provides syntactic sugar, allowing a developer to use the same name for distinct methods.

Coercion, on the other hand, obviates cumbersome explicit type casts or unnecessary compiler type errors.

How our type system works

Our type system checks types before evaluating and is an explicit parametric polymorphism.

It stores type information in a symtable and typetable. When an ECall is made types are checked for both generic and non-generic functions.

Code Time

Questions

Citation

http://lucacardelli.name/Papers/BasicTypechecking.pdf

https://www.quora.com/When-is-Polymorphism-useful

http://www.javaworld.com/article/2075223/core-java/reveal-the-magic-behind-subtype-polymorphism.html

http://wiki.c2.com/?TypeInference