Polymorphism

Based on the slides of Maurizio Gabbrielli

Polymorphism

which has many forms

- The same function identifier can operate on operands of a different type.
  - sum: 3 + 5 or 3.2 + 4.4
  - sorting: Both C array of characters and N array of integers: sort (C) or sort (N)
- We distinguish three forms of polymorphism (Strachey)
  - Ad hoc **polymorphism, or overloading**
  - Universal polymorphism
    - **Parametric polymorphism** (explicit and implicit)
    - **Subtype polymorphism** (we'll treat this in the oo context)

Ad hoc polymorphism: overloading

- The same symbol denotes different meanings:
  - `3 + 5`
  - `4.5 + 5.3`
- The compiler translates + in different ways
- Always resolved at compile time
  - after type inference
- Do not confuse with compatibility (automatic conversion):
  - `3 + 4.5` if it is correct, it has an automatic coercion (from `int` to `float`) and `+` overloaded resolved as `float * float -> float`
  - ML has overloading but has no automatic coercions

Parametric polymorphism

- A value has Parametric Universal Polymorphism when it has an infinity of different types, they are obtained by instantiation from a single general-type scheme.
- One polymorphic function consists of a single code that applies uniformly to all instances of its general type

```
Identity(x) = x;
sort(v) = ….;
Identity has type <T> → <T>
sort has type <T>[ ] →  void
```

T is a variable of type

Explicit parametric polymorphism (generics)

- In C++: function template
  - A swap function that exchanges two integers

```
void swap (int& x, int& y){
int tmp = x; x=y; y=tmp;}
```

  - A swap template that exchanges two data

```
template <typename T>  //T is like a parameter
void swap (T& x, T& y){
T tmp = x; x=y; y=tmp;}
```

  - Automatic instantiation

```
int i,j;   swap(i,j); //T becomes int at link-time
float r,s; swap(r,s); //T becomes float at link time
String v,w;swap(v,w); //T becomes String at link time
```

Implicit Parametric polymorphism (e.g., ML)

- The swap function in ML:

```
swap(x,y)  = let val tmp = !x in
        x = !y; y = tmp end;
val swap = fn : ‘a ref * ‘a ref -> unit
```

- Instantiation at compile time

```
swap(x,y); //x e y are int var (int ref)
val it = (): unit
swap(v,w); //v e w are string var (string ref)
val it = (): unit
```

Subtype polymorphism

- Similar to the explicit one, but not all types can be used to instantiate the general type
- Kind of universal polymorphism but with restrictions
- Suppose given a subtyping relation that T < S means

  T is a subtype of S
- Def. A value exhibits subtype (or bounded) polymorphism when there is an infinity of different types which can be obtained by instantiating a general type scheme, substituting for a parameter the subtypes of an assigned type.
- A polymorphic function consists of a single code that applies uniformly to all "legal" instances of its general type

Polymorphism and Overloading: Summary

- Parametric Polymorphism
  - A single algorithm can have many types
  - Variables of type replaced by any possible type
  - f : ‘t → ’t   => f : int→int,   f : bool→bool, ...
- Overloading (Ad hoc polymorphism)
  - A single symbol can refer to more than one algorithm
  - Each algorithm can have arbitrarily different types
  - Choosing the algorithm is dictated by the context
  - + has type int * int → int, real * real → real

    Not all possible substitutions

A first conclusion: type-safe languages?

- Remember: Language type-safe:

  No unreported errors that result from a type error are occurring

- Non-Safe: Descendants of BCPL, including C and C++
  - Cast, pointer arithmetic
- Almost safe: Descendants of Algol, Pascal, Ada.
  - Union types (Pascal)
  - Dangling pointers
    - Languages with explicit deallocation cannot be fully type-safe
- Safe: Lisp, ML, Smalltalk, Haskell, (and Java)
  - Lisp, Smalltalk: Dynamic control
  - ML, Haskell, Java: Static control

Programming languages and type-safety

- Assembly

  No control

- C

  Static control
  Non-Safe (union, interoperability of pointers and arrays)

- Pascal

  Static control
  Almost Safe (variant records)

- C++

  Both static and dynamic control (virtual methods)
  Not safe (as C)

- Java

  Static and dynamic control (downcasting)
  Type Safe

- Lisp

  Dynamic control
  Safe

- ML

  Static control
  Safe

Homework

- Chapter 8, exercises 1,3-6