Polymorphism in languages and its implications
(Smalltalk, Modula-3, Java, and C++)

Amer Diwan

Outline

• Not much to say about ad-hoc polymorphism in languages
• Case studies from inclusion polymorphism
  – “Smalltalk”
  – Modula-3
  – Java
  – C++
• Case study from parametric polymorphism
  – SML
Smalltalk

• Example:
  myThings ← Bag new
  myThings add (Point new)
  myThings add (Rectangle new)

  Point

  Rectangle

• Point and Rectangle may be unrelated classes (i.e., common supertype is Object)

Example (cont.)

• myThings do: [:oneThing | oneThing print]
  Above code works on any collection of any classes

• “Implicitly” the items in the collection must have a “print” method
Discussion

- Does Smalltalk’s polymorphism fits in with Cardelli and Wegner’s classification?

Run-time representation
(from Mitchell’s book)
How to implement method dispatch?

- Steps:
  - Get class object
  - Get method dictionary
  - Search method dictionary
    - If found, invoke code
    - If not found, continue search in superclass
    - If reached “Object” then “method not understood”

Finishing up the example
How to improve the performance of method dispatch?

• Search is very expensive. What to do?

Modula-3

• Polymorphism is a direct consequence of subtyping
• Example:

  TYPE printableBag =
   OBJECT
   values: ARRAY OF printableThing;
   METHODS add(v: printableThing) …
  VAR myThings: printableBag;
  myThings.add(new Point); myThings.add(new Rect)
Example (cont.)

- FOR i = 0 TO LAST(myThings.values) DO
  myThing.values[i].print()
- How is this different from Smalltalk?
- What kind of polymorphism is this?

How to implement method dispatch?

- Can do it like Smalltalk but can we exploit
  - Static typing?
    - Possible types of object is constrained statically
  - Single inheritance?
    - Exactly one immediate supertype
V-Tables

- Idea:
  - Append the methods of a supertype to a subtype
  - A method $T::m$ appears in the same position in all $T$’s subclasses

V-tables (cont.)

Construct a v-table for each class

v-tables are typically part of the type descriptor
V-tables (cont.)

\[ t \rightarrow g() \text{ becomes} \]
\[ \text{vp} = t \rightarrow \text{vtable\_ptr} \]
\[ \text{gaddr} = * (\text{vp} + \text{g\_s offset}) \]
\[ (* \text{gaddr})() \]

How to improve the performance of method dispatch?
Java

• Similar to Modula-3 except for `invokeinterface`

• Example:
  ```java
  interface hasAPrint { void print(); }
  class text implements hasAPrint {
      void set_text(char *s) {...}
      void print() {...} ;
  }
  class list implements listInterface, hasAPrint {
      void cons(...) {...}
      void count(...) {...}
      void print() {...};
  }
  hasAPrint anobj; anobj->print();
  What's the problem here?
  ```

Implementing `invokeinterface`

• **Problem**: `print` method may have different offsets in each class implementing `hasAPrint`

• **Solution**:
  – Can implement `invokeinterface` using Smalltalk-like run-time method search
How to improve the performance of invokeinterface

- Have v-tables for interfaces that are hashed using (class, interface name) pair
- Other optimizations?

C++

- Similar to Modula-3 except for
  - virtual/non-virtual distinction
- Multiple inheritance
Challenges with multiple inheritance

• **Conflicts:**
  – e.g., what if two inherited methods have the same name?

• **Implementation**
  – class C : A, B {…}
    C *pc; B* pb;
    pc = new C;
    pb = (B*)pc;

  ![Diagram](image)

  pc
  A’s part
  B’s part
  C’s part

  pb
  A’s part
  B’s part
  C’s part

Implementation issues

• Need to adjust pointers when casting, invoking methods, comparing, ...

  ![Diagram](image)

  pc
  pb

  A’s part
  B’s part
  C’s part

  pb = (B*)pc;
  pc = (C*)pb;
  if (pc == pb) {…}

  pc
  pb

  A’s part
  B’s part
  C’s part

• It’s a bit more hairy for virtual method invocations: the paper talks about it
How to speed up method dispatch?

Discussion

• Four kinds of (inclusion-style) polymorphism
  – Smalltalk:
  – Modula-3:
  – Java:
  – C++:
• What would you rather use?
Next lecture

- Parametric polymorphism case study: SML
- Reading: SML document, Bob Harper (link from class web page)