Parametric polymorphism case study: SML

Amer Diwan

SML: Standard ML

- Functional
  - Assignments exist but strongly discouraged
- Statically typed (but types usually inferred)
- Provides parametric polymorphism
Example

- fun map f nil = nil
  | map f (hd::tl) = f(hd) :: map f tl
  fn : ('a -> 'b) -> ('a list) -> ('b list)
- fun find (nil, _) = false
  | find (hd::tl, tofind) = tofind = hd orelse find(tl, tofind)
  fn : ('a list * 'a) -> bool
- Polytypes may or may not allow equality

What kind of polymorphism?

- SML supports parametric polymorphism but not inclusion polymorphism:
- Can’t write:
  - value moveX = fun(p: Point, dx: Int)(returns Point)
    p.x = p.x + dx; p
So what does it support well?

- fun reverse l =
  let fun rev(nil, y) = y
  | rev(hd::tl, y) = rev(tl, hd::y)
  in
  rev(l, nil)
  end
- type: ‘a list -> ‘a list
- Cannot write this in Java, Modula-3, C++, ...

A graphical view

Cons takes a list of alphas and a value of alpha and returns a list of alphas

Problem: How should the list and alphas be represented?
Would like to implement other generic functions that work on lists of any type

Some bad news

- Cons needs to know the size and representation of its inputs
  - Integers may be 32 bits while reals may be 64 or 128 bits
  - Many polymorphic functions need to know something about the representation of their inputs

If a function works on $t \in T$, then all types in $T$ better look the same to the function
Some ugly pictures

See what boxing can do?

Schemes for “representation analysis”

Boxed representation has significant overhead in space and time
- Could use “natural representation” when possible and insert coercions for polymorphic uses [Shao and Appel]
- Could use “natural representation” but pass “type descriptors” to polymorphic functions [Morrisett et al., Tarditi et al.]
- Could use some combination of the two [Shao]
Using coercions

“Generation 2” for SML/NJ [Shao and Appel]

- Copying is expensive so minimize coercions
- Copying impractical for recursive structures
- Copying problematic for assignable structures

Intensional type analysis

Idea: explicitly pass types around [Morrisett]
How to use the types?

```haskell
fun cons [t] (l: α list, v: α) =
    typecase t of
    int = intcons (l, v);
    | float = floatcons (l, v)
    | else = boxed_cons (l, v)
```

Discussion

- How are the representation issues different or same for SML and M-3 like languages?
Summary: Polymorphism in languages

• Simple single inheritance subclass inclusion polymorphism is simple and fast
  – Java interfaces add complexity but only when used
  – C++ multiple inheritance adds complexity but only when used
• Parametric polymorphism in SML is simple and fast too
  – Everything must be boxed, unless optimized

Next topic

• Interaction of inheritance and encapsulation
• Reading: Snyder, “Inheritance and encapsulation in object-oriented languages”. Available from Pat.