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

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]
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