Wanting Type Variables

Will downcast (potential run-time exception) the arguments to \( f \) and the result of \( \text{find} \)

We are not enforcing list-element type-equality

OOP-style subtyping is no replacement for parametric polymorphism; we can have both:

```plaintext
interface Cmp<'a> { Int f('a,'a); } // Cmp not a type
class SList<'a> { // SList not a type (SList<Int> e.g. is)
    ... some field definitions (can use type 'a) ...
    constructor (Cmp<'a> x) {...}
    SList cons('a x) {...}
    'a find('a x) {...}
}
```

No more downcasts; the best of both worlds

Wanting bounds

There are compelling reasons to bound the instantiation of type variables

Simple example: Use at supertype without losing that it’s a subtype

```plaintext
interface I { unit print(); }
class Logger< 'a <: I > { // must apply to subtype of I 'a item;
    'a get_it() { syslog(item.print()); item }
}
```

Without polymorphism or downcasting, client could only use \( \text{get_it} \) result for printing

Without bound or downcasting, Logger could not print

Issue isn’t special to OOP
Fancy Example from “A Theory of Objects” Abadi/Cardelli

With forethought, bounds can avoid some subtyping limitations

interface Omnivore { unit eat(Food); }
interface Herbivore { unit eat(Veg); } // Veg <= Food

Allowing Herbivore ≤ Omnivore could make a vegetarian eat meat (unsound)! But this works:

interface Omnivore< 'a <: Food > { unit eat('a); }
interface Herbivore< 'a <: Veg > { unit eat('a); }

If Herbivore<T> is legal, then Omnivore<T> is legal and
Herbivore<T> ≤ Omnivore<T>!

Useful for unit feed('a food, Omnivore< 'a > animal) {...}

Bounded Polymorphism

This “bounded polymorphism” is useful in any language with universal types and subtyping. Instead of ∀α.τ and Λα.e, we have ∀α < τ′.τ and Λα < τ′.e:

▶ Change Δ to be a list of bounds (α < τ) instead of a set of type variables
▶ In e you can subsume from α to τ'
▶ e₁[τ₁] typechecks when τ₁ "satisfies the bound" in type of e₁

One limitation: When is (∀α₁<τ₁.τ₂) ≤ (∀α₂<τ₃.τ₄)?

▶ Contravariant bounds and covariant bodies assuming bound are sound, but makes subtyping undecidable
▶ Requiring invariant bounds and covariant bodies regains decidability, but obviously allows less subtyping