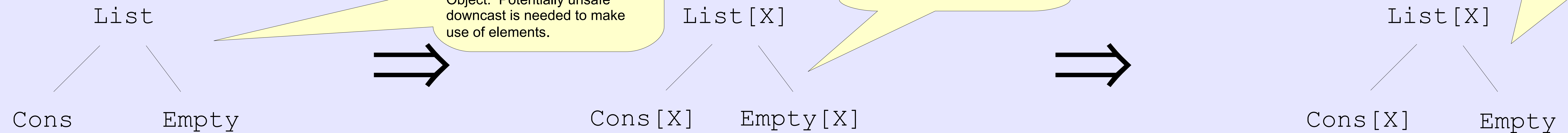


Hidden Type Variables for More Expressive Generic Programs

Joe Hallett, Assaf Kfoury joint with Eric Allen, Victor Luchangco, Sukyoung Ryu
Boston University and Sun Microsystems

What are hidden type variables?



Consider a non-generic list class hierarchy in a conventional object-oriented language. Such lists lack precision. We only know elements are subtypes of Object. Potentially unsafe downcast is needed to make use of elements.

Generics allow for greater precision and static checking.

By separating the ability to introduce universally quantified type variables from parameterization we allow a non-parametric empty list that extends all lists.

What do hidden type variables provide? – Variance Types:

```

trait List [ X extends Y ] extends { List[Y] } where { Y extends Object }
  cons(x : Y) : List[Y] = Cons[Y] (x, self)
end
  
```

List[Z] is a subtype of List[Number] because Z is a subtype of Number.

Notice that Empty is parameterized by a type. Each instantiation of Empty corresponds to a distinct value. This parameterization is misleading, wasteful of memory, and inhibits polymorphism.

We allow such relationships by generalizing conventional nominal subtyping to allow the declaration of type variables in a *where clause* that is separate from ordinary type variables. The declaration of the empty list looks like:

```

trait Empty extends { List[X] }
  where { X extends Object }
end
  
```

```

Cons[Z] (3, Empty).cons (5.7)
  
```

5.7 has type IR.

This call is well typed! It results in a list of type List[Number].

What else do hidden type variables provide? – Infinitely Broad, Selective, and Open Extensions:

Cow extends some, but not all Herbivore's. In particular, only Herbivore's that eat Grass are extended.

```

trait Eater [ X extends Food ]
  eat(x : X) : ()
end

trait Herbivore [ Y extends Plant ] extends { Eater[Y] } end

trait Cow extends { Herbivore[Z] } where { Z extends Grass } end
  
```

Cow extends infinite traits, i.e., all Herbivore's that eat Grass.

We can augment the set of superclasses after a class is defined. For example, if we later add:

```

trait BlueGrass extends Grass end

then Cow extends Herbivore[BlueGrass].
  
```

No Free Lunch:

```

trait Collection extends { Object }
  clear() : Collection
end

trait List [ X extends Object ] extends { Collection }
  clear() : List[X] = Empty
end

object Empty extends { List[Y] } where { Y extends Object }
end
  
```

When checking that Empty is a subtype of Collection we must find a *witness* for Y.

The method call Empty.clear() requires determining which instance of the method clear is invoked.

Even more troubling, if v has static type Collection, but dynamic type Empty then the call v.clear() requires determining which instance of clear is invoked at run time.