

```
fun append (xs,ys) =  
  if xs=[]  
  then ys  
  else (hd xs)::append(tl xs,ys)  
  
fun map (f,xs) =  
  case xs of  
    [] => []  
  | x::xs' => (f x)::(map(f,xs'))  
  
val a = map (increment, [4,8,12,16])  
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman

Another Equivalent Structure

More interesting example

Given a signature with an abstract type, different structures can:

- Have that signature
- But implement the abstract type differently

Such structures might or might not be equivalent

Example (see code):

- **type rational = int * int**
- Does *not* have signature **RATIONAL_A**
- *Equivalent* to both previous examples under **RATIONAL_B** or **RATIONAL_C**

More interesting example

```
structure Rational3 =  
struct  
  type rational = int * int  
  exception BadFrac  
  
  fun make_frac (x,y) = ...  
  fun Whole i = (i,1) (* needed for RATIONAL_C *)  
  fun add ((a,b) (c,d)) = (a*d+b*c,b*d)  
  fun toString r = ... (* reduce at last minute *)  
end
```

Some interesting details

- Internally `make_frac` has type `int * int -> int * int`, but externally `int * int -> rational`
 - Client cannot tell if we return argument unchanged
 - Could give type `rational -> rational` in signature, but this is awful: makes entire module unusable – why?
- Internally `Whole` has type `'a -> 'a * int` but externally `int -> rational`
 - This matches because we can specialize `'a` to `int` and then abstract `int * int` to `rational`
 - `Whole` cannot have types `'a -> int * int` or `'a -> rational` (must specialize all `'a` uses)
 - Type-checker figures all this out for us