```
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

Depth Subtyping

More record subtyping?

[Warning: I am misleading you ☺]

Subtyping rules so far let us drop fields but not change their types

Example: A circle has a center field holding another record

```
fun circleY (c:{center:{x:real,y:real}, r:real}) =
    c.center.y

val sphere: {center:{x:real,y:real,z:real}, r:real} =
    {center={x=3.0,y=4.0,z=0.0}, r=1.0}

val _ = circleY(sphere)
```

For this to type-check, we need:

```
        {center:{x:real,y:real,z:real}, r:real}
                        <:
        {center:{x:real,y:real}, r:real}
```

Do not have this subtyping – could we?

```
{center:{x:real,y:real,z:real}, r:real}
                    <:
     {center:{x:real,y:real}, r:real}
```

- No way to get this yet: we can drop `center`, drop `r`, or permute order, but cannot "reach into a field type" to do subtyping

- So why not add another subtyping rule… "Depth" subtyping:
  If `ta <: tb`, then `{f1:t1, …, f:ta, …, fn:tn} <: {f1:t1, …, f:tb, …, fn:tn}`

- Depth subtyping (along with width on the field's type) lets our example type-check

Stop!

- It is nice and all that our new subtyping rule lets our example type-check

- But it is not worth it if it breaks soundness
  - Also allows programs that can access missing record fields

- Unfortunately, it breaks soundness ☹

Mutation strikes again

If `ta <: tb`,
then `{f1:t1, …, f:ta, …, fn:tn} <:`
   `{f1:t1, …, f:tb, …, fn:tn}`

```
fun setToOrigin (c:{center:{x:real,y:real}, r:real})=
    c.center = {x=0.0, y=0.0}

val sphere: {center:{x:real,y:real,z:real}, r:real} =
    {center={x=3.0, y=4.0, z=0.0}, r=1.0}

val _ = setToOrigin(sphere)
val _ = sphere.center.z (* kaboom! (no z field) *)
```

Moral of the story

- In a language with records/objects with getters and setters, depth subtyping is unsound
  - Subtyping cannot change the type of fields

- If fields are immutable, then depth subtyping is sound!
  - Yet another benefit of outlawing mutation!
  - Choose two of three: setters, depth subtyping, soundness

- Remember: subtyping is not a matter of opinion