Types for objects

$\tau_1 <: \tau_2$

$\{f_1 :: \tau_1, \ldots, f_n :: \tau_n\} <: \{f_1 :: \tau_1, \ldots, f_{n-1} :: \tau_{n-1}\}$

```
oxy = { xs : Link(Link(...)), ys : Empty }
```

```
EMPTY = { is-member:
LIST      (Num -> Bool)}
```

```
LINK = {is-member ::
         (Num -> Bool),
first :: Num,
rest :: LIST }
```

{ xs :: LINK , ys :: EMPTY }

```
fun memberxy (o :: { xs :: List, ys :: List}
                elt :: Num) -> Bool
  o.xs.is-member(elt) or o.ys.is-member(elt)
end
memberxy(oxy, 5)
```

{ xs :: LINK , ys :: EMPTY } $\overset{?}{<:}$ { xs :: List, ys :: List }

$\{f_1 :: \tau_1, \ldots, f_n :: \tau_n\} <: \{f_1 :: \tau_1', \ldots, f_n :: \tau_n\}$

DEPTH SUBTYPING

$\tau_1 <: \tau_1'$ ← CO-VARIANT

```
class LinkWFold extends Link
    fold: lam(f :: (Num, Num -> Num),
              base :: Num):
        ...
    end
end
```

```
merge (Link(first, rest),
       [dict: "fold", lam(f, base): ...end])
```

LINKWFOLD <: LINK

```
LINK = {is-member::
          (Num->Bool),
        first:: Num,
        rest:: LIST }
```

```
LINKWFOLD =
    {is-member::
       (Num->Bool),
     first:: Num,
     rest:: LIST,
     fold::((Num,Num->Num),
            Num
            -> Num) }
```

```
class LinkWJoeFold extends LinkWFold
    fold: lam(base :: Num,
              f :: (Num, Num -> Num)) -> Num
      super(self).fold(f, base)
    end
end
```

```
merge([LinkWFold(f,i),
[dict:
  "fold", lam(base,f)
    ... end])
```

LINKWJOEFOLD =

```
{is-member::
   (Num->Bool),
 first:: Num,
 rest:: JOEFOLDLIST,
 fold::(Num,
        (Num,Num -> Num)
        -> Num) }
```

X

<:

LINKWFOLD =

```
{is-member::
   (Num->Bool),
 first:: Num,
 rest:: FOLDLIST,
 fold::((Num,Num -> Num),
        Num
        -> Num) }
```

(Num, (Num, Num -> Num) -> Num)

X

<: ((Num,Num -> Num), Num -> Num)

Class1 extends Class2

BUT

X

Class1 <: Class2

INHERITANCE IS NOT SUBTYPING

WILLIAM COOK

```
class FuzzyMemberLink extends Link
    is-member : lam(elt :: Inexact) -> Bool
        within(elt, self.first, 0.01) or
            self.rest.is-member(elt)
    end
end
```

Inexact <: Num

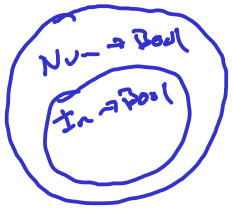
```
FML = {
    is-member :: (Inexact -> Bool)
    first :: Num,
    rest :: FMList
}
```

<:

```
LINK = {
    is-member :: (Num -> Bool)
    fst :: Num,
    rest :: LIST
}
```

FMList <: List ← [what we are trying to show]

(Inexact → Bool) <: (Num → Bool)



?

Inexact <: Num

CONTRA-VARIANCE

$(T_1 \to T_2) <: (T_3 \to T_4)$

$T_1 <: T_3$

$T_2 <: T_4$

S <: U

{x: Num} <: {}

fun f(o: {x: Num})

...

end

fun f(o :: {}):

...

end

more possible operations (example below)

```
fun f1(o :: {x : Num}) -> Num :
  o.x
end

f1 :: ({x : Num} -> Num)
```

$$(\{x : Num\} \to Num) \overset{?}{<:} (\{\} \to Num)$$

Better not be true, but assume for this example

if this were true, think about this example:

```
fun call-g(g :: ({} -> Num)) -> Num :
  g({y : 10})
end
call-g(f1)
```

↑ $\{y : Num\} <: \{\}$ by width rule, but this call raises field-not-found!

use red box to make this app ok