Kinds review

Consider

type nat = int list
type 'a env = 'a -> string

What’s good? How do you know?

val addNat : nat * nat -> nat
val isBound : name * env -> bool

Δ has these bindings:

nat :: *

env :: * ⇒ *
New topic: Type inference
What type inference accomplishes

\[
\rightarrow (\text{define } \quad \text{double } (x) \quad (+ \ x \ x))
\]

double

\[
\rightarrow (\text{define } \text{int } \text{double } ([x : \text{int}]) \ ( + \ x \ x))
\]
double : (int -> int)

\[
\rightarrow (\text{define } \quad \text{double } (x) \quad (+ \ x \ x))
\]
double : (int -> int)
What else type inference accomplishes

-> (@ cons bool) #t ((@ cons bool) #f (@ '() bool))
(#t #f) : (list bool) ;; typed uScheme

-> ( cons #t ( cons #f '() )
(#t #f) : (list bool) ;; nML
How it works

1. For each unknown type, a fresh type variable
2. Every typing rule adds equality constraints
3. Instantiate every variable automatically
4. Introduce polymorphism at let/val bindings
Examples

At the board…