Lesson 11
Universal Types

2/28

Chapter 23

Universal Types and System F

- Varieties of polymorphism
- System F
- Examples
- Basic properties
- Erasure
- Evaluation issues
- Parametricity
- Impredicativity

Varieties of polymorphism

- parametric polymorphism
- *ad hoc* polymorphism (overloading)
  - conventional
  - multimethods
  - Haskell type classes
- intensional polymorphism
  - analyzing and dispatching off of type structure
- subtype polymorphism (subsumption)
- OO "polymorphism" ("dynamic binding")
- row polymorphism (open, extensible record types)

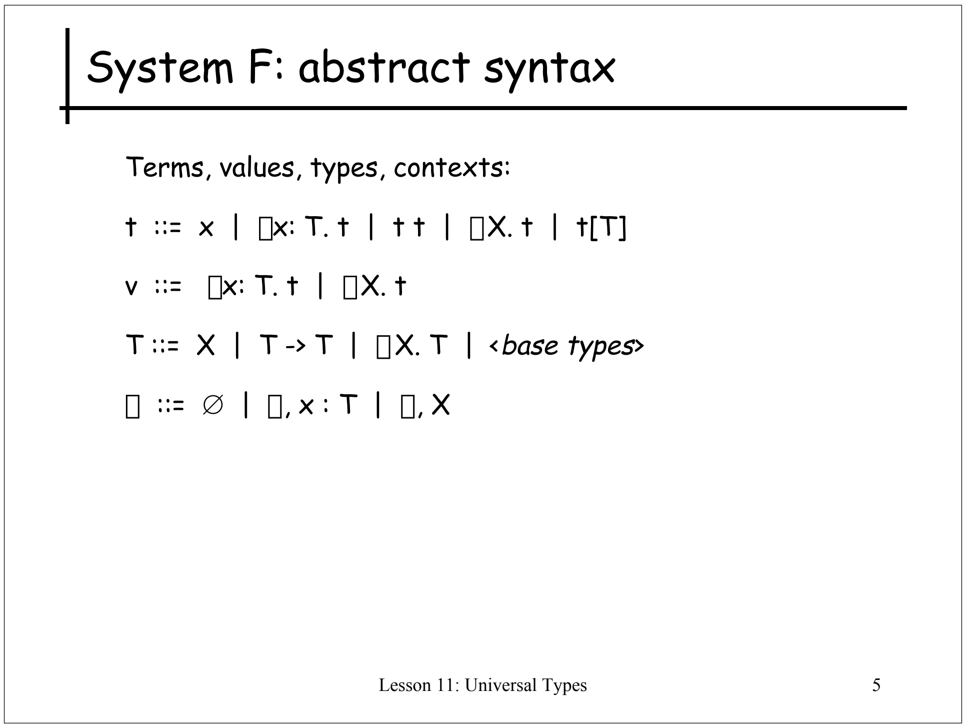
System F

History: Girard 1972; Reynolds 1974

Idea: lambda abstraction over type variables, defining functions over types.

```
id  =  □X. □x: X. x
id  :  □X. X -> X

id [Nat]  □   [X => Nat] □x: X. x  =  □x: Nat. x
id [Nat] : [X => Nat](X -> X)  =  Nat -> Nat
```



System F: abstract syntax

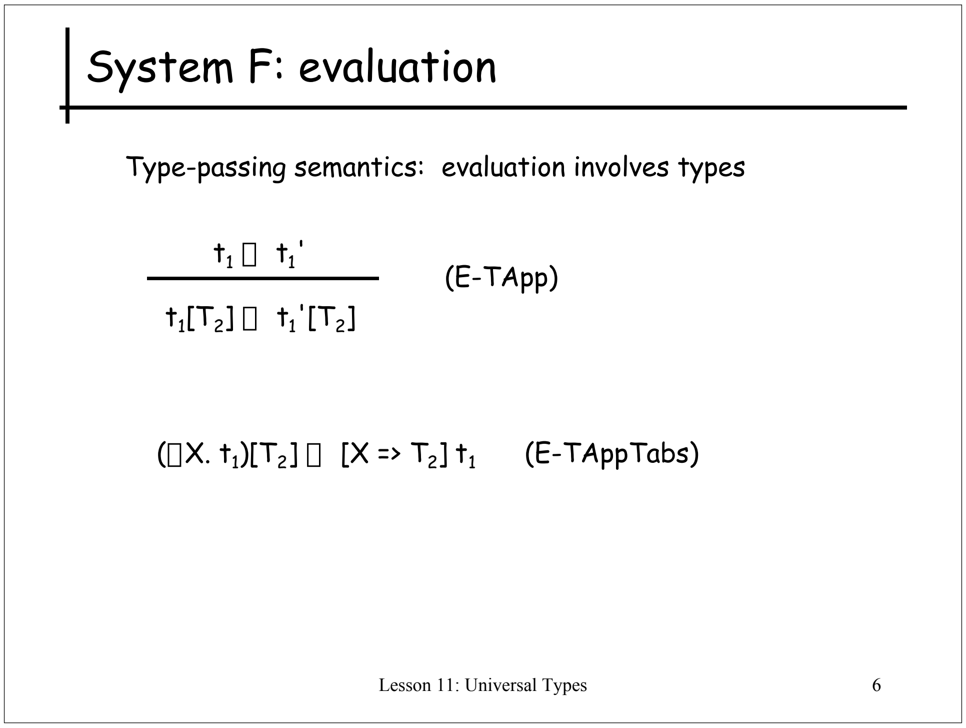
Terms, values, types, contexts:

$t ::= x \mid \lambda x{:}\, T.\, t \mid t\ t \mid \Lambda X.\, t \mid t[T]$

$v ::= \lambda x{:}\, T.\, t \mid \Lambda X.\, t$

$T ::= X \mid T \text{->} T \mid \forall X.\, T \mid$ <*base types*>

$\Gamma ::= \varnothing \mid \Gamma, x : T \mid \Gamma, X$

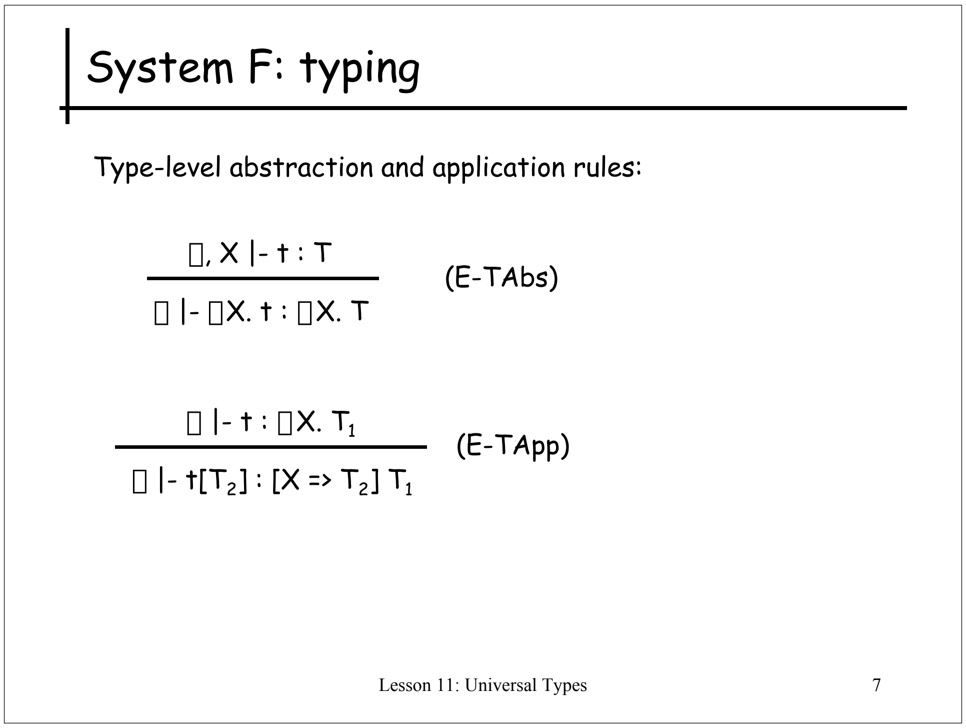


System F: evaluation

Type-passing semantics: evaluation involves types

$$\frac{t_1 \to t_1'}{t_1[T_2] \to t_1'[T_2]} \qquad \text{(E-TApp)}$$

$$(\Lambda X.\, t_1)[T_2] \to [X \Rightarrow T_2]\, t_1 \qquad \text{(E-TAppTabs)}$$



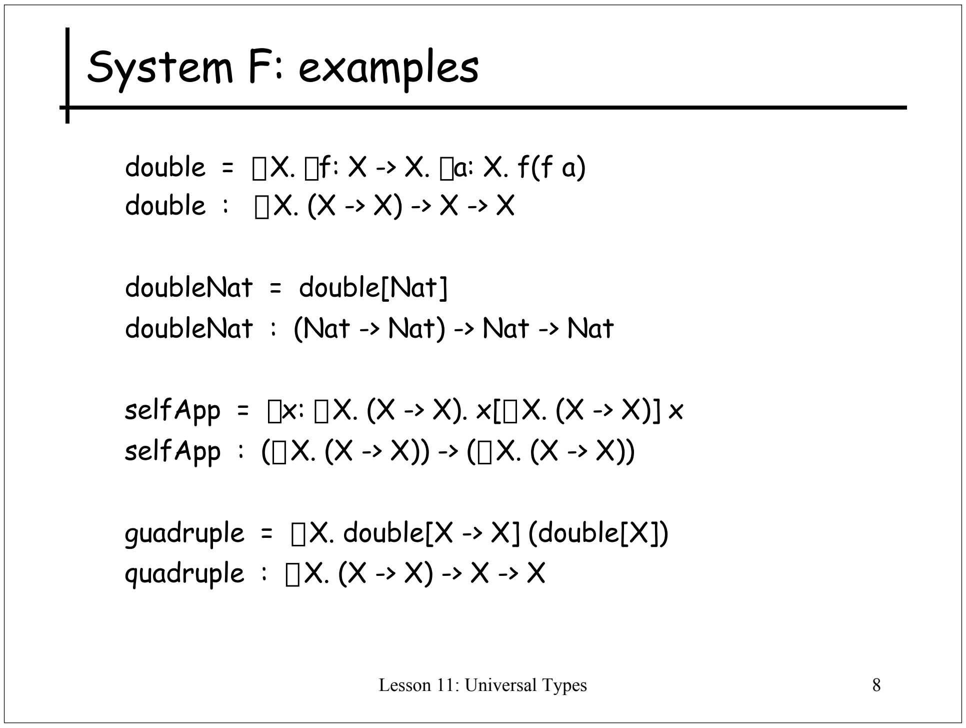
System F: typing

Type-level abstraction and application rules:

$$\frac{\square,\ X\ |\text{-}\ t : T}{\square\ |\text{-}\ \square X.\ t : \square X.\ T} \qquad \text{(E-TAbs)}$$

$$\frac{\square\ |\text{-}\ t : \square X.\ T_1}{\square\ |\text{-}\ t[T_2] : [X \Rightarrow T_2]\ T_1} \qquad \text{(E-TApp)}$$

Lesson 11: Universal Types 7



System F: examples

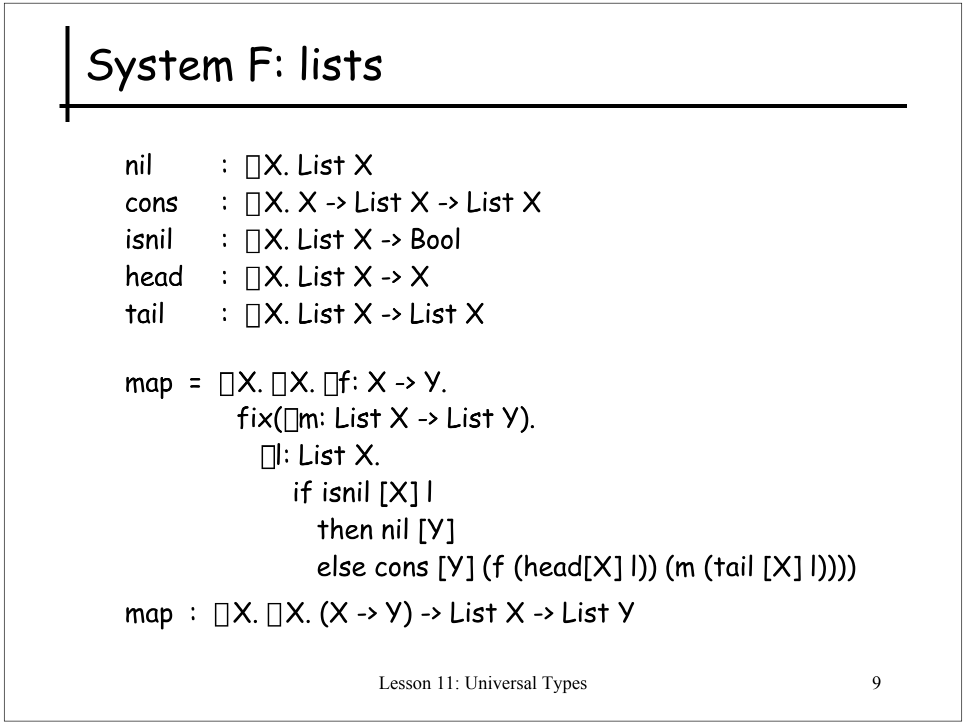
double = □X. □f: X -> X. □a: X. f(f a)
double : □X. (X -> X) -> X -> X

doubleNat = double[Nat]
doubleNat : (Nat -> Nat) -> Nat -> Nat

selfApp = □x: □X. (X -> X). x[□X. (X -> X)] x
selfApp : (□X. (X -> X)) -> (□X. (X -> X))

guadruple = □X. double[X -> X] (double[X])
quadruple : □X. (X -> X) -> X -> X

Lesson 11: Universal Types 8

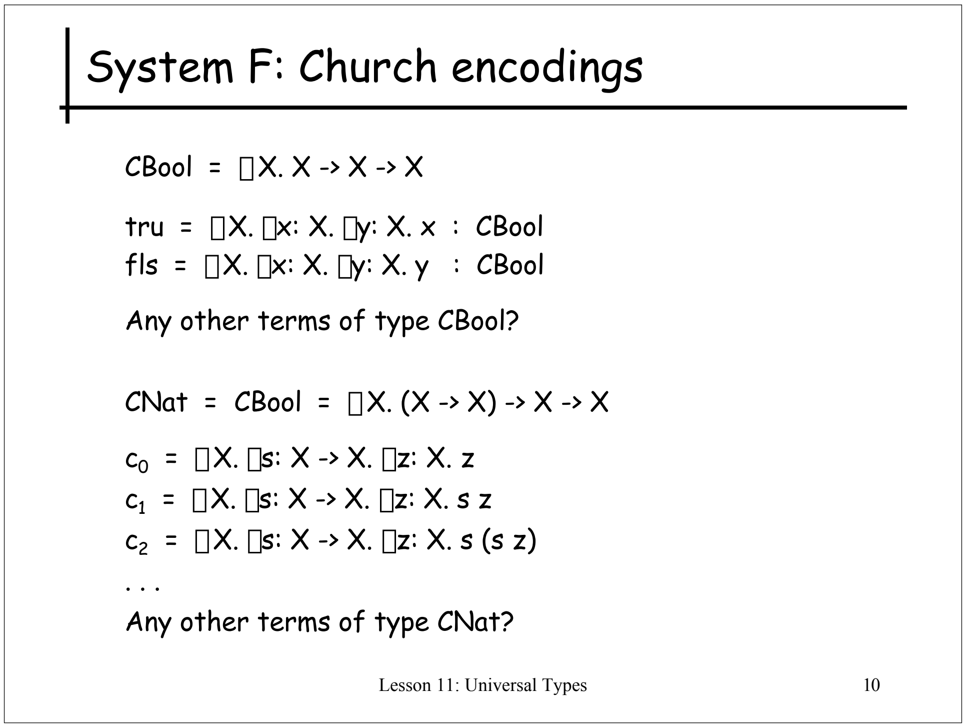


System F: lists

```
nil     :  □X. List X
cons    :  □X. X -> List X -> List X
isnil   :  □X. List X -> Bool
head    :  □X. List X -> X
tail    :  □X. List X -> List X

map  =  □X. □X. □f: X -> Y.
           fix(□m: List X -> List Y).
             □l: List X.
                if isnil [X] l
                  then nil [Y]
                  else cons [Y] (f (head[X] l)) (m (tail [X] l))))
map  :  □X. □X. (X -> Y) -> List X -> List Y
```

Lesson 11: Universal Types 9



System F: Church encodings

CBool = □X. X -> X -> X

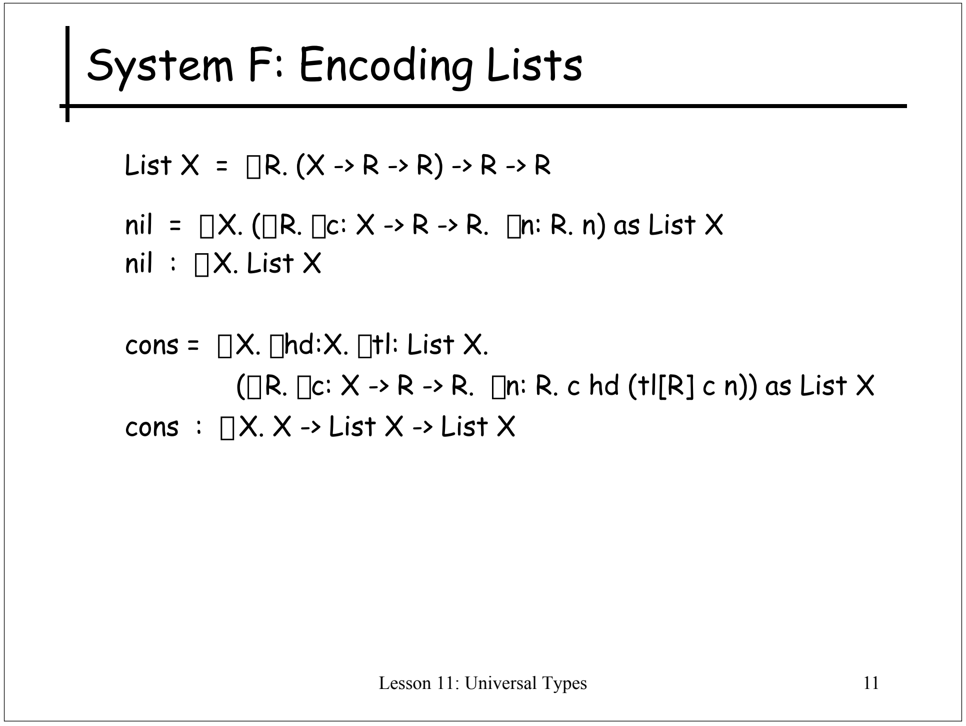
tru = □X. □x: X. □y: X. x : CBool
fls = □X. □x: X. □y: X. y : CBool

Any other terms of type CBool?

CNat = CBool = □X. (X -> X) -> X -> X

c_0 = □X. □s: X -> X. □z: X. z
c_1 = □X. □s: X -> X. □z: X. s z
c_2 = □X. □s: X -> X. □z: X. s (s z)
. . .
Any other terms of type CNat?

Lesson 11: Universal Types 10



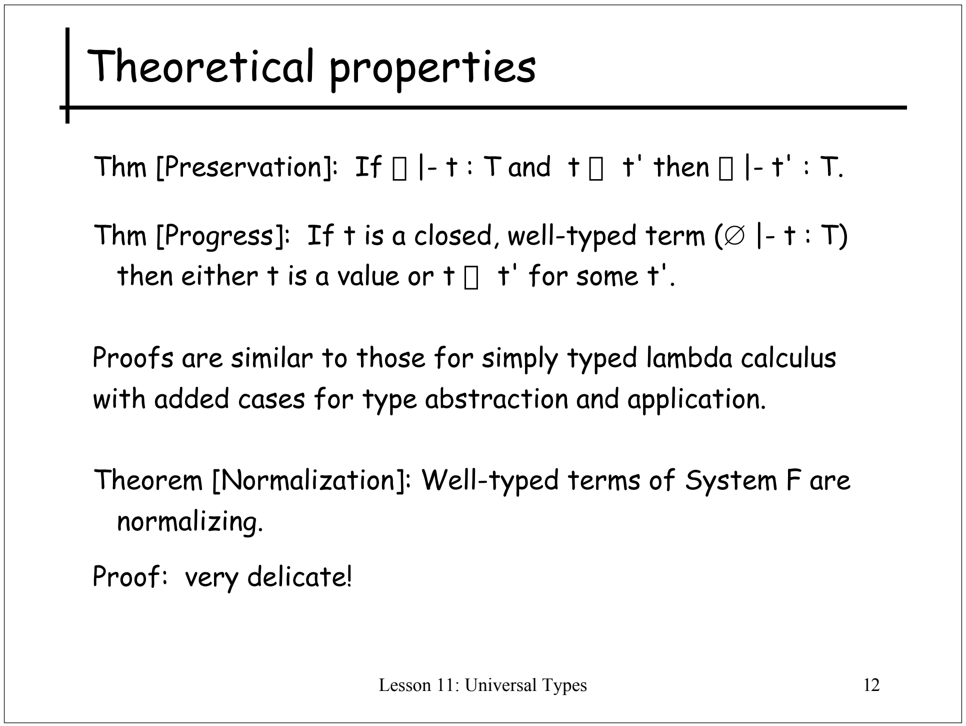
System F: Encoding Lists

List X = ∀R. (X -> R -> R) -> R -> R

nil = ΛX. (ΛR. λc: X -> R -> R. λn: R. n) as List X
nil : ∀X. List X

cons = ΛX. λhd:X. λtl: List X.
(ΛR. λc: X -> R -> R. λn: R. c hd (tl[R] c n)) as List X
cons : ∀X. X -> List X -> List X

Lesson 11: Universal Types 11



Theoretical properties

Thm [Preservation]: If Γ |- t : T and t → t' then Γ |- t' : T.

Thm [Progress]: If t is a closed, well-typed term (∅ |- t : T) then either t is a value or t → t' for some t'.

Proofs are similar to those for simply typed lambda calculus with added cases for type abstraction and application.

Theorem [Normalization]: Well-typed terms of System F are normalizing.

Proof: very delicate!

Lesson 11: Universal Types 12

Erasure and type reconstruction

Easy to map System F to untyped lambda calculus:

erase (x)	=	x
erase (□x: T. t)	=	□x. erase(t)
erase (t_1 t_2)	=	(erase(t_1)) (erase(t_2))
erase (□X. t)	=	erase(t)
erase (t[T])	=	erase(t)

Thm [Wells, 94]: It is undecidable whether, given a closed term m of the untyped lambda calculus, there is a well-typed term t in System F such that m = erase(t).

However, there is lots of work on partial solutions to the type reconstruction problem for System F.

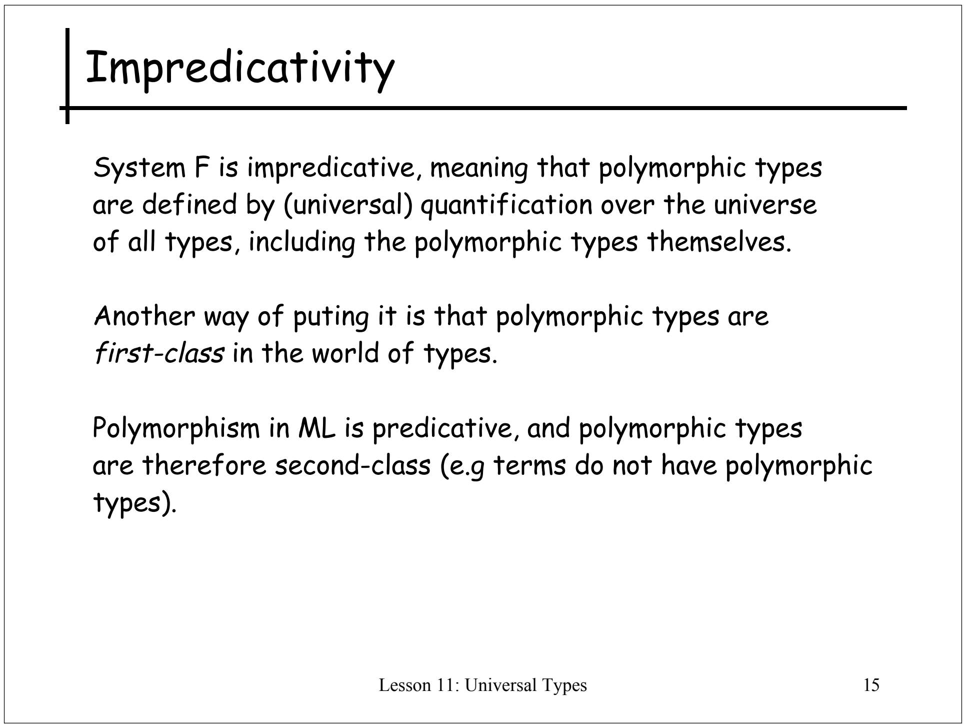
Erasure and evaluation

Erasure operational semantics throws away types before evaluation. But have to preserve *value* nature of □X. t:

let f = □X. error in 0

produces no error because □X is suspending.
So define erasure for evaluation as follows:

erase (x)	=	x
erase (□x: T. t)	=	□x. erase(t)
erase (t_1 t_2)	=	(erase(t_1)) (erase(t_2))
erase (□X. t)	=	□_.erase(t)
erase (t[T])	=	erase(t)()



Impredicativity

System F is impredicative, meaning that polymorphic types
are defined by (universal) quantification over the universe
of all types, including the polymorphic types themselves.

Another way of puting it is that polymorphic types are
first-class in the world of types.

Polymorphism in ML is predicative, and polymorphic types
are therefore second-class (e.g terms do not have polymorphic
types).

Lesson 11: Universal Types 15