

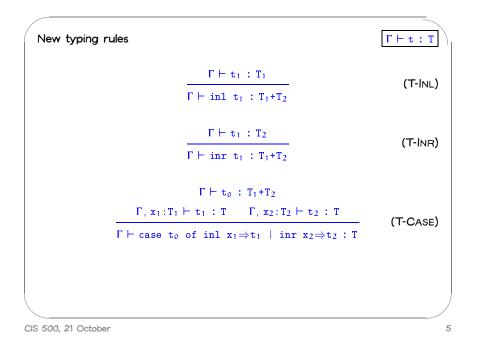
Sums - example
PhysicalAddr = {firstlast:String, addr:String}
VirtualAddr = {name:String, email:String}
Addr = PhysicalAddr + VirtualAddr
inl : "PhysicalAddr → PhysicalAddr+VirtualAddr"
inr : "VirtualAddr → PhysicalAddr+VirtualAddr"
getName = \la:Addr.
case a of
inl x ⇒ x.firstlast
| inr y ⇒ y.name;

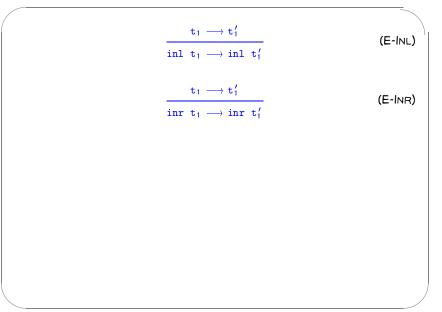
New syntactic forms t ::= ... terms inl t tagging (left) tagging (right) inr t case t of inl $x \Rightarrow t$ | inr $x \Rightarrow t$ case v ::= ... values tagged value (left) inl v inr v tagged value (right) T ::= ... types T+T sum type

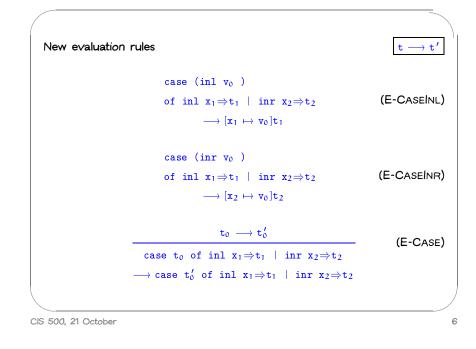
3

1

2

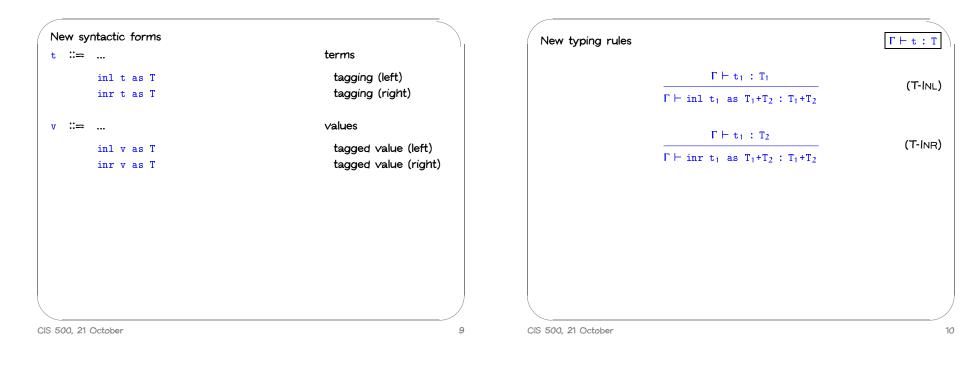


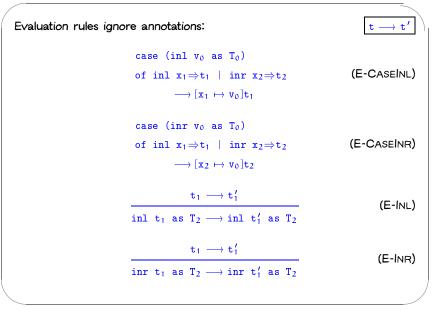


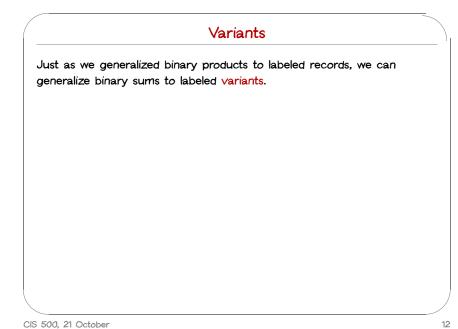


Sums and Uniqueness of Types				
Problen	n:			
lf t	has type T, then inl t has type T+U for every U.			
l.e., we	ve lost uniqueness of types.			
Possible	solutions:			
♦ "Inf	er" ${\tt U}$ as needed during typechecking			
app	e constructors different names and only allow each name to ear in one sum type (requires generalization to "variants," whicl I see next) — OCaml's solution			
Anr	notate each inl and inr with the intended sum type.			
For sim	plicity, let's choose the third.			

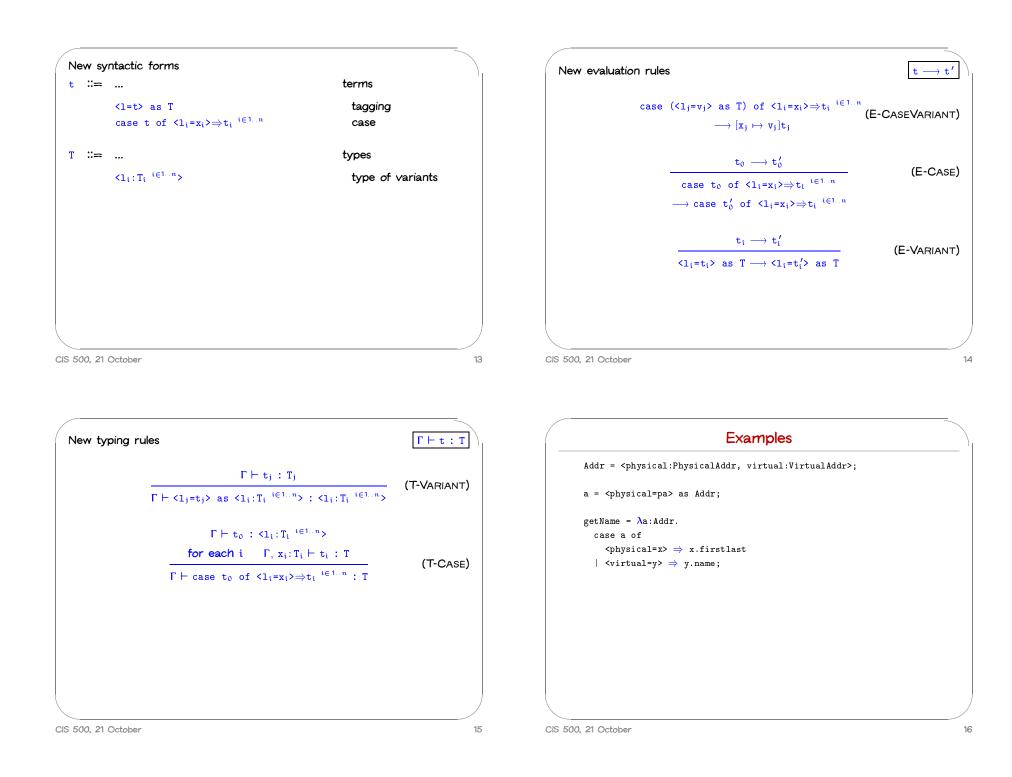
CIS 500, 21 October

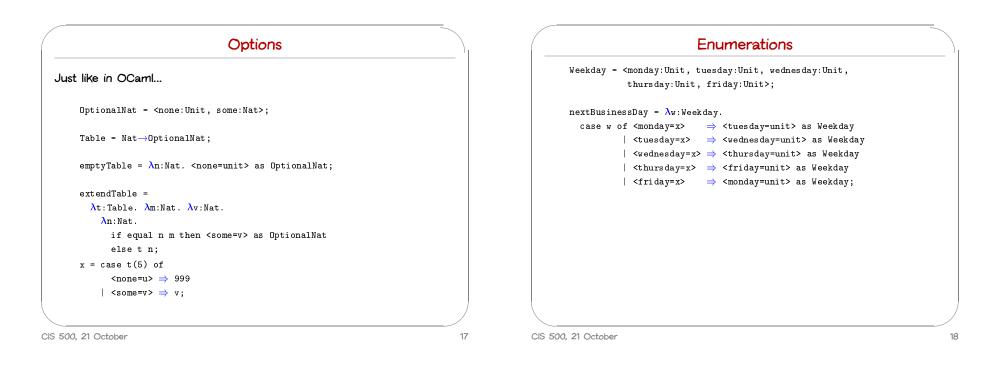






11





Terminology: "Union Types"

 T_1+T_2 is a disjoint union of T_1 and T_2 (the tags inl and inr ensure disjointness)

We could also consider a non-disjoint union $T_1 \vee T_2$, but its properties are more complex because it induces an interesting subtype relation...

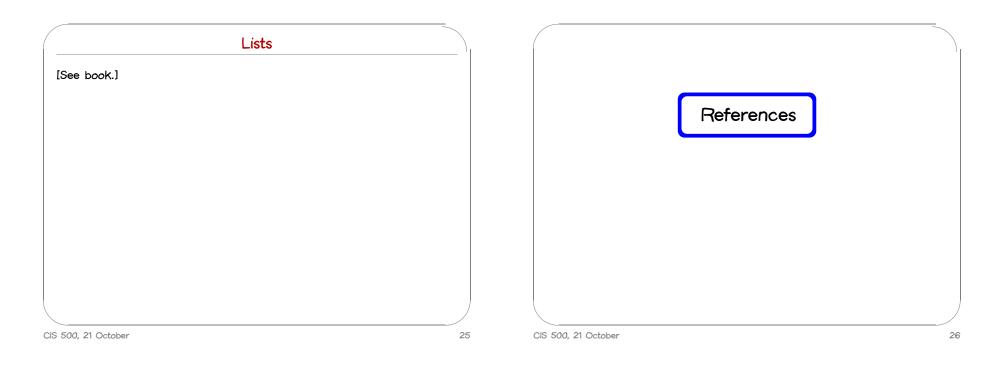
General Recursion

- In λ_{\rightarrow} , all programs terminate. (Cf. Chapter 12.)
- Hence, untyped terms like omega and fix are not typable.
- But we can extend the system with a (typed) fixed-point operator...



19

Example)	New syntactic forms	towns
ff = λ ie:Nat \rightarrow Bool. λ x:Nat. if iszero x then true		t ∷≕ fix t	terms fixed point of t
else if iszero (pred x) then false else ie (pred (pred x));		New evaluation rules	$t \longrightarrow t'$
iseven = fix ff; iseven 7;		$\begin{array}{c} \texttt{fix} \ (\lambda \texttt{x}:\texttt{T}_1,\texttt{t}_2) \\ & \longrightarrow [\texttt{x} \mapsto (\texttt{fix} \ (\lambda \texttt{x}:\texttt{T}_1,\texttt{t}_2)] \end{array}$	(E-FIXBETA)
		$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{fix} \ \mathtt{t}_1 \longrightarrow \mathtt{fix}}$	
 00, 21 October	21	CIS 500, 21 October	2
w typing rules	$\Gamma \vdash t : T$	A more convenier	nt form
ew typing rules $\frac{\Gamma \vdash t_1 \ : \ T_1 \rightarrow T_1}{\Gamma \vdash fix \ t_1 \ : \ T_1}$	[[] [] [] [] [] [] [] [] [] [] [] [] []	letrec $x:T_1=t_1$ in $t_2 = let x =$	
		letrec $x:T_1=t_1$ in $t_2 \stackrel{\text{def}}{=} \text{let } x =$ letrec iseven : Nat \rightarrow Bool = $\lambda x:$ Nat. if iszero x then true else if iszero (pred x) then false	
$\underline{\Gamma \vdash \mathtt{t}_1 \ : \ \mathtt{T}_1 \rightarrow \mathtt{T}_1}$		letrec $x:T_1=t_1$ in $t_2 \stackrel{\text{def}}{=} \text{let } x =$ letrec iseven : Nat \rightarrow Bool = $\lambda_x:Nat.$ if iszero x then true	
$\underline{\Gamma \vdash \mathtt{t}_1 \ : \ \mathtt{T}_1 \rightarrow \mathtt{T}_1}$		<pre>letrec x:T₁=t₁ in t₂ $\stackrel{\text{def}}{=}$ let x = letrec iseven : Nat\rightarrowBool = $\lambda x:$Nat. if iszero x then true else if iszero (pred x) then false else iseven (pred (pred x)) in</pre>	





- In most programming languages, variables are mutable. I.e., a variable provides both
 - + a name that refers to a previously calculated value
 - the possibility of overwriting this value with another (which will be referred to by the same name)
- In some languages (e.g., OCaml), these two features are kept separate
 - variables are only for naming the binding between a variable and its value is immutable
 - + introduce a new class of mutable cells or references
 - at any given moment, a reference holds a value (and can be dereferenced to obtain this value)
 - a new value may be assigned to a reference

We choose OCaml's style, which is easier to work with formally.

So a variable of type T in most languages (except OCaml) will correspond to a Ref T (actually, a Ref(Option T)) here.

27

CIS 500, 21 October

