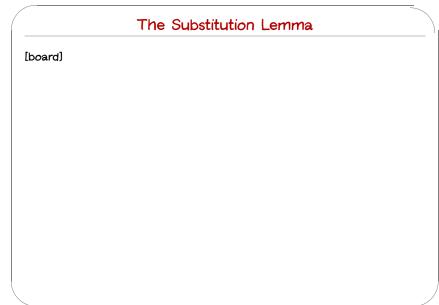


CIS 500, 16 October

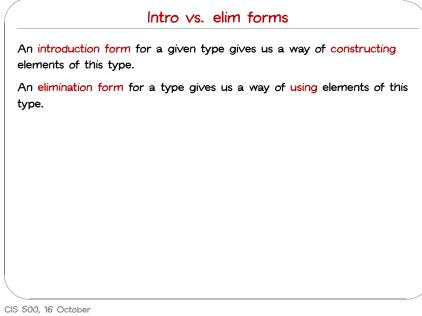
Simply typed lambda-calculus with booleans		
true : Bool	(T-TRUE	
false : Bool	(T-False	
$t_1$ : Bool $t_2$ : T $t_3$ : T	(T-IF	
if $t_1$ then $t_2$ else $t_3$ : T	(11	
$x: T \in \Gamma$	(T-VAF	
$\Gamma \vdash x : T$		
$\Gamma, \mathbf{x}: \mathbf{T}_1 \vdash \mathbf{t}_2 : \mathbf{T}_2$	(T-ABS	
$\Gamma \vdash \lambda \mathtt{x} : \mathtt{T}_1 . \mathtt{t}_2 : \mathtt{T}_1 \rightarrow \mathtt{T}_2$		
$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_2 : T_{11}}$	(T-App	
$\Gamma \vdash t_1 \ t_2 : T_{12}$		



CIS 500, 16 October

1

3



## The Curry-Howard Correspondence

In constructive logics, a proof of P must provide evidence for P.

 $\blacklozenge$  "law of the excluded middle" —  $P \lor \neg P$  — not recognized.

A proof of  $P \wedge Q$  is a pair of evidence for P and evidence for Q.

A proof of  $P \supset Q$  is a procedure for transforming evidence for P into evidence for Q.

5

propositions typ	pes
proposition $P \supset Q$ type	De P→Q
proposition $P \wedge Q$ type	be $P \times Q$
proof of proposition P ter	mt of type P
proposition P is provable typ	pe P is inhabited (by some term)

Logic	PROGRAMMING LANGUAGES	
propositions	types	
proposition $P \supset Q$	type P→Q	
proposition $P \wedge Q$	type $P \times Q$	
proof of proposition P	term t of type P	
proposition P is provable	type P is inhabited (by some term) evaluation	

CIS 500, 16 October

6

Порос	sitions as Types		Erasu	ire
LOGIC propositions proposition $P \supset Q$ proposition $P \land Q$ proof of proposition P proposition P is provable proof simplification (cut elimination)	PROGRAMMING LANGUAGES type P→Q type P × Q term t of type P type P is inhabited (by some term) evaluation		$erase(x) = 2$ $erase(\lambda x: T_1 \cdot t_2) = 2$ $erase(t_1 \cdot t_2) = 0$	
10, 16 October		7-b CIS 500	), 16 October	
	Typability			
	be typable if there is some term t in the type T, and some context $\Gamma$ such	he		

rogramming languages...

= erase(t<sub>1</sub>) erase(t<sub>2</sub>)

CIS 500, 16 October

9

CIS 500, 16 October

8

## Base types

Up to now, we've formulated "base types" (e.g. Nat) by adding them to the syntax of types, extending the syntax of terms with associated constants (zero) and operators (succ, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

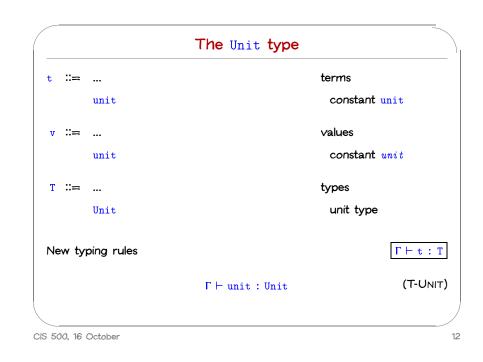
E.g., suppose B and C are some base types. Then we can ask (without knowing anything more about B or C) whether there are any types S and T such that the term

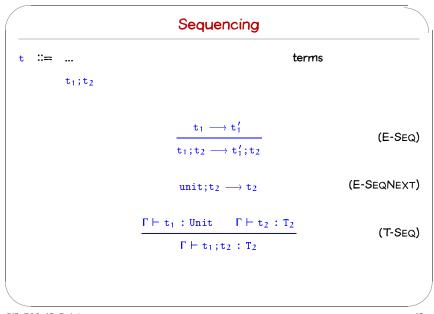
 $(\lambda f:S. \lambda g:T. f g) (\lambda x:B. x)$ 

is well typed.

CIS 500, 16 October

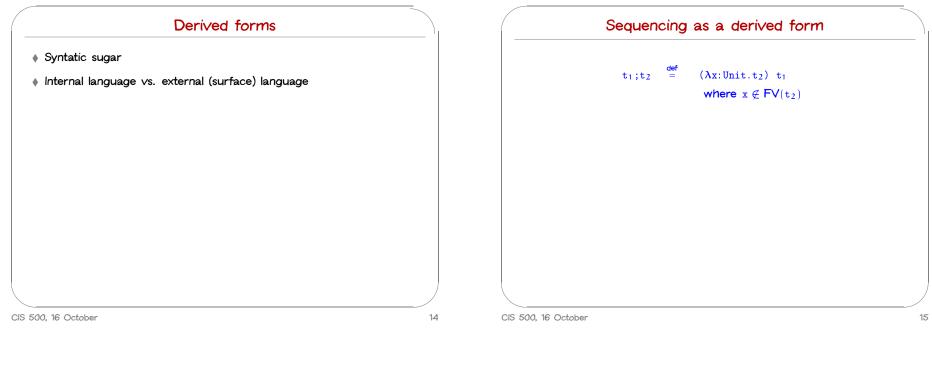
Sequencing	
t ::=	terms
t <sub>1</sub> ;t <sub>2</sub>	



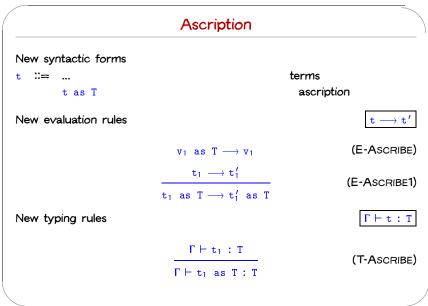


CIS 500, 16 October

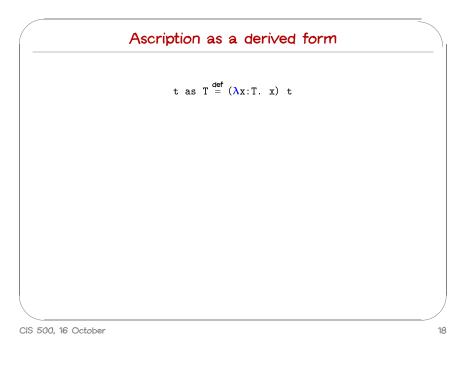
11

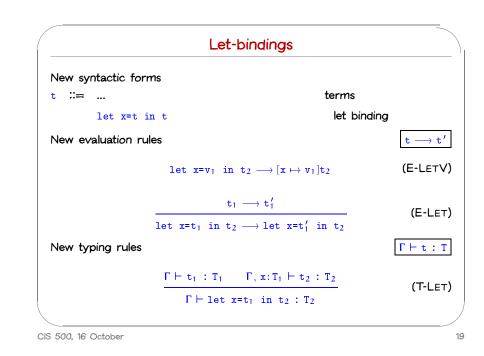


	Equivalence of the two definitions	
[board]		



16





Pairs t ::= ... terms  $\{t,t\}$ pair first projection t.1 t.2 second projection v ::= ... values {v,v} pair value T ∷= ... types  $T_1 \times T_2$ product type

Evaluation rules for pa	airs
$\{v_1, v_2\}, 1 \longrightarrow v_1$	(E-PAIRBETA1)
$\{v_1, v_2\}. 2 \longrightarrow v_2$	(E-PAIRBETA2)
$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1, 1 \longrightarrow \mathtt{t}_1', 1}$	(E-Proj1)
$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.2 \longrightarrow \mathtt{t}_1'.2}$	(E-PROJ <b>2</b> )
$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\{\mathtt{t}_1, \mathtt{t}_2\} \longrightarrow \{\mathtt{t}_1', \mathtt{t}_2\}}$	(E-PAIR1)
$\mathtt{t}_2 \longrightarrow \mathtt{t}_2'$	(E-PAIR2)
$\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}$	

