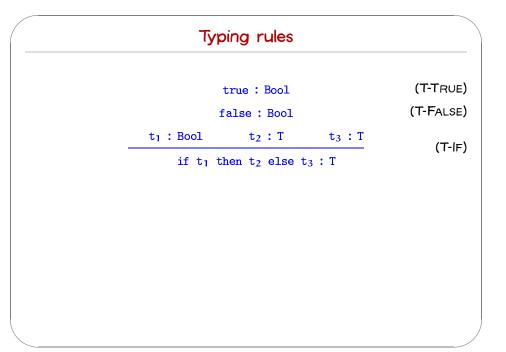
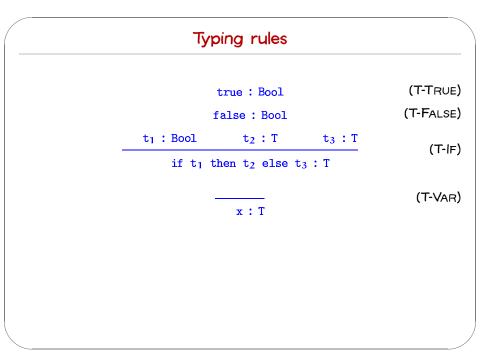
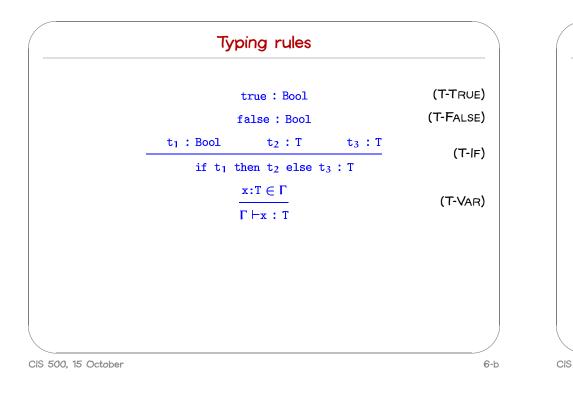
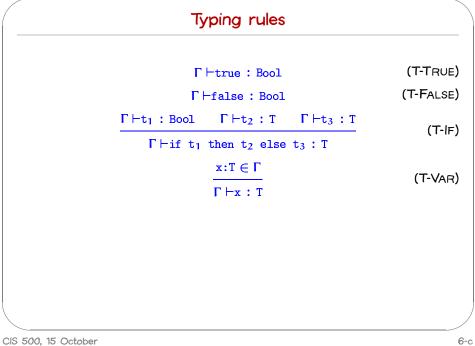


Lambda-calculus with booleans		"Simple Types"	
; ::=	terms	T ::=	types
x	variable	Bool	type of booleans
λ x.t	abstraction	$T \rightarrow T$	types of functions
tt	application		-
true	constant true		
false	constant false		
if t then t else t	conditional		
=	values		
λ x.t	abstraction value		
true	true value		
false	false value		
500, 15 October	4	CIS 500, 15 October	









Typing rules $\Gamma \vdash true : Bool$ (T-TRUE) $\Gamma \vdash false : Bool$ (T-FALSE) $\Gamma \vdash t_1 : Bool$ $\Gamma \vdash t_2 : T$ $\Gamma \vdash t_3 : T$ $\Gamma \vdash if t_1$ then t_2 else $t_3 : T$ (T-IF) $\Gamma \vdash if t_1$ then t_2 else $t_3 : T$ (T-VAR) $\Gamma \vdash x : T$ (T-VAR) $\Gamma, x:T_1 \vdash t_2 : T_2$ (T-ABS) $\Gamma \vdash \lambda x:T_1 . t_2 : T_1 \rightarrow T_2$

Typing rules	
Γ⊢true : Bool Γ⊢false : Bool	(T-True) (T-False)
$\frac{\Gamma \vdash t_1 : \text{Bool} \Gamma \vdash t_2 : T \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}$	(T-IF)
$\frac{\mathbf{x}:T\in\Gamma}{\Gamma\vdashx:T}$	(T-VAR)
$\frac{\Gamma, \mathtt{x}: \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}: \mathtt{T}_1 \cdot \mathtt{t}_2 : \mathtt{T}_1 \rightarrow \mathtt{T}_2}$	(T-ABS)
$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \ t_2 : T_{12}}$	(Т-Арр)

Typing Derivations

What derivations justify the following typing statements?

- \vdash (λ x:Bool.x) true : Bool
- f:Bool→Bool ⊢ f (if false then true else false) : Bool
- f:Bool \rightarrow Bool $\vdash \lambda x$:Bool. f (if x then false else x) : Bool \rightarrow Bool

Properties of λ_{\rightarrow}

As before, the fundamental property of the type system we have just defined is soundness with respect to the operational semantics.

CIS 500, 15 October

7

Properties of λ_{\rightarrow}

As before, the fundamental property of the type system we have just defined is soundness with respect to the operational semantics.

- 1. Progress: A closed, well-typed term is not stuck
 - If $\vdash t : T$, then either t is a value or else $t \longrightarrow t'$ for some t'.
- 2. Preservation: Types are preserved by one-step evaluation

```
If \Gamma \vdash t: T and t \longrightarrow t', then \Gamma \vdash t': T.
```

Proving progress	\
Same steps as before	
)

CIS 500, 15 October

Proving progress

Same steps as before...

- inversion lemma for typing relation
- canonical forms lemma
- progress theorem

CIS 500, 15 October

9-a

Inversion

Lemma:

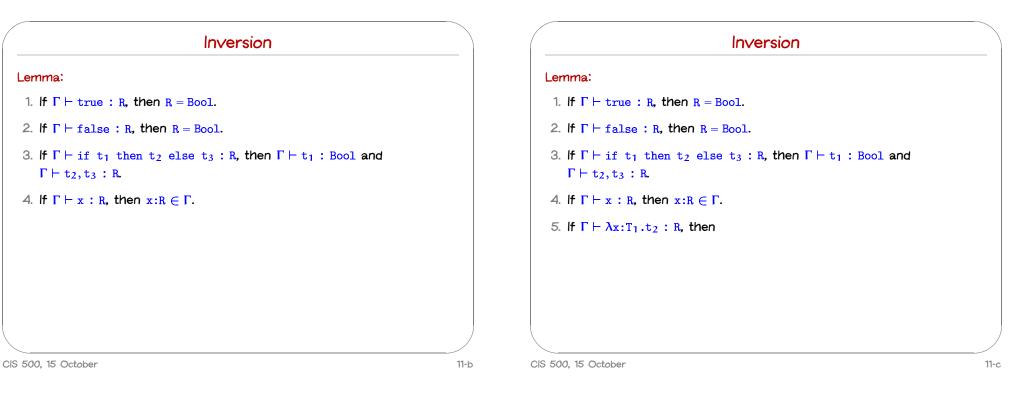
- 1. If $\Gamma \vdash \text{true}$: R, then R = Bool.
- 2. If $\Gamma \vdash false : R$, then R = Bool.
- 3. If $\Gamma \vdash$ if t_1 then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 :$ Bool and $\Gamma \vdash t_2, t_3 : R$.

Lemma: 1. If $\Gamma \vdash true : R$, then R = Bool. 2. If $\Gamma \vdash false : R$, then R = Bool. 3. If $\Gamma \vdash if t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 : Bool$ and $\Gamma \vdash t_2, t_3 : R$. 4. If $\Gamma \vdash x : R$, then

Typing rules again (for reference)

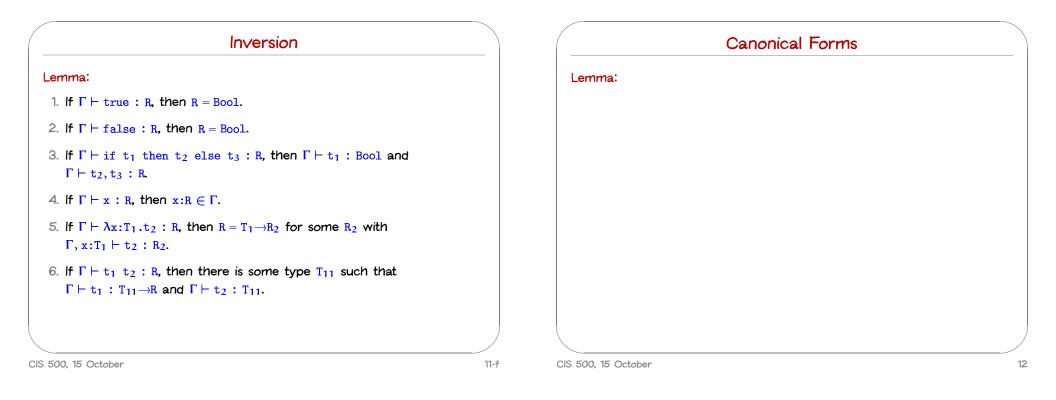
Γ⊢ true : Bool Γ⊢ false : Bool	(T-TRUE) (T-False)
$\frac{\Gamma \vdash t_1 : \text{Bool} \Gamma \vdash t_2 : T \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}$	(T-IF)
$\frac{\mathbf{x}:\mathbf{T}\in\boldsymbol{\Gamma}}{\boldsymbol{\Gamma}\vdash\mathbf{x}:\mathbf{T}}$	(T-VAR)
$\frac{\Gamma, \mathbf{x}: \mathbf{T}_1 \vdash \mathbf{t}_2 : \mathbf{T}_2}{\Gamma \vdash \lambda \mathbf{x}: \mathbf{T}_1 \cdot \mathbf{t}_2 : \mathbf{T}_1 \rightarrow \mathbf{T}_2}$	(T-ABS)
$\frac{\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 : t_2 : T_{12}}$	(T-App)

CIS 500, 15 October



Inversion	
	Lemma:
	1. If $\Gamma \vdash \text{true}$: R, then $R = Bool$.
	2. If $\Gamma \vdash false : R$, then $R = Bool$.
	3. If $\Gamma \vdash if t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 :$ Bool and $\Gamma \vdash t_2, t_3 : R$.
	4. If $\Gamma \vdash x : R$, then $x:R \in \Gamma$.
	5. If $\Gamma \vdash \lambda x: T_1.t_2$: R, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x: T_1 \vdash t_2$: R_2 .

Inversion		
emma:		
1. If $\Gamma \vdash \text{true}$: R, then $R = Bool$.		
2. If $\Gamma \vdash false : R$, then $R = Bool$.		
3. If $\Gamma \vdash if t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 :$ Bool and $\Gamma \vdash t_2, t_3 : R$.		
1. If $\Gamma \vdash x : R$, then $x: R \in \Gamma$.		
5. If $\Gamma \vdash \lambda x: T_1 \cdot t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x: T_1 \vdash t_2 : R_2$.		
B. If $\Gamma \vdash t_1 \ t_2 : R$, then		



Canonical Forms

Lemma:

1. If v is a value of type Bool, then

Canonical Forms

Lemma:

1. If v is a value of type Bool, then v is either true or false.

Canonical Forms Canonical Forms Lemma: Lemma: 1. If v is a value of type Bool, then v is either true or false. 1. If v is a value of type Bool, then v is either true or false. 2. If v is a value of type $T_1 \rightarrow T_2$, then 2. If v is a value of type $T_1 \rightarrow T_2$, then v has the form $\lambda x: T_1 \cdot t_2$. CIS 500, 15 October 12-c CIS 500, 15 October 12-d Progress Progress Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with t \rightarrow t'.

Proof: By induction

some T). Then either t is a value or else there is some t' with t \longrightarrow t'.

Proof: By induction on typing derivations.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1 \ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$.

CIS 500, 15 October

13-b

CIS 500, 15 October

Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1 \ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either t_1 is a value or else it can make a step of evaluation, and likewise t_2 .

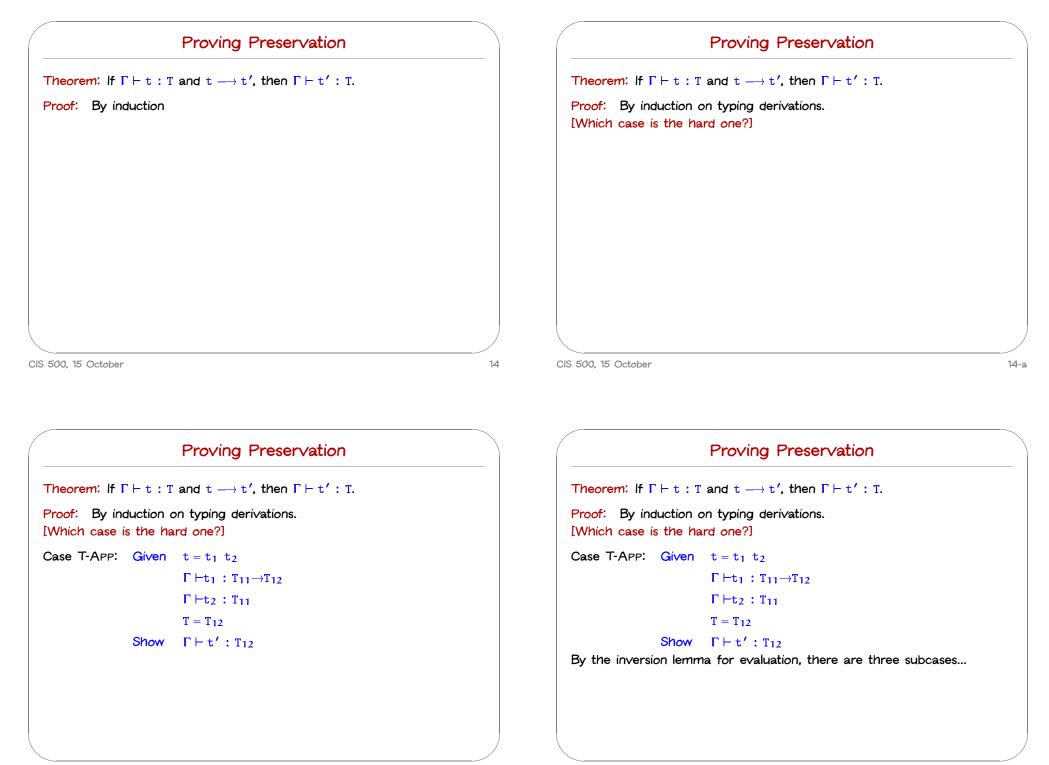
Progress

Theorem: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t' with $t \longrightarrow t'$.

Proof: By induction on typing derivations. The cases for boolean constants and conditions are the same as before. The variable case is trivial (because t is closed). The abstraction case is immediate, since abstractions are values.

Consider the case for application, where $t = t_1 \ t_2$ with $\vdash t_1 : T_{11} \rightarrow T_{12}$ and $\vdash t_2 : T_{11}$. By the induction hypothesis, either t_1 is a value or else it can make a step of evaluation, and likewise t_2 . If t_1 can take a step, then rule E-APP1 applies to t. If t_1 is a value and t_2 can take a step, then rule E-APP2 applies. Finally, if both t_1 and t_2 are values, then the canonical forms lemma tells us that t_1 has the form $\lambda x:T_{11}.t_{12}$, and so rule E-APPABS applies to t.

13-c



Proving Preservation

Theorem: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$. Proof: By induction on typing derivations. [Which case is the hard one?] Case T-APP: Given $t = t_1 t_2$ $\Gamma \vdash t_1 : T_{11} \rightarrow T_{12}$ $\Gamma \vdash t_2 : T_{11}$ $T = T_{12}$ Show $\Gamma \vdash t' : T_{12}$ By the inversion lemma for evaluation, there are three subcases... Subcase: $t_1 = \lambda x : T_{11} . t_{12}$ t_2 a value v_2 $t' = [x \mapsto v_2]t_{12}$

CIS 500, 15 October

14-d

The "Substitution Lemma"

Lemma: Types are preserved under substitution.

If Γ , x:S \vdash t : T and Γ \vdash s : S, then Γ \vdash $[x \mapsto s]t$: T.

	Proving Preservation		
Theorem: If $\Gamma \vdash t$: T and t \longrightarrow t', then $\Gamma \vdash$ t' : T.		
Proof: By induction [Which case is the	on on typing derivations. • hard one?]		
Case T-APP: Give	en $t = t_1 t_2$		
	$\Gamma \vdash t_1 : T_{11} \rightarrow T_{12}$		
	$\Gamma \vdash t_2 : T_{11}$		
	$T = T_{12}$		
Sho	w $\Gamma \vdash t': T_{12}$		
By the inversion le	mma for evaluation, there are three subcases		
Subcase: $t_1 = \lambda x$:T ₁₁ . t ₁₂		
t ₂ a va	alue v ₂		
t' = [x +	\mapsto v ₂]t ₁₂		
Uh oh.			

CIS 500, 15 October



If Γ , x:S \vdash t : T and Γ \vdash s : S, then Γ \vdash [x \mapsto s]t : T.

Proof: ...

14-e

Preservation

Homework: Complete the proof of preservation



CIS 500, 15 October

16

Intro vs. elim forms

An introduction form for a given type gives us a way of constructing elements of this type.

An elimination form for a type gives us a way of using elements of this type.

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.

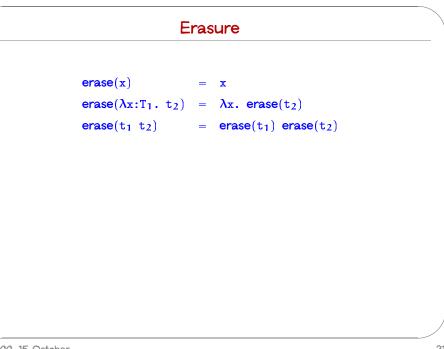
A proof of $P \supset Q$ is a procedure for transforming evidence for P into evidence for Q.

Propositions as Types			
Logic	PROGRAMMING LANGUAGES		
propositions	types		
proposition $P \supset Q$	type P→Q		
proposition $\mathbf{P} \wedge \mathbf{Q}$	type P × Q		
proof of proposition P	term t of type P		
proposition P is provable	type P is inhabited (by some term)		

Propositions a	as Types
----------------	----------

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

	_
Logic	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type P × Q
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)
proof simplification	evaluation



CIS 500, 15 October

20-a

Typability

An untyped λ -term m is said to be typable if there is some term t in the simply typed lambda-calculus, some type T, and some context Γ such that erase(t) = m and $\Gamma \vdash t$: T.

Cf. type reconstruction in OCaml.

On to real programming languages...

CIS 500, 15 October

22

CIS 500, 15 October

Base types

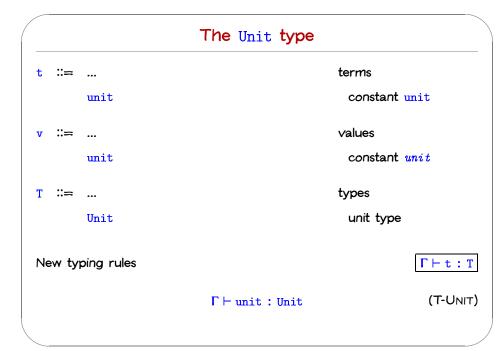
Up to now, we've formulated "base types" (e.g. Nat) by adding them to the syntax of types, extending the syntax of terms with associated constants (zero) and operators (succ, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

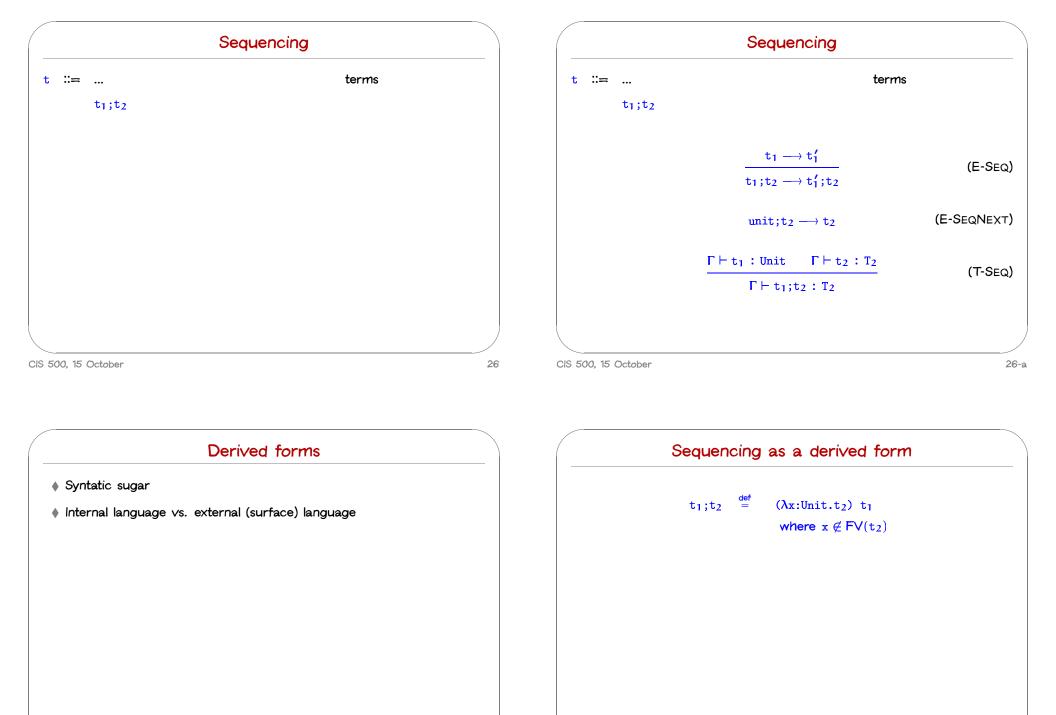
For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

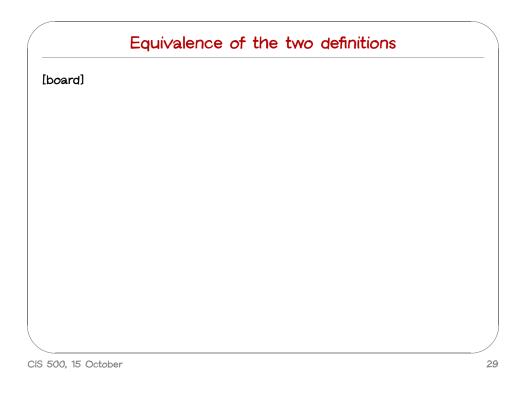
E.g., suppose B and C are some base types. Then we can ask (without knowing anything more about B or C) whether there are any types S and T such that the term

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

is well typed.

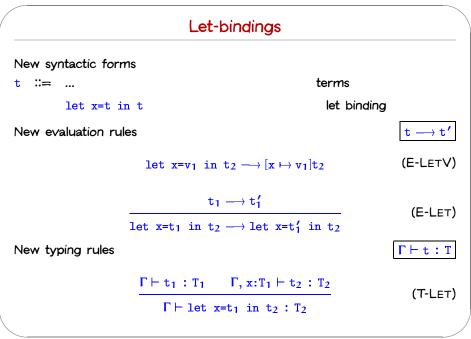


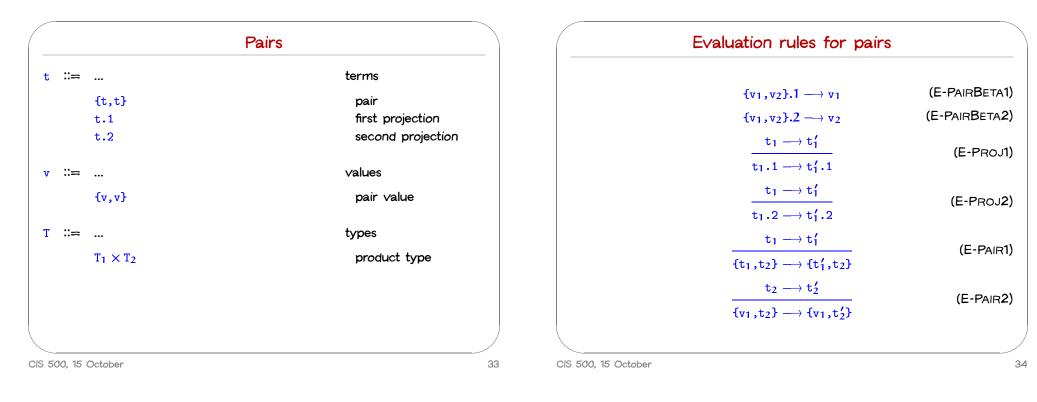




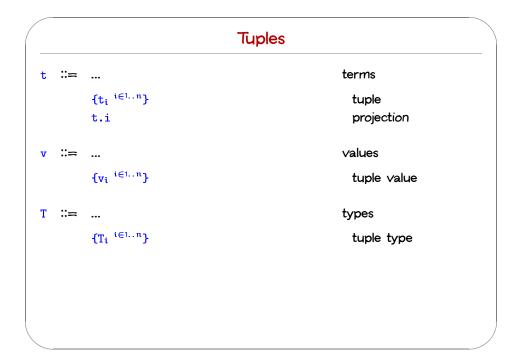
	Ascription	
New syntactic forms		
t ∷= t as T		terms ascription
New evaluation rules		$\texttt{t} \longrightarrow \texttt{t'}$
	v_1 as $T \longrightarrow v_1$	(E-Ascribe)
	$\frac{t_1 \longrightarrow t'_1}{t_1 \text{ as } T \longrightarrow t'_1 \text{ as } T}$	(E-Ascribe1)
New typing rules		$\Gamma \vdash t : T$
	$\frac{\Gamma \vdash t_1 : T}{\Gamma \vdash t_1 \text{ as } T : T}$	(T-Ascribe)
~		

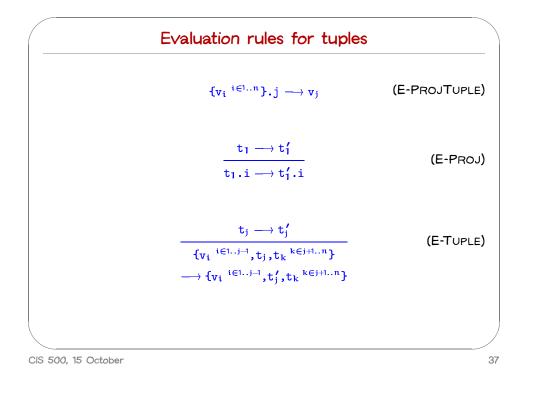
Ascription as a derived form t as $T \stackrel{\text{def}}{=} (\lambda x:T. x) t$

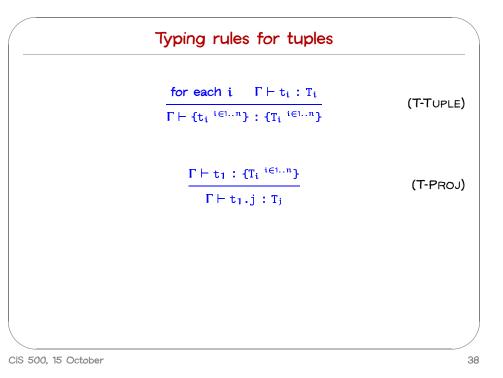


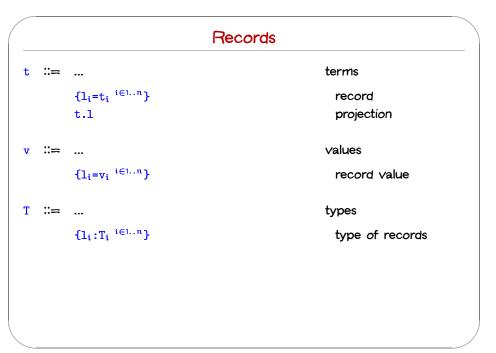


 Typing rules for pairs	
$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2}$	(T-Pair)
$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}}$	(T-PROJ1)
$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}}$	(T-Proj2)









Evaluation rules for records		
$\{l_i = v_i^{i \in 1n}\} \cdot l_j \longrightarrow v_j$	(E-PROJRCD)	
$\frac{\mathtt{t_1} \longrightarrow \mathtt{t'_1}}{\mathtt{t_1.l} \longrightarrow \mathtt{t'_1.l}}$	(E-Proj)	
$ \begin{array}{c} t_{j} \longrightarrow t_{j}' \\ \hline \\ \hline \{l_{i}=v_{i} \stackrel{i\in 1j\rightarrow i}{\rightarrow}, l_{j}=t_{j}, l_{k}=t_{k} \stackrel{k\in j+1n}{\rightarrow} \} \\ \longrightarrow \{l_{i}=v_{i} \stackrel{i\in 1j\rightarrow i}{\rightarrow}, l_{j}=t_{j}', l_{k}=t_{k} \stackrel{k\in j+1n}{\rightarrow} \} \end{array} $	(E-Rcd)	

	Typing rules for records	
	$\frac{\text{for each } i \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i \ ^{i \in 1n}\} : \{l_i : T_i \ ^{i \in 1n}\}}$	(T-Rcd)
	$\frac{\Gamma \vdash t_1 \ : \ \{l_i: T_i^{-i \in 1n}\}}{\Gamma \vdash t_1 . l_j \ : \ T_j}$	(T-PROJ)
S 500, 15 October		