

CIS 500, 15 October

Plans

Where we've been:

- Inductive definitions
 - abstract syntax
 - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

Plans

Where we've been:

- Inductive definitions
 - abstract syntax
 - inference rules
- Proofs by structural induction
- Operational semantics
- The lambda-calculus
- Typing rules and type soundness

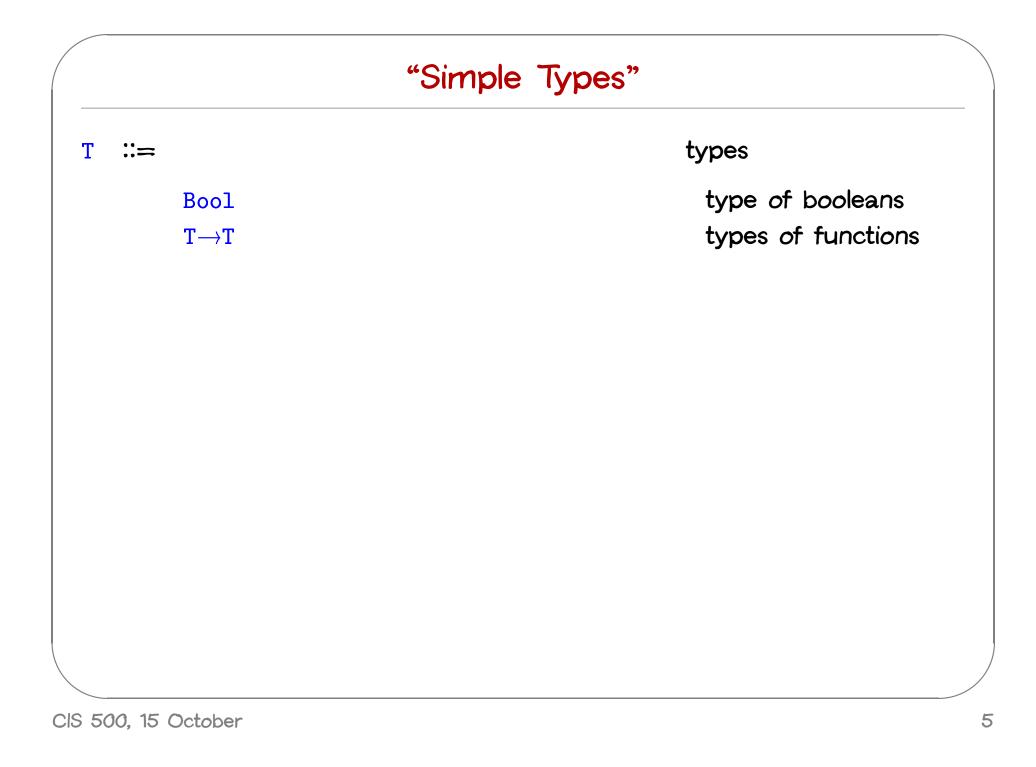
Where we're going:

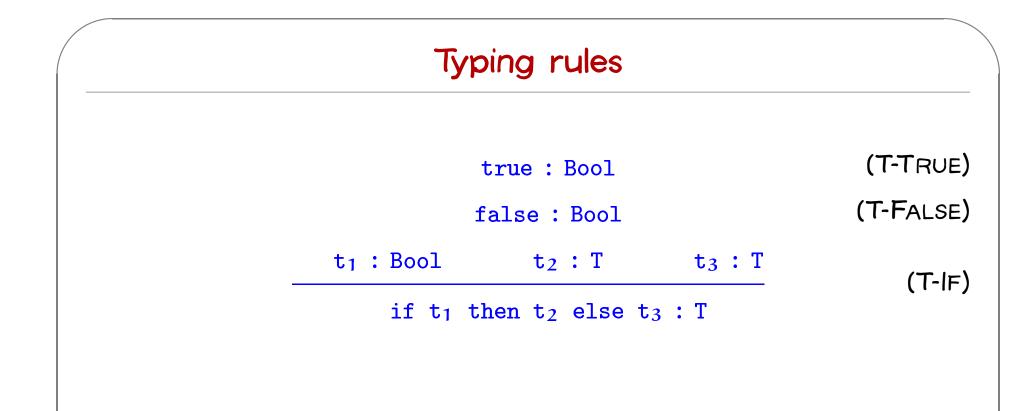
- Simple types" for the lambda-calculus
- Formalizing more features of real-world languages (records, datatypes, references, exceptions, etc.)
- Subtyping
- Objects

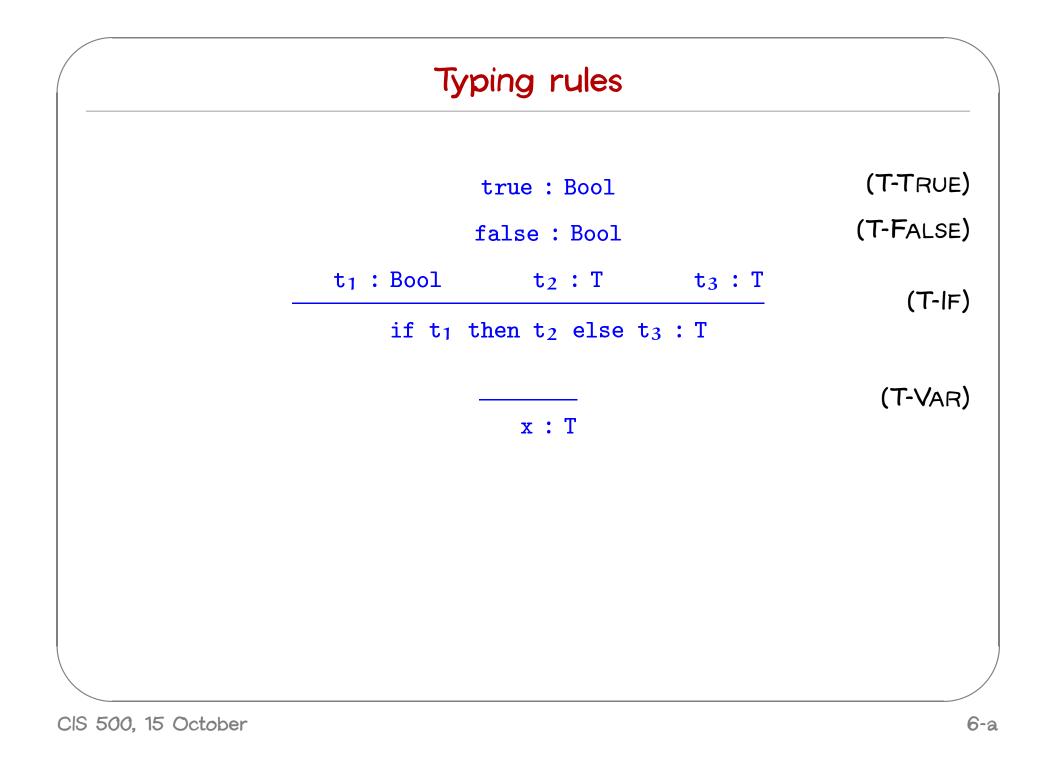


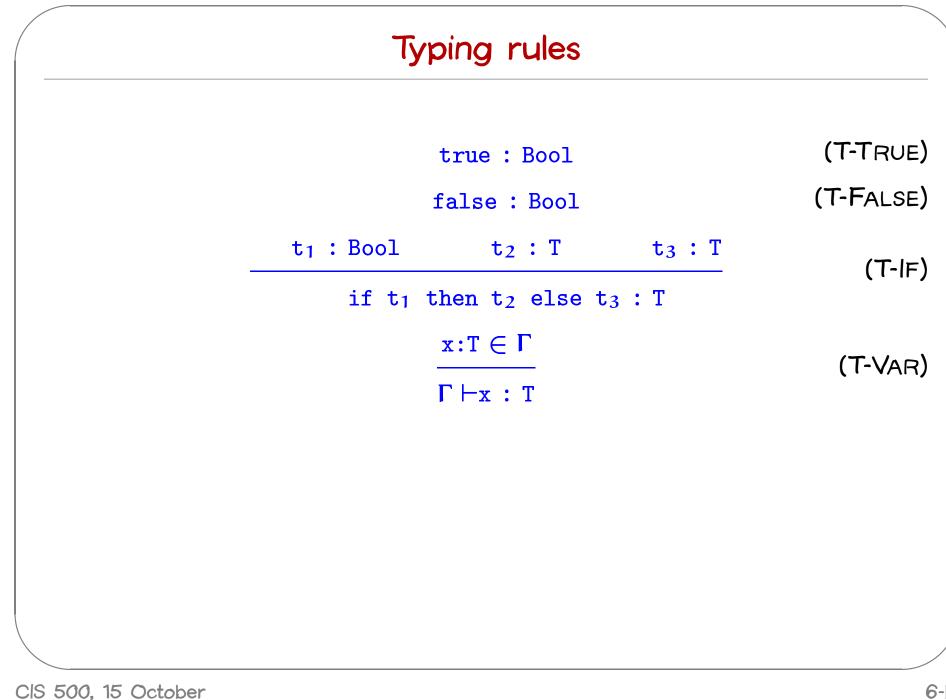
Lambda-calculus with booleans

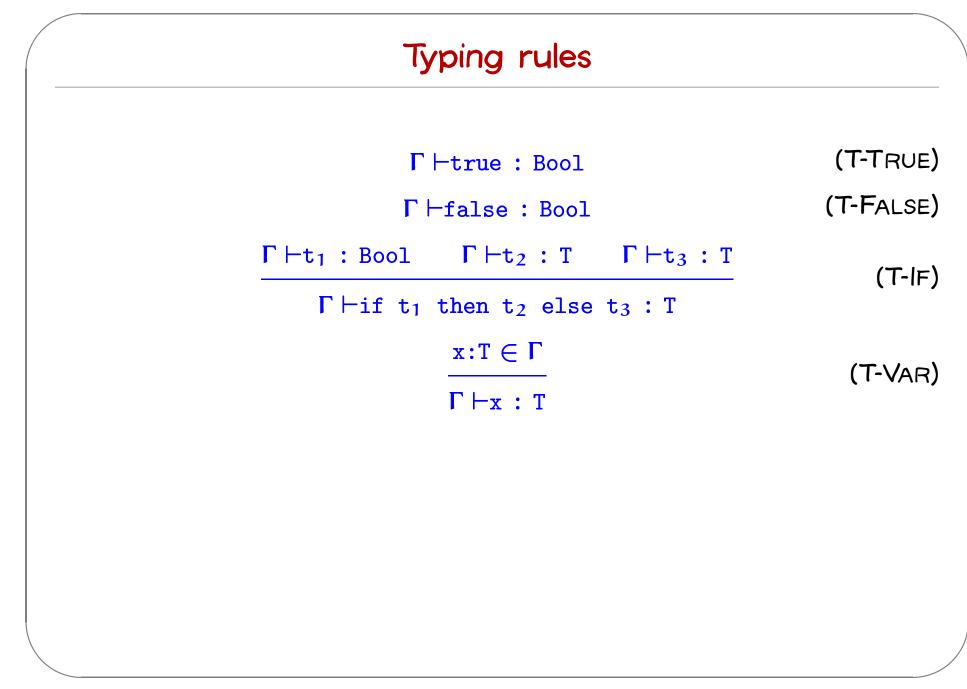
t ::=		terms
	x	variable
	λ x.t	abstraction
	t t	application
	true	constant true
	false	constant false
	if t then t else t	conditional
v ∷=		values
	$\lambda x.t$	abstraction value
	true	true value
	false	false value



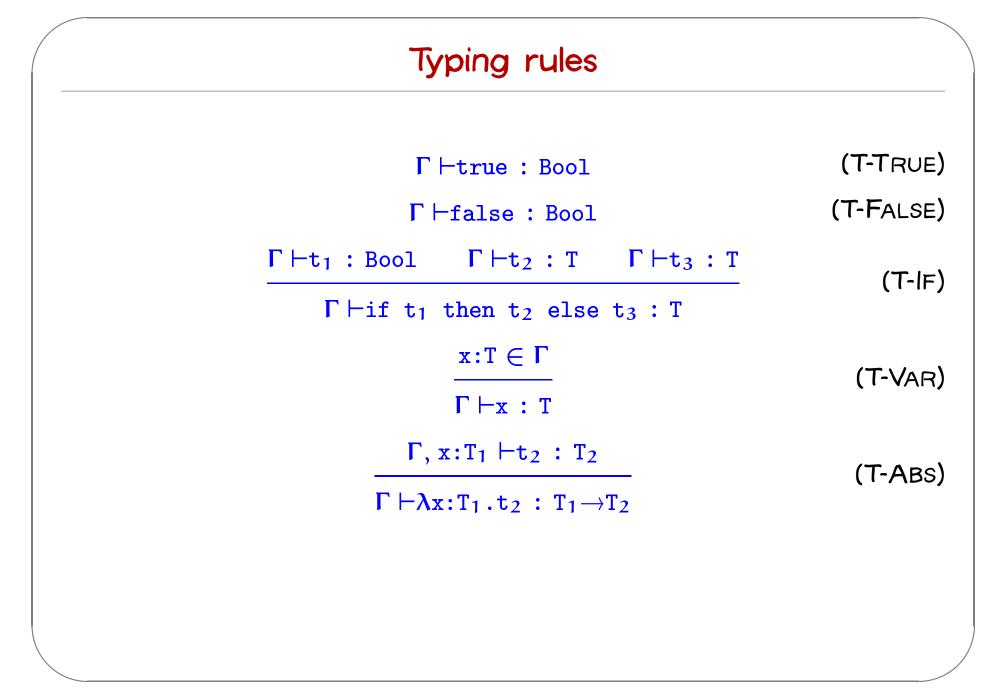


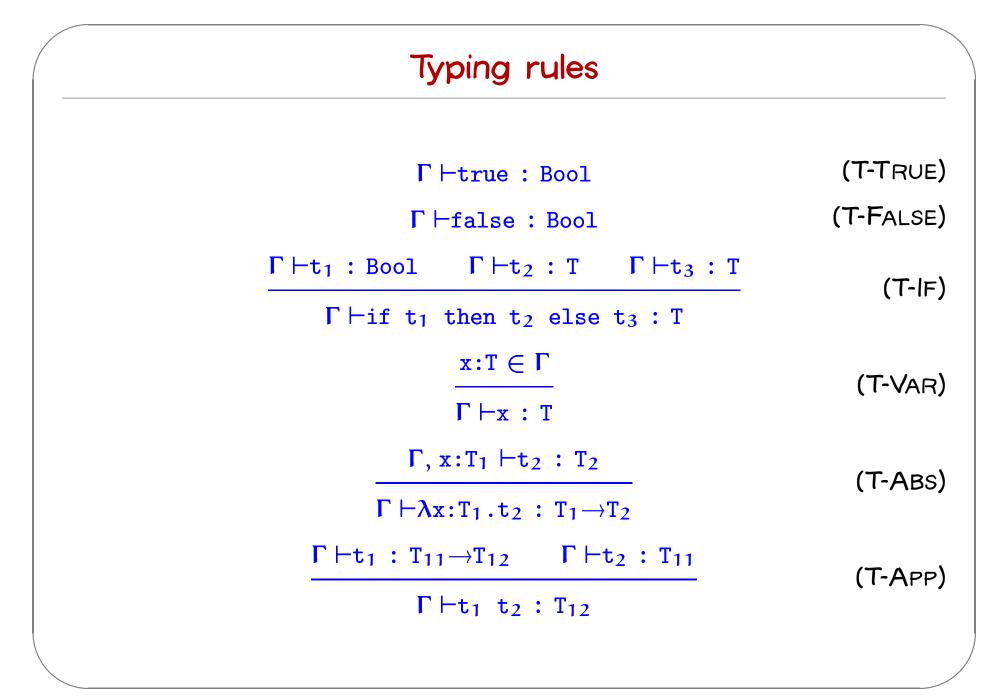






CIS 500, 15 October





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.

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...

Proving progress

Same steps as before...

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

Typing rules again (for reference)

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

CIS 500, 15 October

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

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

- 1. If $\Gamma \vdash \text{true}$: R, then R = Bool.
- 2. If $\Gamma \vdash$ false : R, then R = Bool.
- 3. If $\Gamma \vdash \text{if } t_1$ 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 $x : R \in \Gamma$.

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

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

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

- 1. If $\Gamma \vdash \text{true}$: R, then R = Bool.
- 2. If $\Gamma \vdash false : R$, then R = Bool.
- 3. If $\Gamma \vdash \text{if } t_1$ then t_2 else $t_3 : R$, then $\Gamma \vdash t_1 : Bool and$ $<math>\Gamma \vdash t_2, t_3 : R$.
- 4. If $\Gamma \vdash x : R$, then $x : R \in \Gamma$.
- 5. If $\Gamma \vdash \lambda x: T_1 \cdot t_2 : R$, then $R = T_1 \rightarrow R_2$ for some R_2 with $\Gamma, x: T_1 \vdash t_2 : R_2$.
- 6. If $\Gamma \vdash t_1 \ t_2 : R$, then there is some type T_{11} such that $\Gamma \vdash t_1 : T_{11} \rightarrow R$ and $\Gamma \vdash t_2 : T_{11}$.

Lemma:

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

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

- 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 v has the form $\lambda x: T_1.t_2$.

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

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.

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.

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 $\vdash t_2 : T_{11}$.

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 .

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.

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

Proof: By induction

```
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?]

```
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}
```

```
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...
```

```
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
                \mathbf{t'} = [\mathbf{x} \mapsto \mathbf{v}_2]\mathbf{t}_{12}
```

```
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
               \mathbf{t'} = [\mathbf{x} \mapsto \mathbf{v}_2]\mathbf{t}_{12}
Uh oh.
```

The "Substitution Lemma"

Lemma: Types are preserved under substitution.

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

The "Substitution Lemma"

Lemma: Types are preserved under substitution.

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

Proof: ...

Preservation

Homework: Complete the proof of preservation

Discussion

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.

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

A proof of $P \land Q$ 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

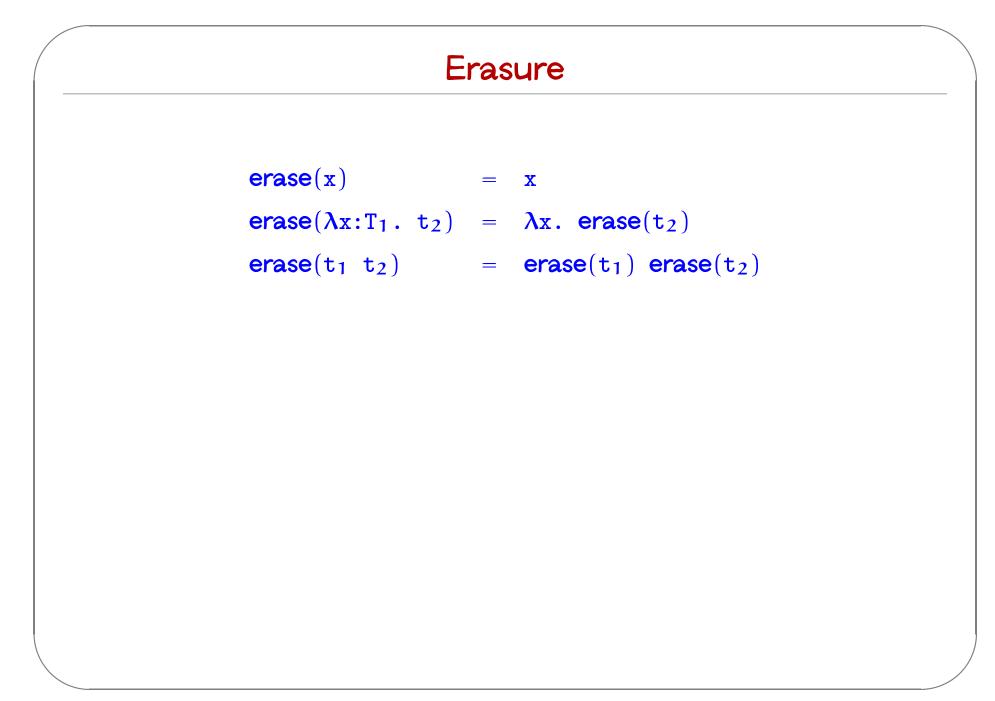
LOGIC	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type $P \times Q$
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)
proposition P is provable	type P is inhabited (by some term

Propositions as Types

roposition $P \supset Q$ type $P \rightarrow Q$ roposition $P \land Q$ type $P \times Q$ roof of proposition Pterm t of type P	$\begin{array}{llllllllllllllllllllllllllllllllllll$	proposition $P \supset Q$ t	
roposition $P \land Q$ type $P \times Q$ roof of proposition Pterm t of type P	on $P \wedge Q$ type $P \times Q$ proposition P term t of type P on P is provable type P is inhabited (by some term)		ype P→Q
roof of proposition P term t of type P	proposition P term t of type P on P is provable type P is inhabited (by some term)		
	on P is provable type P is inhabited (by some term)	proposition $P \land Q$ t	ype $P \times Q$
		proof of proposition P to	erm t of type P
oposition P is provable type P is inhabited (by some terr	evaluation	proposition P is provable t	ype P is inhabited (by some term)
evaluation		e	valuation

Propositions as Types

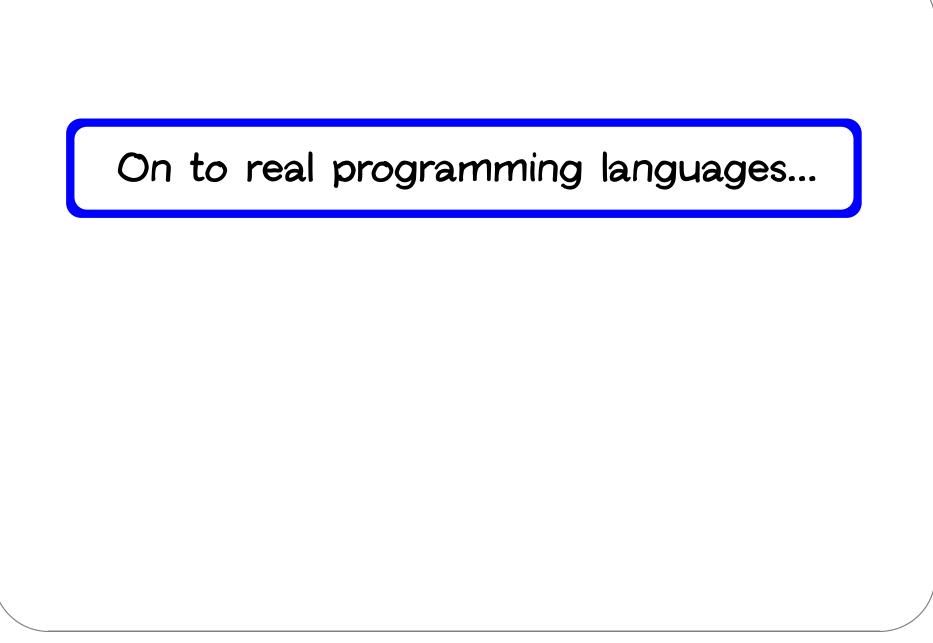
Logic	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type P × Q
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)
proof simplification	evaluation
(a.K.a. "cut elimination")	



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.



Base types

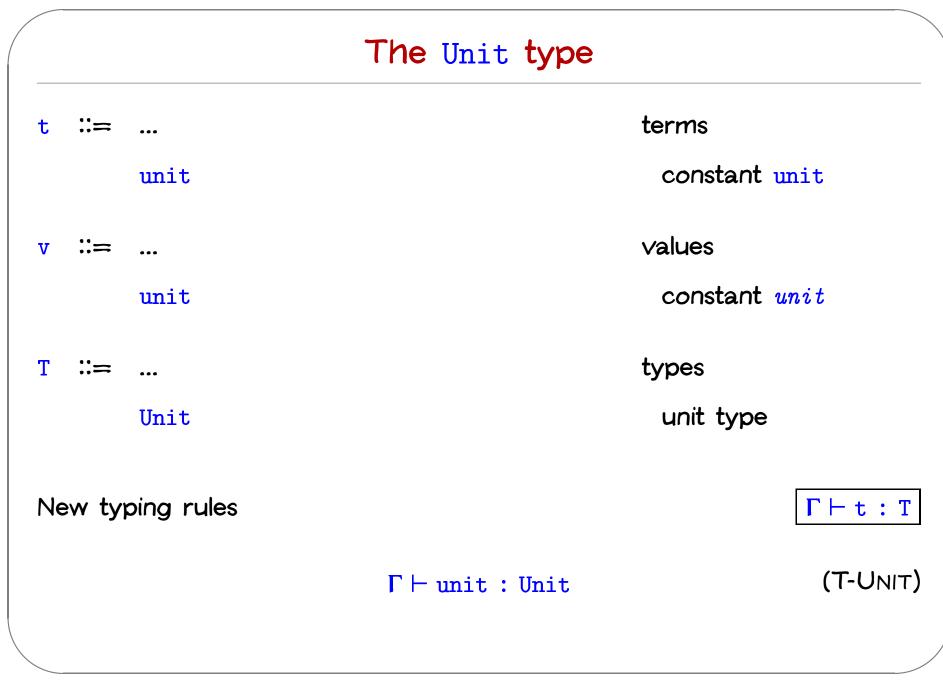
Up to now, we've formulated "base types" (e.g. Nat) by adding them to the syntax of types, extending the syntax of terms with associated constants (zero) and operators (succ, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

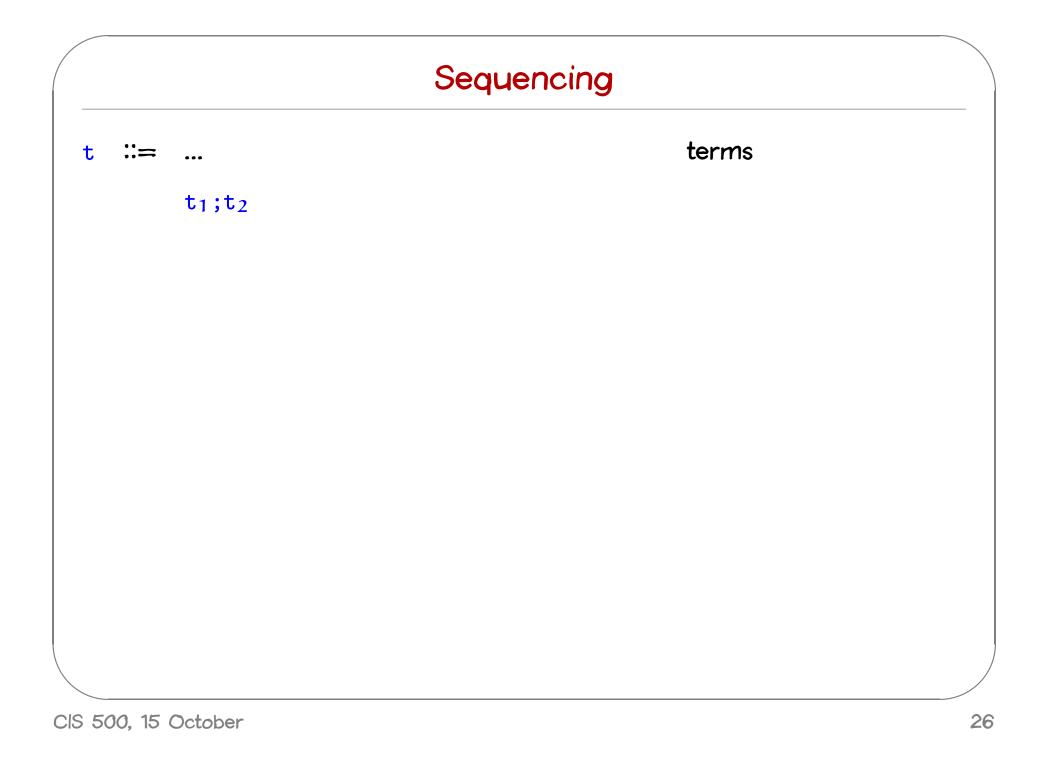
For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

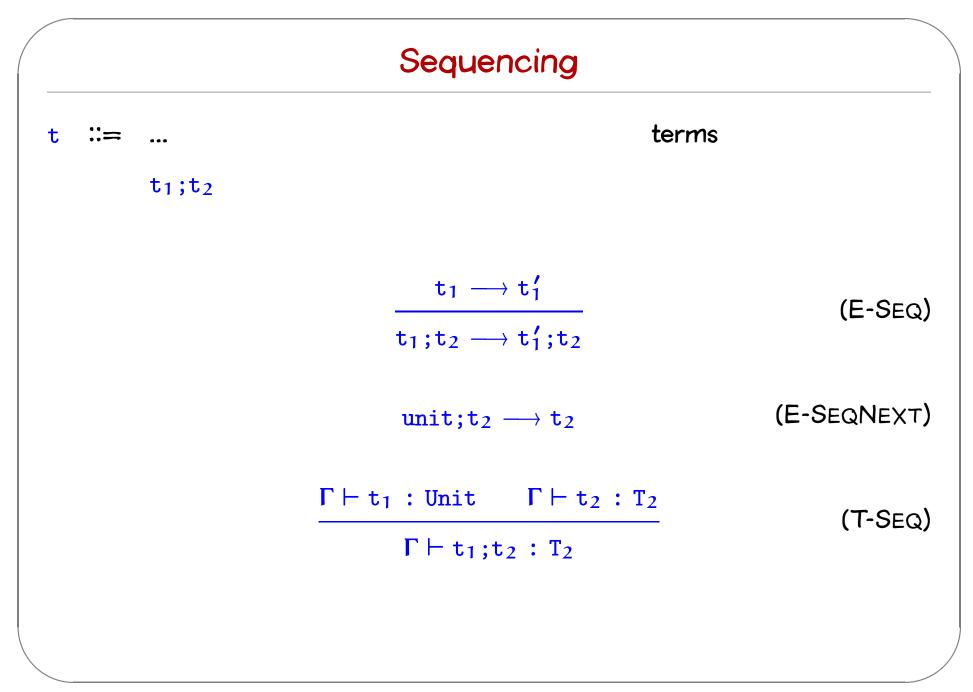
E.g., suppose B and C are some base types. Then we can ask (without knowing anything more about B or C) whether there are any types S and T such that the term

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

is well typed.



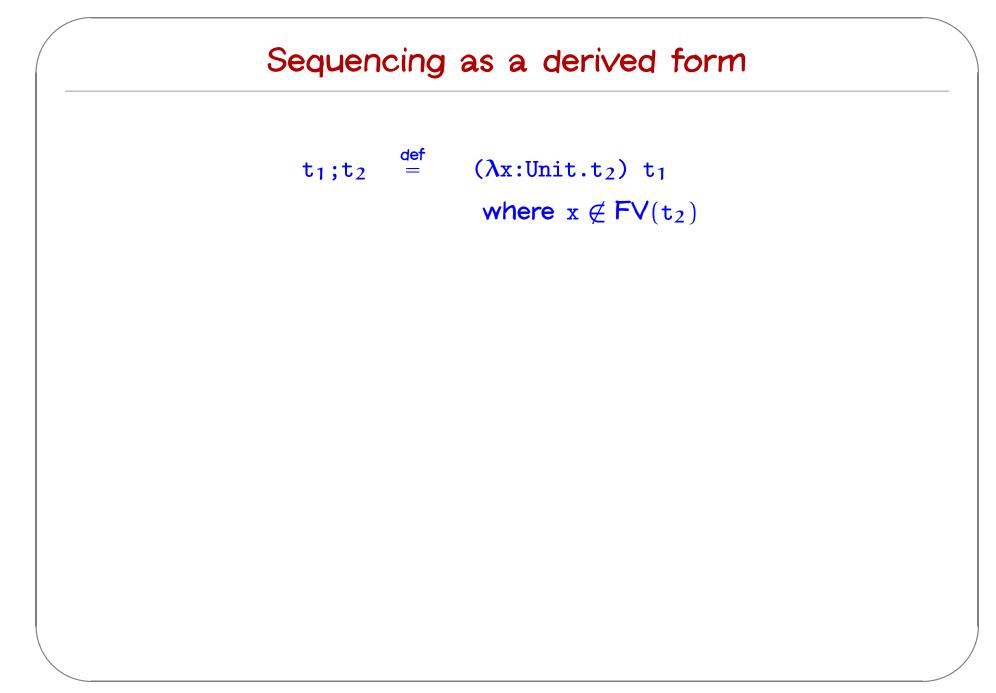




Derived forms

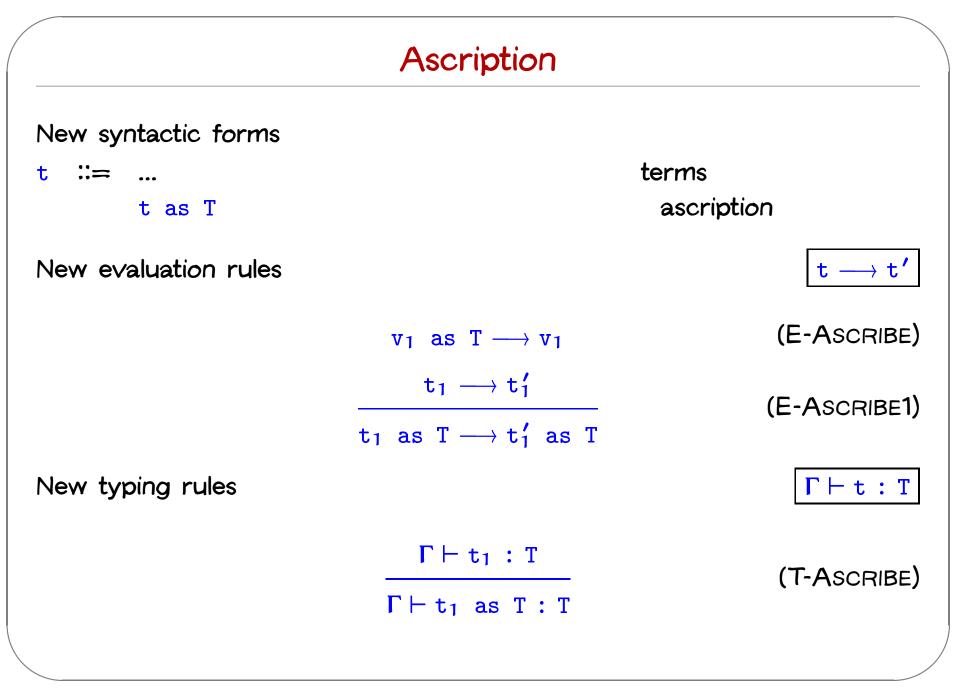
♦ Syntatic sugar

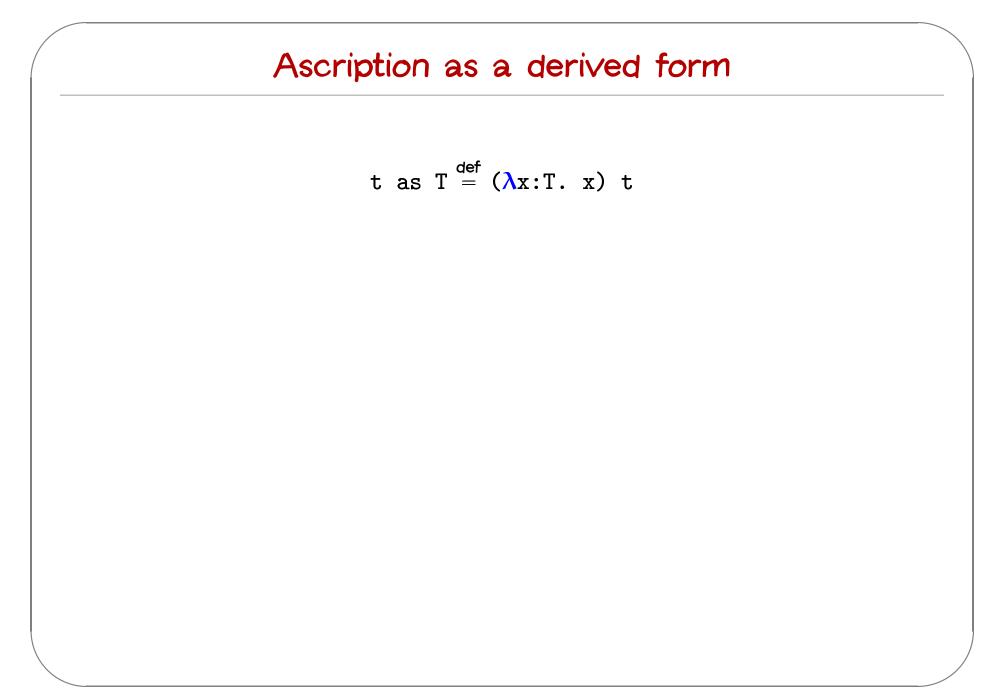
♦ Internal language vs. external (surface) language

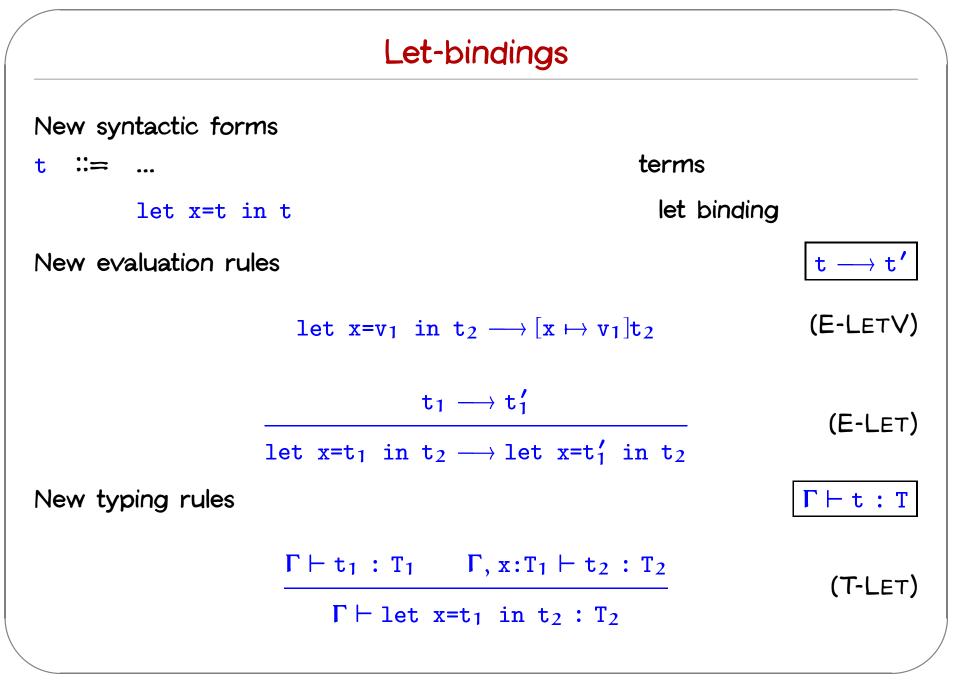


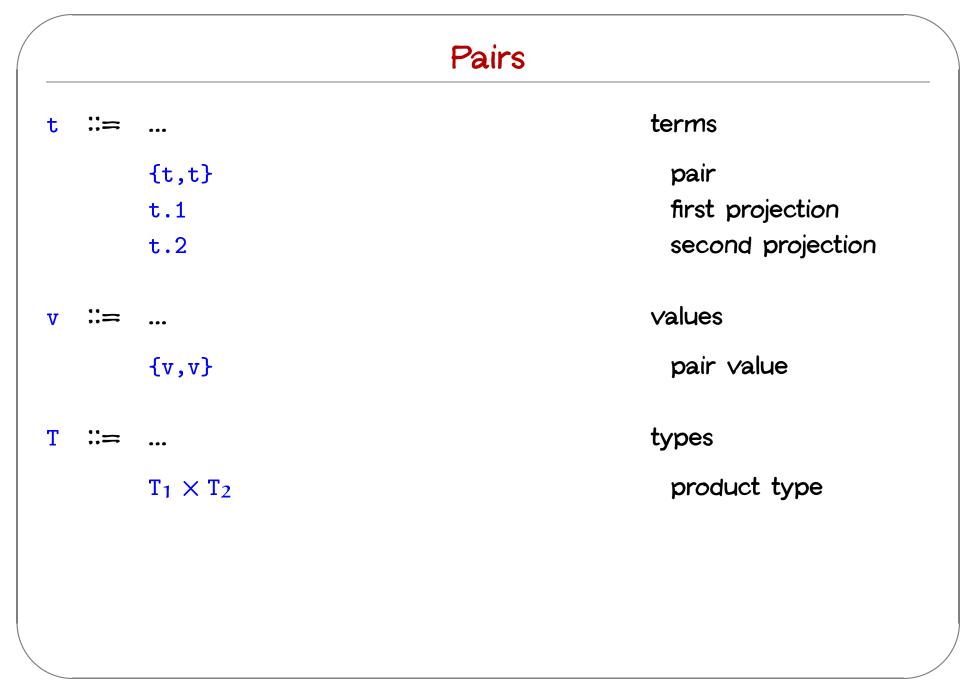
Equivalence of the two definitions

[board]









Evaluation rules for pairs

(E-PAIRBETA1) (E-PAIRBETA2)
(E-Proj1)
(E-PROJ 2)
(E-Pair1)
(E-Pair2)

