

Administrivia

- ♦ Exams will be graded over the weekend
- ♦ HW5??

(T-TRUE	
·	true : Bool
(T-False	false : Bool
(T-IF	$t_1: Bool t_2: T t_3: T$
	if t_1 then t_2 else t_3 : T
(T-VAF	$x: T \in \Gamma$
(I-VAP	$\Gamma \vdash \mathbf{x} : \mathbf{T}$
(T-ABS	$\Gamma, \mathtt{x}:\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2$
	$\overline{\Gamma \vdash \lambda \mathbf{x} : \mathbf{T}_1 \cdot \mathbf{t}_2 : \mathbf{T}_1 \rightarrow \mathbf{T}_2}$
(T-App	$\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}$
	$\Gamma \vdash t_1 \ t_2 : T_{12}$

The Substitution Lemma

[board]

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

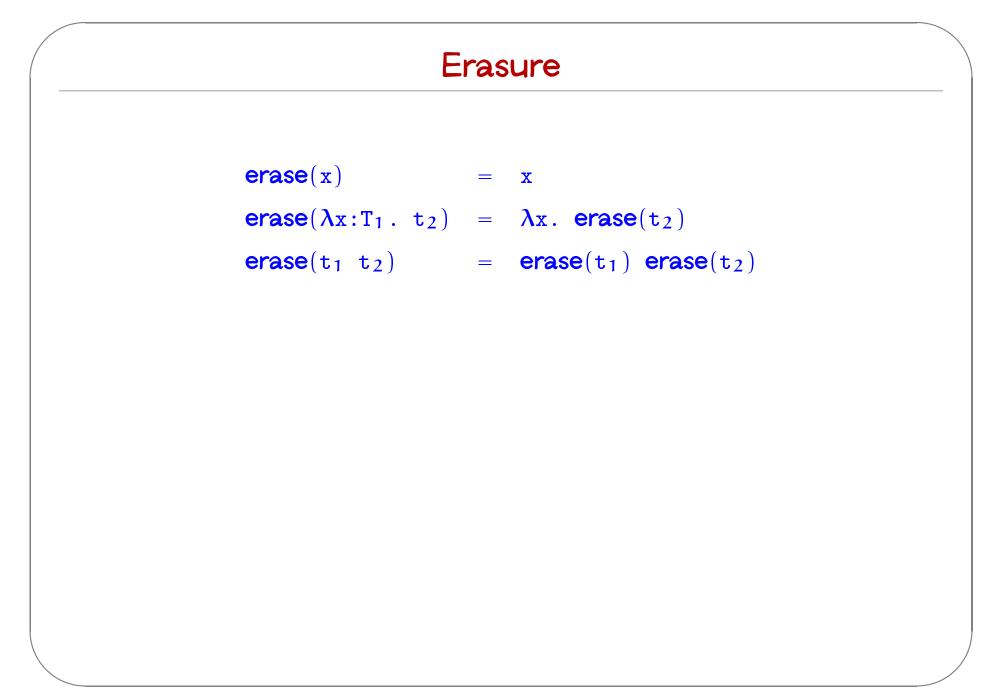
Logic	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type $P \times Q$
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)

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) evaluation

Propositions as Types

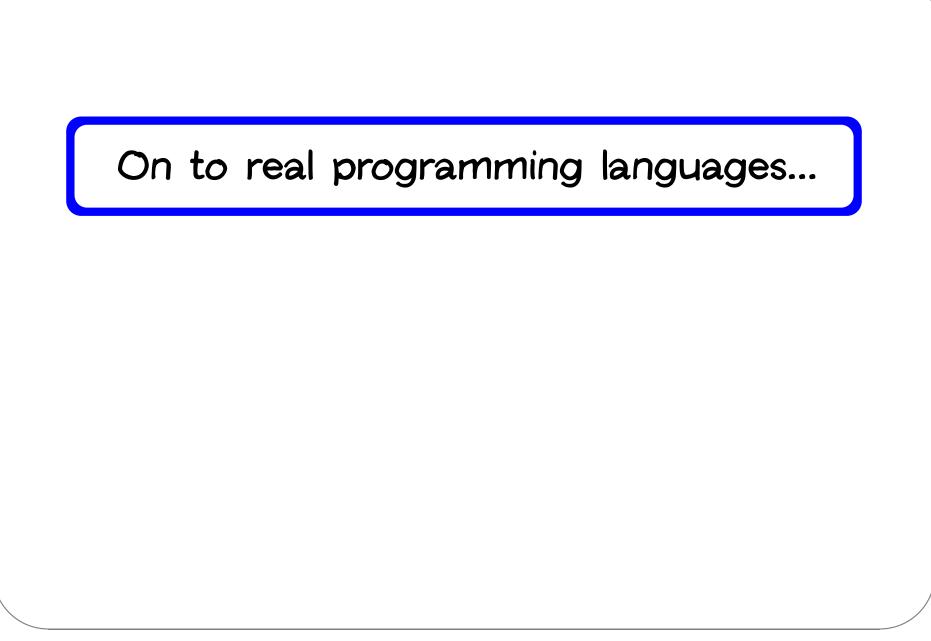
Logic	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type P→Q
proposition $\mathbf{P} \wedge \mathbf{Q}$	type $P \times Q$
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)
proof simplification	evaluation
(cut elimination)	



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

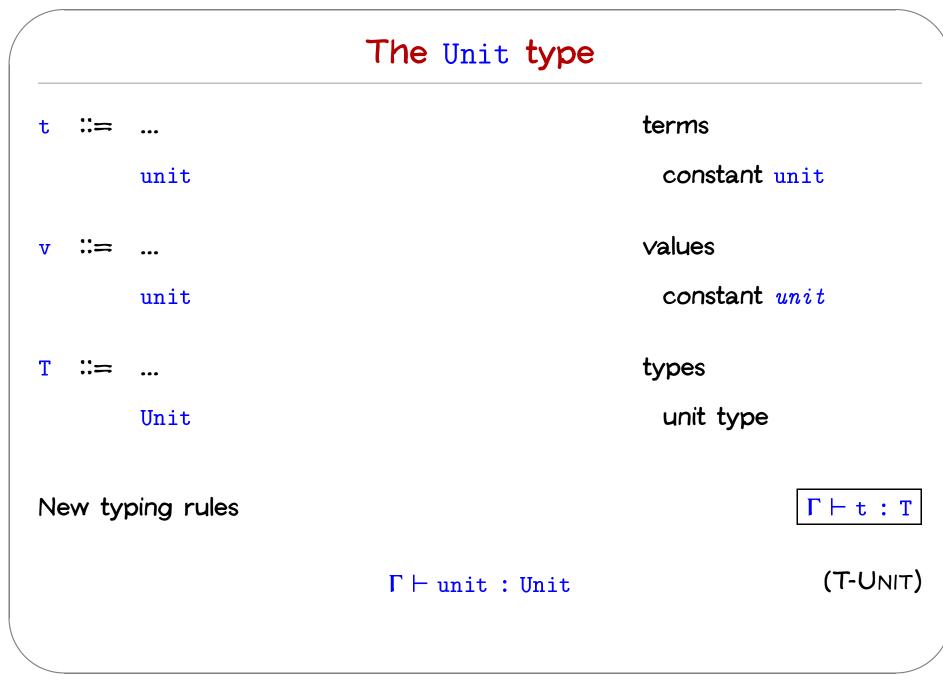
Up to now, we've formulated "base types" (e.g. Nat) by adding them to the syntax of types, extending the syntax of terms with associated constants (zero) and operators (succ, etc.) and adding appropriate typing and evaluation rules. We can do this for as many base types as we like.

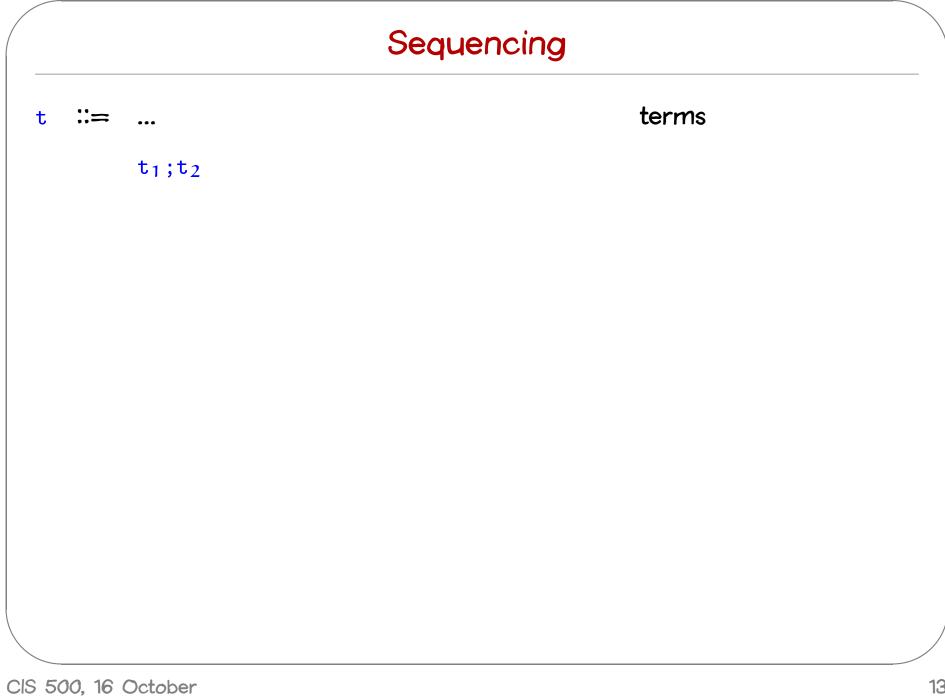
For more theoretical discussions (as opposed to programming) we can often ignore the term-level inhabitants of base types, and just treat these types as uninterpreted constants.

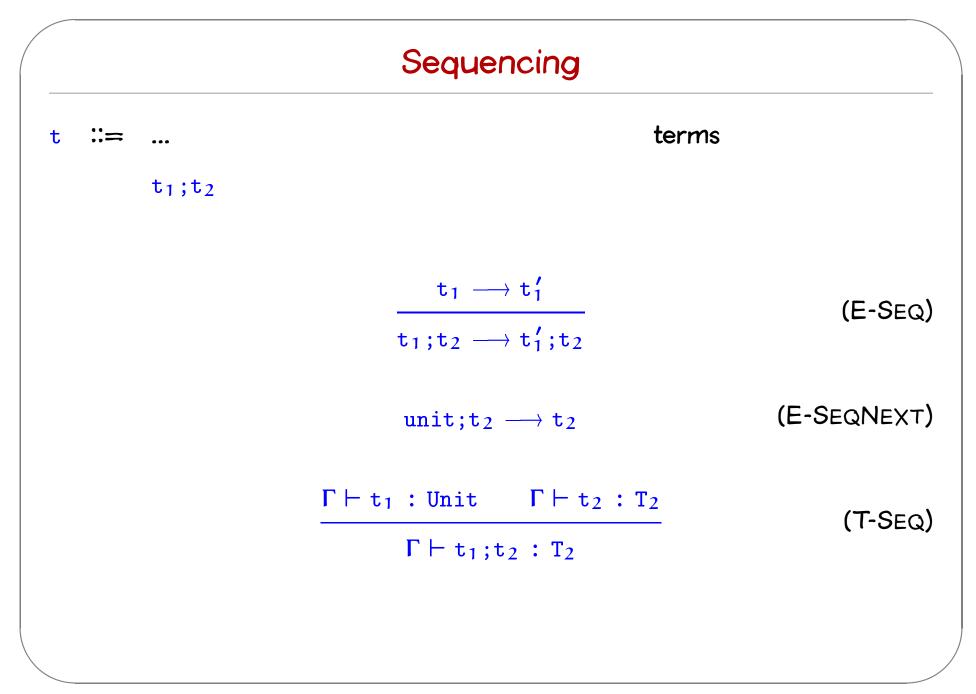
E.g., suppose B and C are some base types. Then we can ask (without knowing anything more about B or C) whether there are any types S and T such that the term

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

is well typed.



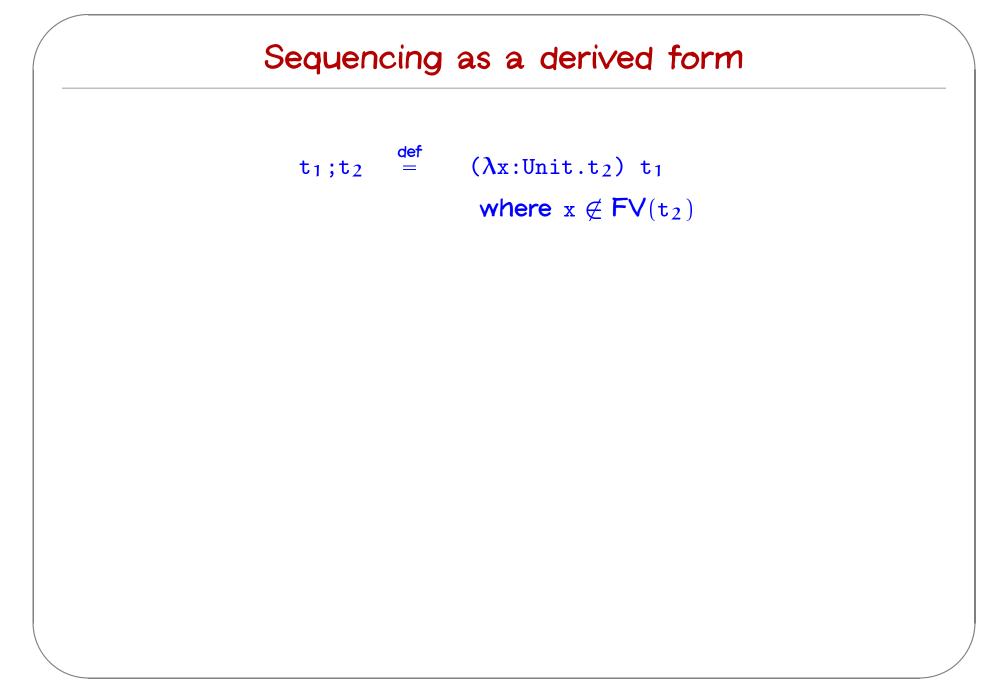




Derived forms

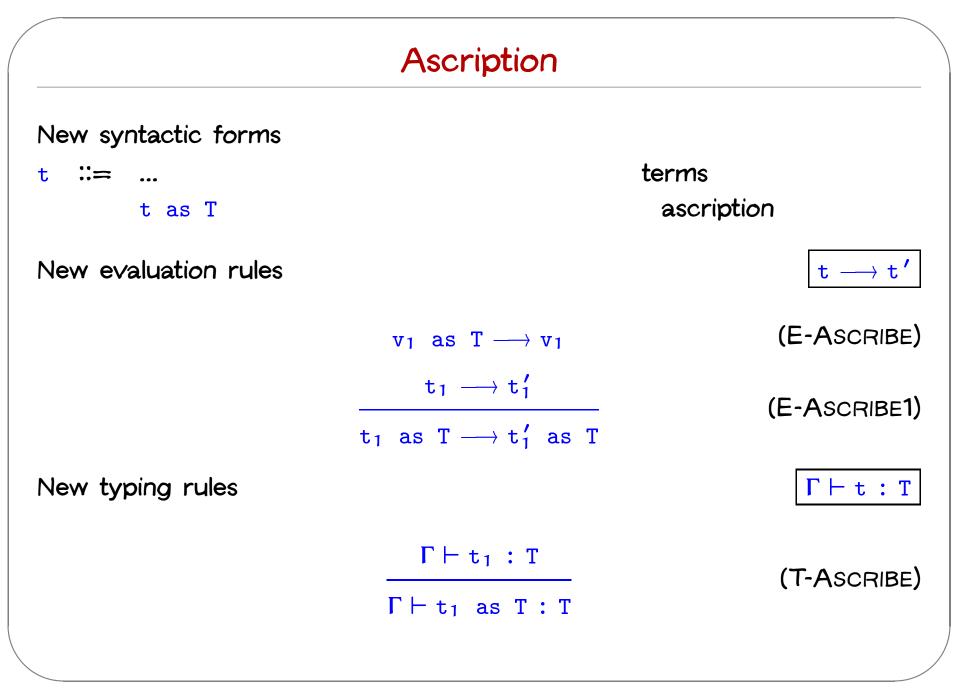
• Syntatic sugar

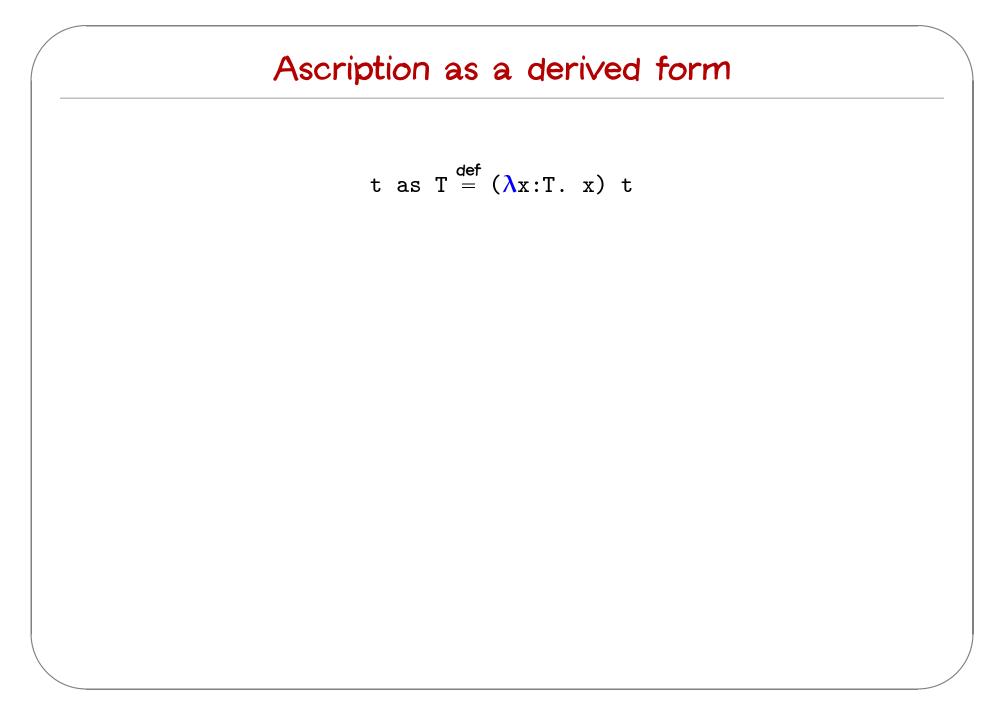
♦ Internal language vs. external (surface) language

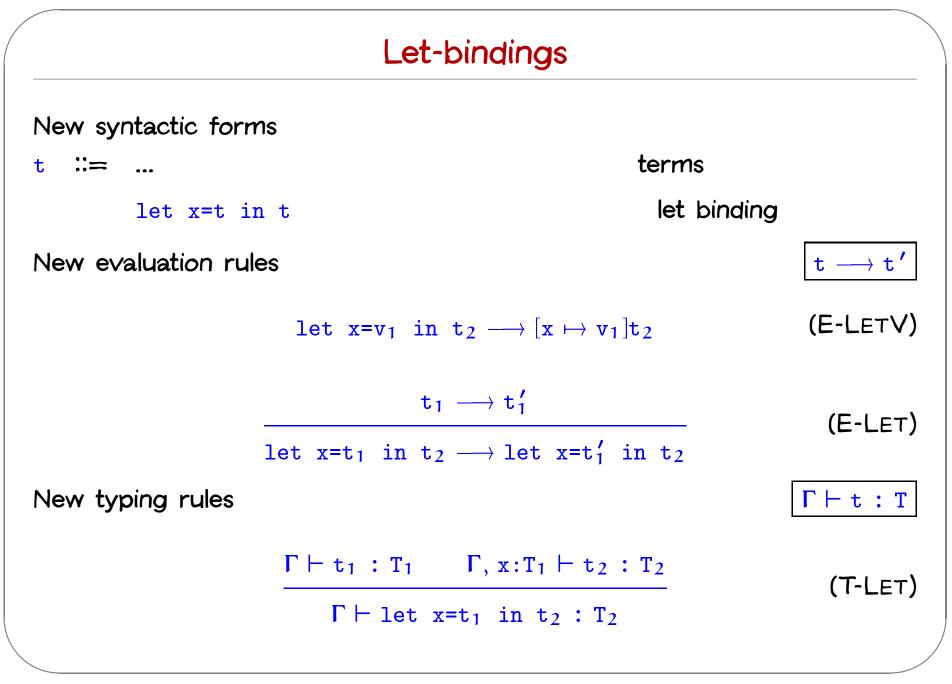


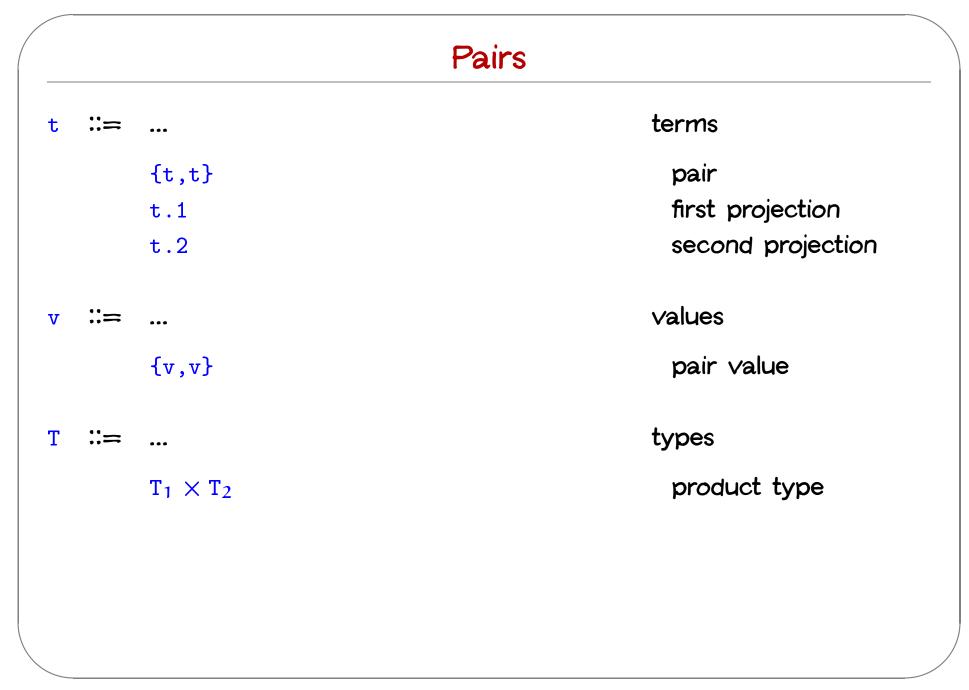
Equivalence of the two definitions

[board]









Evaluation rules for pairs

$ \{v_1, v_2\}.1 \longrightarrow v_1 \\ \{v_1, v_2\}.2 \longrightarrow v_2 $	(E-PAIRBETA1) (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-Proj2)
$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}'_1}{\{\mathtt{t}_1, \mathtt{t}_2\} \longrightarrow \{\mathtt{t}'_1, \mathtt{t}_2\}}$	(E-PAIR1)
$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\{\mathtt{v}_1, \mathtt{t}_2\} \longrightarrow \{\mathtt{v}_1, \mathtt{t}_2'\}}$	(E-PAIR2)

