

Chapter 11: Simply Extensions

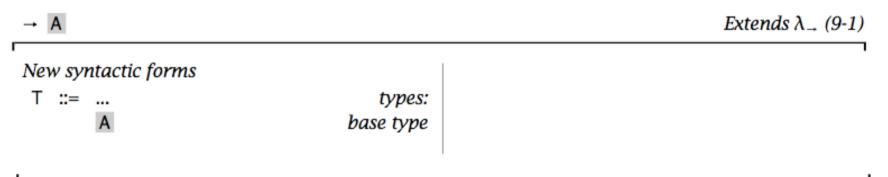
Basic Types / The Unit Type Derived Forms: Sequencing and Wildcard Ascription / Let Binding Pairs/Tuples/Records Sums/Variants General Recursion / Lists



Base Types



- Base types in every programming language:
 - sets of simple, unstructured values such as numbers, booleans, or characters, and
 - primitive operations for manipulating these values.
- Theoretically, we may consider our language is equipped with some uninterpreted base types.





A, B, C, ...



 λ x:A. x; <fun>: A \rightarrow A

 λ x:B. x; <fun>: B \rightarrow B

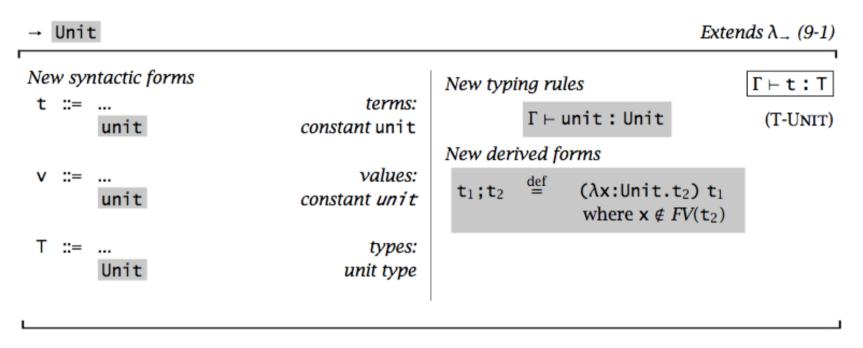
 λ f:A \rightarrow A. λ x:A. f(f(x)); <fun>: (A \rightarrow A) \rightarrow A \rightarrow A



The Unit Type



• It is the singleton type (like void in C).



Application: Unit-type expressions care more about "side effects" rather than "results".



Derived Form: Sequencing t₁; t₂



• A direct extension (λ^{E})

- New valuation relation rules

$$\frac{t_1 \rightarrow t'_1}{t_1; t_2 \rightarrow t'_1; t_2}$$
(E-SEQ)
unit; $t_2 \rightarrow t_2$ (E-SEQNEXT)

- New typing rules

$$\frac{\Gamma \vdash \mathsf{t}_1 : \mathsf{Unit} \quad \Gamma \vdash \mathsf{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{t}_1; \mathsf{t}_2 : \mathsf{T}_2}$$



Derived Form: Sequencing t_1 ; t_2



• Derived form (λ ^I): syntactic sugar

$$t_1; t_2 \stackrel{\text{def}}{=} (\lambda x: \text{Unit.} t_2) t_1$$

where $x \notin FV(t_2)$

• Theorem [Sequencing is a derived form]: Let $e \in \lambda^{E} \rightarrow \lambda^{I}$

be the elaboration function (desugaring) that translates from the external to the internal language by replacing every occurrence of t1;t2 with (λ x:Unit.t2) t1. Then

- $t \longrightarrow_E t'$ iff $e(t) \longrightarrow_I e(t')$
- $\Gamma \vdash^{E} t : T \text{ iff } \Gamma \vdash^{I} e(t) : T$



Derived Form: Wildcard



• A derived form



where x is some variable not occurring in t.



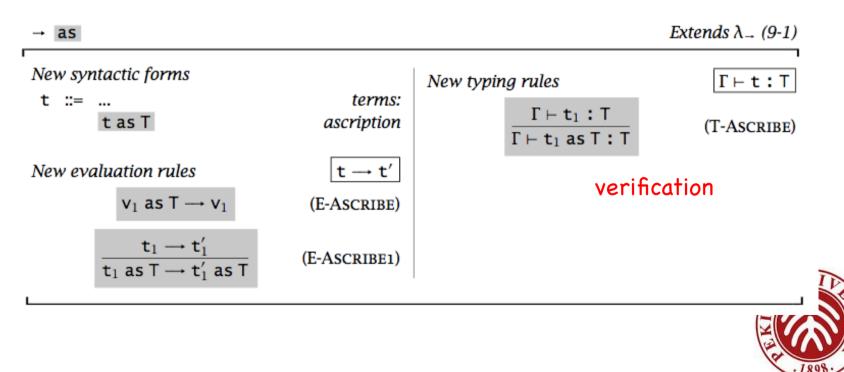
Ascription: t as T



• t as T

meaning for the term t, we ascribe the type T

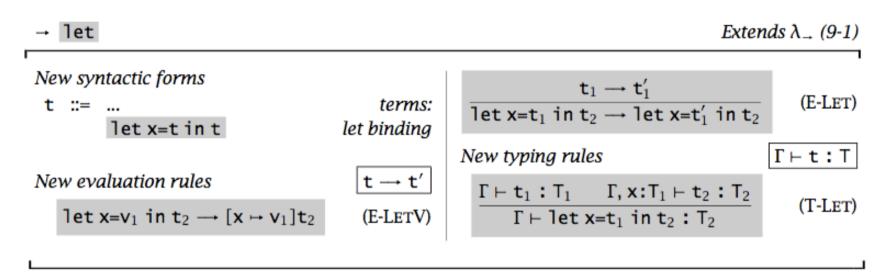
- Useful for documentation and pinpointing error sources
- Useful for controlling type printing
- Useful for specializing types



Let Bindings

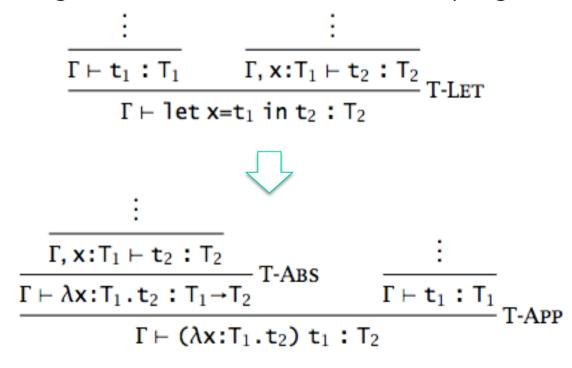


• To give names to some of its subexpressions.





- Is "let binding" a derived form? let $x=t_1$ in $t_2 \rightarrow (\lambda x:T_1.t_2) t_1$
- Desugaring is not on terms but on typing derivations





Pairs



→ × Extends λ_{-} (9-1) New syntactic forms $\frac{t_1 \rightarrow t_1'}{t_1.2 \rightarrow t_1'.2}$ (E-Proj2) t ::= ... terms: {t,t} pair t.1 first projection $\frac{\mathtt{t}_1 \rightarrow \mathtt{t}_1'}{\{\mathtt{t}_1, \mathtt{t}_2\} \rightarrow \{\mathtt{t}_1', \mathtt{t}_2\}}$ (E-PAIR1) t.2 second projection values: V ::= $\frac{\mathtt{t}_2 \rightarrow \mathtt{t}_2'}{\{\mathtt{v}_1, \mathtt{t}_2\} \rightarrow \{\mathtt{v}_1, \mathtt{t}_2'\}}$... (E-PAIR2) $\{v,v\}$ pair value Γ⊢t:T Т ::= ... New typing rules types: $T_1 \times T_2$ product type $\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2$ (T-PAIR) $\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2$ New evaluation rules $\mathsf{t} \to \mathsf{t}'$ $\Gamma \vdash t_1 : T_{11} \times T_{12}$ $\{\mathbf{v}_1,\mathbf{v}_2\}.1 \rightarrow \mathbf{v}_1$ (E-PAIRBETA1) (T-Proj1) $\Gamma \vdash t_1 . 1 : T_{11}$ $\{\mathbf{v}_1, \mathbf{v}_2\}.2 \longrightarrow \mathbf{v}_2$ (E-PAIRBETA2) $\Gamma \vdash t_1 : T_{11} \times T_{12}$ (T-PROJ2) $\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.\mathtt{l} \longrightarrow \mathtt{t}_1'.\mathtt{l}}$ $\Gamma \vdash t_1.2:T_{12}$ (E-Proj1)

• To build compound data structures.



Tuples



Generalization: binary \rightarrow n-ary products

→ {}		E	xtends λ_{\rightarrow} (9-1)
New syntactic forms t ::=	terms:	$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1 . \mathtt{i} \longrightarrow \mathtt{t}_1' . \mathtt{i}}$	(E-Proj)
$\{t_i^{i\in l.n}\}$ t.i	tuple projection		(E Tupur)
$\mathbf{v} ::= \dots$ { $\mathbf{v}_i \in I \dots n$ }	values: tuple value	$\frac{t_{j} \longrightarrow t'_{j}}{\{v_{i} \stackrel{i \in Ij-1}{,} t_{j}, t_{k} \stackrel{k \in j+In}{,} t_{j} \\ \longrightarrow \{v_{i} \stackrel{i \in Ij-l}{,} t'_{j}, t_{k} \stackrel{k \in j+In}{,} t_{j} \}}$	(E-Tuple)
		New typing rules	$\Gamma \vdash t:T$
$T ::= \dots \\ \{T_i^{i \in 1n}\}$	types: tuple type	$\frac{\text{for each } i \Gamma \vdash t_i : T_i}{\Gamma \vdash \{t_i \ ^{i \in In}\} : \{T_i \ ^{i \in In}\}}$	(T-Tuple)
New evaluation rules $\{v_i^{i \in ln}\} : j \longrightarrow v_j$	$t \rightarrow t'$ (E-ProjTuple)	$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{T}_i \ ^{i \in In}\}}{\Gamma \vdash \mathtt{t}_1 . \mathtt{j} : \mathtt{T}_j}$	(T-Proj)
		1	



Records



Generalization: n-ary products \rightarrow labeled records

→ {}		Exte	nds λ_{\rightarrow} (9-1)
New syntactic forms	terms:	$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.1 \longrightarrow \mathtt{t}_1'.1}$	(E-Proj)
$\{l_i=t_i \stackrel{i\in In}{=} $	record projection	$t_j \longrightarrow t_j'$	– (E-RCD)
$ V ::= \dots \\ \{I_i = v_i \ i \in I \dots n\} $	values: record value	$ \{ \exists_i = v_i^{i \in Ij-1}, \exists_j = t_j, \exists_k = t_k^{k \in j+I.n} \} \\ \longrightarrow \{ \exists_i = v_i^{i \in Ij-1}, \exists_j = t'_j, \exists_k = t_k^{k \in j+I.n} \} $	
Τ	t	New typing rules	$\Gamma \vdash t : T$
$T ::= \dots \\ \{I_i : T_i \in \mathbb{I} \ n\}$	types: type of records	$\frac{\text{for each } i \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i \ ^{i \in In}\} : \{l_i : T_i \ ^{i \in In}\}}$	(T-RCD)
<i>New evaluation rules</i> $\{ l_i = v_i^{i \in ln} \} . l_j \longrightarrow v_j$	$t \rightarrow t'$ (E-PROJRCD)	$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{l}_i : \mathtt{T}_i \stackrel{i \in In}{}\}}{\Gamma \vdash \mathtt{t}_1 . \mathtt{l}_j : \mathtt{T}_j}$	(T-Proj)
		1	

Question: {partno=5524, cost=30.27} = {cost=30.27,partno=5524}?



Sums



- To deal with heterogeneous collections of values.
- An Example: Address books

```
PhysicalAddr = {firstlast:String, addr:String};
VirtualAddr = {name:String, email:String};
```

Addr = PhysicalAddr + VirtualAddr;

- Injection by tagging (disjoint unions)
 - in] : PhysicalAddr \rightarrow PhysicalAddr+VirtualAddr
 - inr : VirtualAddr → PhysicalAddr+VirtualAddr
- Processing by case analysis

```
getName = λa:Addr.
case a of
inl x ⇒ x.firstlast
| inr y ⇒ y.name;
```



Sums



• To deal with heterogeneous collections of values.

lew syntactic forms	terms:	$t_0 \rightarrow t'_0$	
inlt tagging (le		case t_0 of inl $x_1 \Rightarrow t_1 $ in \rightarrow case t'_0 of inl $x_1 \Rightarrow t_1 $ i	
casetofinlx⇒t	tagging (right) inr x⇒t case		(E-CASE
v ::= inl v ta	values:	$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\texttt{inl} \mathtt{t}_1 \longrightarrow \texttt{inl} \mathtt{t}_1'}$	(E-Inl
	tagged value (left) tagged value (right)	$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{inr} \mathtt{t}_1 \longrightarrow \mathtt{inr} \mathtt{t}_1'}$	(E-INR
T ::= T+T	types: sum type	New typing rules	Γ⊢t:T
lew evaluation rules	$t \to t'$	$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash inl t_1 : T_1 + T_2}$	(T-INL
case (inl v_0) of inl $x_1 \Rightarrow t_1 \mid inr x_2 \Rightarrow t_2$ $\rightarrow [x_1 \mapsto v_0]t_1$	2 (E-CASEINL)	$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_2}{\Gamma \vdash \mathtt{inr} \mathtt{t}_1 : \mathtt{T}_1 + \mathtt{T}_2}$	(T-INR
case (inr v_0) of inl $x_1 \Rightarrow t_1 \mid inr x_2 \Rightarrow t_2$ $\rightarrow [x_2 \rightarrow v_0]t_2$	2 (E-CASEINR)	$\begin{split} \Gamma &\vdash \mathbf{t}_0 : T_1 {+} T_2 \\ \Gamma, \mathbf{x}_1 {:} T_1 &\vdash \mathbf{t}_1 : T \Gamma, \mathbf{x}_2 {:} \\ \Gamma &\vdash case \ \mathbf{t}_0 \ ofinl \ \mathbf{x}_1 {\Rightarrow} t_1 \mid \end{split}$	



Sums (with Unique Typing)



→ +	Extends λ_{\rightarrow} (11-9)
New syntactic forms t ::= terms:	case (inr v_0 as T_0)
inlt as T tagging (left)	of inl $x_1 \Rightarrow t_1 \mid inr x_2 \Rightarrow t_2$ (E-CASEINR) $\rightarrow [x_2 \mapsto v_0]t_2$
inrt as T tagging (right) v ::= values:	$ \begin{array}{c} t_1 \to t_1' \\ \hline inl \ t_1 \ as \ T_2 \ \to \ inl \ t_1' \ as \ T_2 \end{array} \tag{E-INL} $
inl v as T tagged value (left) inr v as T tagged value (right)	$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1'}{\texttt{inr t}_1 \texttt{ as } \texttt{T}_2 \rightarrow \texttt{inr t}_1' \texttt{ as } \texttt{T}_2} \tag{E-INR}$
New evaluation rules $t \rightarrow t'$	<i>New typing rules</i> $\Gamma \vdash t:T$
case (inl v_0 as T_0) of inl $x_1 \Rightarrow t_1 \mid inr x_2 \Rightarrow t_2$ (E-CASEINL)	$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{inl } t_1 \text{ as } \underline{T_1 + T_2} : T_1 + T_2} $ (T-INL)
$\rightarrow [\mathbf{x}_1 \mapsto \mathbf{v}_0]\mathbf{t}_1$	$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash inr t_1 \text{ as } \underline{T_1 + T_2} : T_1 + T_2} $ (T-INR)



Variant

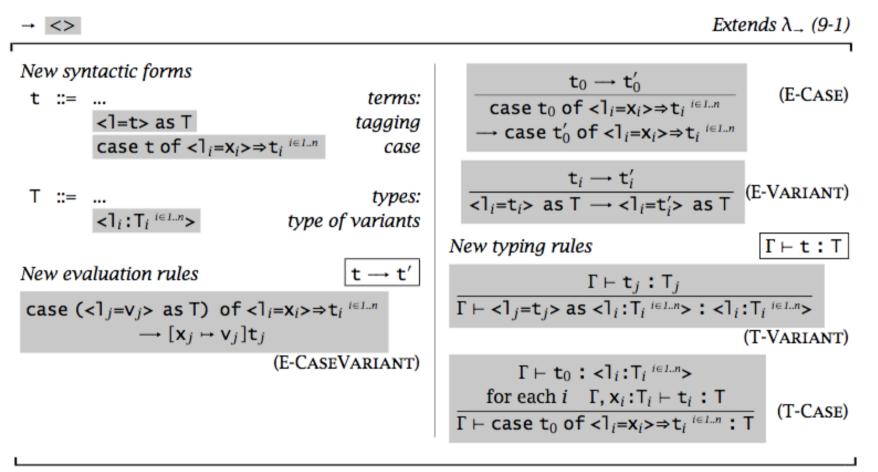


- Generalization: Sums \rightarrow Labeled variants
 - T1 + T2 → <l1:T1, l2:Te>
 - inl t as T1+T2 → <l1=t> as <l1:T1, l2:Te>
- Example:

```
Addr = <physical:PhysicalAddr, virtual:VirtualAddr>;
a = <physical=pa> as Addr;
a : Addr
getName = λa:Addr.
case a of
  <physical=x> ⇒ x.firstlast
| <virtual=y> ⇒ y.name;
> getName : Addr → String
```









Special Instances of Variants



• Options

OptionalNat = <none:Unit, some:Nat>;

• Enumerations

Weekday = <monday:Unit, tuesday:Unit, wednesday:Unit, thursday:Unit, friday:Unit>;

• Single-Field Variants

 $V = \langle l:T \rangle$

Operations on T cannot be applied to elements of V without first unpackaging them: a V cannot be accidentally mistaken for a T.

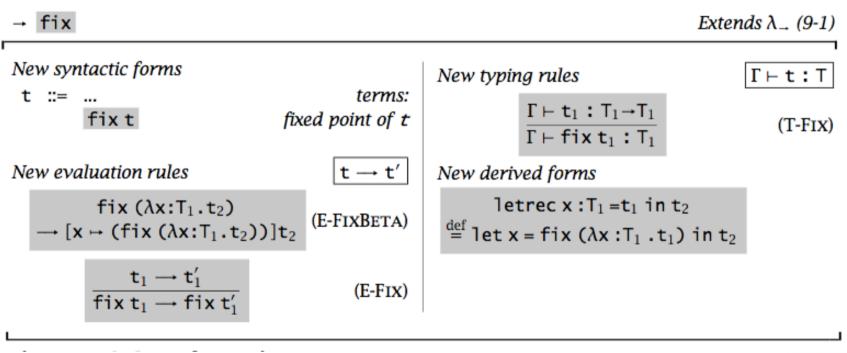


General Recursions



• Introduce "fix" operator: fix f = f (fix f)

(It cannot be defined as a derived form in simply typed lambda calculus)







• Example 1:

▶ false : Bool



• Example 2:

```
ff = \lambdaieio:{iseven:Nat\rightarrowBool, isodd:Nat\rightarrowBool}.
          {iseven = \lambda x:Nat.
                       if iszero x then true
                       else ieio.isodd (pred x),
           isodd = \lambda x: Nat.
                       if iszero x then false
                       else ieio.iseven (pred x)};
▶ ff : {iseven:Nat→Bool,isodd:Nat→Bool} →
       {iseven:Nat→Bool, isodd:Nat→Bool}
   r = fix ff;
r : {iseven:Nat→Bool, isodd:Nat→Bool}
   iseven = r.iseven;
iseven : Nat → Bool
```

iseven 7;

▶ false : Bool





• Example 3: Given any type T, can you define a term that has type T?

x as T

fix (λ x:T. x)

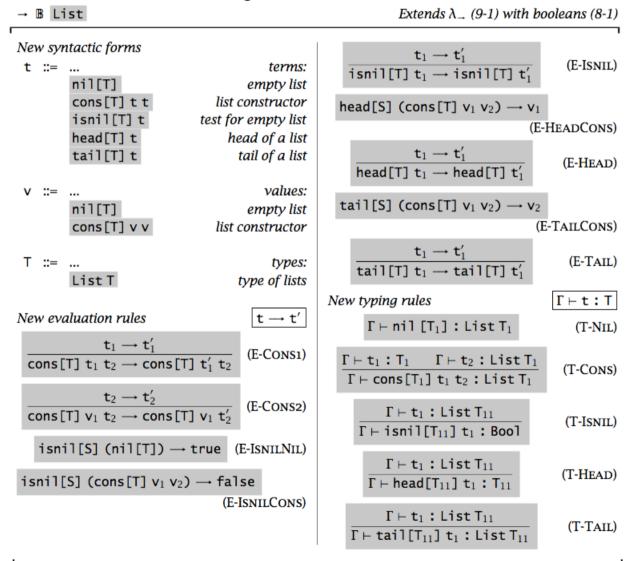
diverge_T = $\lambda_{-}:$ Unit. fix ($\lambda x:T.x$);

• diverge_T : Unit \rightarrow T





• List T describes finite-length lists whose elements are drawn from T.





Lists

Homework



- Read Chapter 11.
- Do Exercise 11.11.2.

