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.

$\to$ $\boxed{A}$ *Extends* $\lambda_{\to}$ *(9-1)*

New syntactic forms

T	::=	...	*types:*
		$\boxed{A}$	*base type*

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).

→ Unit *Extends* $\lambda_\to$ *(9-1)*

New syntactic forms

t	::=	...	*terms:*
		unit	*constant* unit
v	::=	...	*values:*
		unit	*constant* unit
T	::=	...	*types:*
		Unit	*unit type*

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\Gamma \vdash \texttt{unit} : \texttt{Unit} \qquad \text{(T-UNIT)}$$

New derived forms

$$t_1;t_2 \overset{\text{def}}{=} (\lambda x{:}\texttt{Unit}.t_2)\ t_1 \quad \text{where } x \notin FV(t_2)$$

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

Derived Form: Sequencing t_1 ; t_2

- A direct extension (λ^{E})
  - t ::= …

    t1 ; t2

  - New valuation relation rules

$$\frac{t_1 \longrightarrow t_1'}{t_1;t_2 \longrightarrow t_1';t_2} \qquad \text{(E-Seq)}$$

$$\texttt{unit};t_2 \longrightarrow t_2 \qquad \text{(E-SeqNext)}$$

  - New typing rules

$$\frac{\Gamma \vdash t_1 : \texttt{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1;t_2 : T_2}$$

Derived Form: Sequencing t_1 ; t_2

- Derived form (λ^I): syntactic sugar

$$t_1;t_2 \stackrel{\text{def}}{=} (\lambda x{:}\mathrm{Unit}.t_2)\ t_1 \quad \text{where } x \notin FV(t_2)$$

- **Theorem** [Sequencing is a derived form]: Let

  $e \in \lambda^E \to \lambda^I$

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

Derived Form: Wildcard

- A derived form

$$\lambda\_:S.t \Rightarrow \lambda x:S.t$$

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

$\rightarrow$ as *Extends $\lambda_{\rightarrow}$ (9-1)*

New syntactic forms

$\mathtt{t} ::= \ldots$ *terms:*

$\quad \mathtt{t\ as\ T}$ *ascription*

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t'}}$

$$\mathtt{v_1\ as\ T} \longrightarrow \mathtt{v_1} \qquad \text{(E-ASCRIBE)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ as\ T} \longrightarrow \mathtt{t_1'\ as\ T}} \qquad \text{(E-ASCRIBE1)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T}}{\Gamma \vdash \mathtt{t_1\ as\ T} : \mathtt{T}} \qquad \text{(T-ASCRIBE)}$$

verification

Let Bindings

- To give names to some of its subexpressions.

$\rightarrow$ `let` *Extends* $\lambda_{\rightarrow}$ *(9-1)*

New syntactic forms

$\mathtt{t} ::= \ldots$		*terms:*
	$\mathtt{let}\ \mathtt{x}=\mathtt{t}\ \mathtt{in}\ \mathtt{t}$	*let binding*

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\mathtt{let}\ \mathtt{x}=\mathtt{v}_1\ \mathtt{in}\ \mathtt{t}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_1]\mathtt{t}_2 \qquad \text{(E-LetV)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{let}\ \mathtt{x}=\mathtt{t}_1\ \mathtt{in}\ \mathtt{t}_2 \longrightarrow \mathtt{let}\ \mathtt{x}=\mathtt{t}_1'\ \mathtt{in}\ \mathtt{t}_2} \qquad \text{(E-Let)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{let}\ \mathtt{x}=\mathtt{t}_1\ \mathtt{in}\ \mathtt{t}_2 : \mathtt{T}_2} \qquad \text{(T-Let)}$$

- Is "let binding" a derived form?

  let $x{=}t_1$ in t_2 ➔ $(\lambda x{:}T_1.t_2)\ t_1$

- Desugaring is not on terms but on typing derivations

$$
\dfrac{\dfrac{\vdots}{\Gamma \vdash t_1 : T_1} \qquad \dfrac{\vdots}{\Gamma, x{:}T_1 \vdash t_2 : T_2}}{\Gamma \vdash \texttt{let } x{=}t_1 \texttt{ in } t_2 : T_2}\ \text{T-LET}
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x{:}T_1 \vdash t_2 : T_2}}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2}\ \text{T-ABS} \qquad \dfrac{\vdots}{\Gamma \vdash t_1 : T_1}}{\Gamma \vdash (\lambda x{:}T_1.t_2)\ t_1 : T_2}\ \text{T-APP}
$$

Pairs

- To build compound data structures.

$\rightarrow \times$ *Extends $\lambda_{\rightarrow}$ (9-1)*

New syntactic forms

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

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\{v_1, v_2\}.1 \longrightarrow v_1 \qquad \text{(E-PairBeta1)}$$

$$\{v_1, v_2\}.2 \longrightarrow v_2 \qquad \text{(E-PairBeta2)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1} \qquad \text{(E-Proj1)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.2 \longrightarrow t_1'.2} \qquad \text{(E-Proj2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}} \qquad \text{(E-Pair1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}} \qquad \text{(E-Pair2)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}} \qquad \text{(T-Proj2)}$$

Tuples

Generalization: binary ➔ n-ary products

$\rightarrow$ {} *Extends $\lambda_{\rightarrow}$ (9-1)*

New syntactic forms

t	::=	...	*terms:*
		$\{t_i{}^{i\in 1..n}\}$	*tuple*
		$t.i$	*projection*
v	::=	...	*values:*
		$\{v_i{}^{i\in 1..n}\}$	*tuple value*
T	::=	...	*types:*
		$\{T_i{}^{i\in 1..n}\}$	*tuple type*

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\{v_i{}^{i\in 1..n}\}.j \longrightarrow v_j \quad \text{(E-ProjTuple)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.i \longrightarrow t_1'.i} \quad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\{v_i{}^{i\in 1..j-1}, t_j, t_k{}^{k\in j+1..n}\} \longrightarrow \{v_i{}^{i\in 1..j-1}, t_j', t_k{}^{k\in j+1..n}\}} \quad \text{(E-Tuple)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{t_i{}^{i\in 1..n}\} : \{T_i{}^{i\in 1..n}\}} \quad \text{(T-Tuple)}$$

$$\frac{\Gamma \vdash t_1 : \{T_i{}^{i\in 1..n}\}}{\Gamma \vdash t_1.j : T_j} \quad \text{(T-Proj)}$$

Records

Generalization: n-ary products ➔ labeled records

→ {} *Extends* $\lambda_\to$ *(9-1)*

New syntactic forms

t ::=	...		*terms:*
	$\{l_i=t_i\ ^{i\in 1..n}\}$		*record*
	$t.l$		*projection*
v ::=	...		*values:*
	$\{l_i=v_i\ ^{i\in 1..n}\}$		*record value*
T ::=	...		*types:*
	$\{l_i:T_i\ ^{i\in 1..n}\}$		*type of records*

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\{l_i=v_i\ ^{i\in 1..n}\}.l_j \longrightarrow v_j \qquad \text{(E-ProjRcd)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.l \longrightarrow t_1'.l} \qquad \text{(E-Proj)}$$

$$\frac{t_j \longrightarrow t_j'}{\begin{array}{l}\{l_i=v_i\ ^{i\in 1..j-1}, l_j=t_j, l_k=t_k\ ^{k\in j+1..n}\}\\ \longrightarrow \{l_i=v_i\ ^{i\in 1..j-1}, l_j=t_j', l_k=t_k\ ^{k\in j+1..n}\}\end{array}} \qquad \text{(E-Rcd)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i=t_i\ ^{i\in 1..n}\} : \{l_i:T_i\ ^{i\in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma \vdash t_1 : \{l_i:T_i\ ^{i\in 1..n}\}}{\Gamma \vdash t_1.l_j : T_j} \qquad \text{(T-Proj)}$$

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)

```
inl  :  PhysicalAddr → 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.

$\to$ $+$ *Extends $\lambda_\to$ (9-1)*

New syntactic forms

t	::=	...	*terms:*
		$\texttt{inl}\ t$	*tagging (left)*
		$\texttt{inr}\ t$	*tagging (right)*
		$\texttt{case}\ t\ \texttt{of}\ \texttt{inl}\ x \Rightarrow t \mid \texttt{inr}\ x \Rightarrow t$	*case*
v	::=	...	*values:*
		$\texttt{inl}\ v$	*tagged value (left)*
		$\texttt{inr}\ v$	*tagged value (right)*
T	::=	...	*types:*
		$T+T$	*sum type*

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\texttt{case}\ (\texttt{inl}\ v_0)\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_0]t_1 \quad \text{(E-CaseInl)}$$

$$\texttt{case}\ (\texttt{inr}\ v_0)\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_0]t_2 \quad \text{(E-CaseInr)}$$

$$\frac{t_0 \longrightarrow t_0'}{\texttt{case}\ t_0\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \longrightarrow \texttt{case}\ t_0'\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2} \quad \text{(E-Case)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inl}\ t_1 \longrightarrow \texttt{inl}\ t_1'} \quad \text{(E-Inl)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inr}\ t_1 \longrightarrow \texttt{inr}\ t_1'} \quad \text{(E-Inr)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \texttt{inl}\ t_1 : T_1 + \boxed{T_2}} \quad \text{(T-Inl)}$$

$$\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \texttt{inr}\ t_1 : T_1 + \boxed{T_2}} \quad \text{(T-Inr)}$$

$$\frac{\Gamma \vdash t_0 : T_1+T_2 \qquad \Gamma, x_1{:}T_1 \vdash t_1 : T \qquad \Gamma, x_2{:}T_2 \vdash t_2 : T}{\Gamma \vdash \texttt{case}\ t_0\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 : T} \quad \text{(T-Case)}$$

Sums (with Unique Typing)

$\rightarrow +$ *Extends $\lambda_{\rightarrow}$ (11-9)*

New syntactic forms

$$
\begin{array}{lllr}
\texttt{t} & ::= & \ldots & \textit{terms:} \\
 & & \texttt{inl t } \colorbox{lightgray}{\texttt{as T}} & \textit{tagging (left)} \\
 & & \texttt{inr t } \colorbox{lightgray}{\texttt{as T}} & \textit{tagging (right)} \\
 & & & \\
\texttt{v} & ::= & \ldots & \textit{values:} \\
 & & \texttt{inl v } \colorbox{lightgray}{\texttt{as T}} & \textit{tagged value (left)} \\
 & & \texttt{inr v } \colorbox{lightgray}{\texttt{as T}} & \textit{tagged value (right)}
\end{array}
$$

New evaluation rules $\boxed{\texttt{t} \longrightarrow \texttt{t}'}$

$$
\begin{array}{l}
\texttt{case (inl } \texttt{v}_0 \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_0$} \texttt{)} \\
\texttt{of inl } \texttt{x}_1 \Rightarrow \texttt{t}_1 \mid \texttt{inr } \texttt{x}_2 \Rightarrow \texttt{t}_2 \\
\qquad \longrightarrow [\texttt{x}_1 \mapsto \texttt{v}_0]\texttt{t}_1
\end{array}
\quad \text{(E-CASEINL)}
$$

$$
\begin{array}{l}
\texttt{case (inr } \texttt{v}_0 \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_0$} \texttt{)} \\
\texttt{of inl } \texttt{x}_1 \Rightarrow \texttt{t}_1 \mid \texttt{inr } \texttt{x}_2 \Rightarrow \texttt{t}_2 \\
\qquad \longrightarrow [\texttt{x}_2 \mapsto \texttt{v}_0]\texttt{t}_2
\end{array}
\quad \text{(E-CASEINR)}
$$

$$
\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inl } \texttt{t}_1 \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_2$} \longrightarrow \texttt{inl } \texttt{t}_1' \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_2$}} \quad \text{(E-INL)}
$$

$$
\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{inr } \texttt{t}_1 \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_2$} \longrightarrow \texttt{inr } \texttt{t}_1' \text{ } \colorbox{lightgray}{\texttt{as } $\texttt{T}_2$}} \quad \text{(E-INR)}
$$

New typing rules $\boxed{\Gamma \vdash \texttt{t} : \texttt{T}}$

$$
\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1}{\Gamma \vdash \texttt{inl } \texttt{t}_1 \text{ } \colorbox{lightgray}{\texttt{as } $\underline{\texttt{T}_1+\texttt{T}_2}$} : \texttt{T}_1+\texttt{T}_2} \quad \text{(T-INL)}
$$

$$
\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_2}{\Gamma \vdash \texttt{inr } \texttt{t}_1 \text{ } \colorbox{lightgray}{\texttt{as } $\underline{\texttt{T}_1+\texttt{T}_2}$} : \texttt{T}_1+\texttt{T}_2} \quad \text{(T-INR)}
$$

Variant

- Generalization: Sums ➔ 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
```

$\rightarrow$ $\langle\rangle$ *Extends $\lambda_{\rightarrow}$ (9-1)*

New syntactic forms

$$
\begin{array}{lll}
t ::= & \ldots & \textit{terms:} \\
 & \langle l{=}t\rangle \text{ as } T & \textit{tagging} \\
 & \text{case } t \text{ of } \langle l_i{=}x_i\rangle \Rightarrow t_i \;{}^{i\in 1..n} & \textit{case}
\end{array}
$$

$$
\begin{array}{lll}
T ::= & \ldots & \textit{types:} \\
 & \langle l_i{:}T_i \;{}^{i\in 1..n}\rangle & \textit{type of variants}
\end{array}
$$

New evaluation rules $\boxed{t \longrightarrow t'}$

$$\text{case } (\langle l_j{=}v_j\rangle \text{ as } T) \text{ of } \langle l_i{=}x_i\rangle \Rightarrow t_i \;{}^{i\in 1..n} \longrightarrow [x_j \mapsto v_j]t_j \qquad \text{(E-CaseVariant)}$$

$$\frac{t_0 \longrightarrow t_0'}{\text{case } t_0 \text{ of } \langle l_i{=}x_i\rangle \Rightarrow t_i \;{}^{i\in 1..n} \longrightarrow \text{case } t_0' \text{ of } \langle l_i{=}x_i\rangle \Rightarrow t_i \;{}^{i\in 1..n}} \qquad \text{(E-Case)}$$

$$\frac{t_i \longrightarrow t_i'}{\langle l_i{=}t_i\rangle \text{ as } T \longrightarrow \langle l_i{=}t_i'\rangle \text{ as } T} \qquad \text{(E-Variant)}$$

New typing rules $\boxed{\Gamma \vdash t : T}$

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j{=}t_j\rangle \text{ as } \langle l_i{:}T_i \;{}^{i\in 1..n}\rangle : \langle l_i{:}T_i \;{}^{i\in 1..n}\rangle} \qquad \text{(T-Variant)}$$

$$\frac{\Gamma \vdash t_0 : \langle l_i{:}T_i \;{}^{i\in 1..n}\rangle \qquad \text{for each } i \quad \Gamma, x_i{:}T_i \vdash t_i : T}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i{=}x_i\rangle \Rightarrow t_i \;{}^{i\in 1..n} : T} \qquad \text{(T-Case)}$$

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 = <l:T>

  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)

$\rightarrow$ fix *Extends* $\lambda_{\rightarrow}$ *(9-1)*

New syntactic forms

$\mathtt{t} ::= \ldots$ *terms:*

$\quad \mathtt{fix\ t}$ *fixed point of* t

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t'}}$

$$\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T_1}.\mathtt{t_2}) \longrightarrow [\mathtt{x} \mapsto (\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T_1}.\mathtt{t_2}))]\mathtt{t_2} \qquad \text{(E-FixBeta)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{fix\ t_1} \longrightarrow \mathtt{fix\ t_1'}} \qquad \text{(E-Fix)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1} \rightarrow \mathtt{T_1}}{\Gamma \vdash \mathtt{fix\ t_1} : \mathtt{T_1}} \qquad \text{(T-Fix)}$$

New derived forms

$$\mathtt{letrec\ x{:}T_1 = t_1\ in\ t_2} \stackrel{\text{def}}{=} \mathtt{let\ x = fix\ (\lambda x{:}T_1.t_1)\ in\ t_2}$$

- Example 1:

```
ff = λie:Nat→Bool.
        λx:Nat.
          if iszero x then true
          else if iszero (pred x) then false
          else ie (pred (pred x));
```

```
▸ ff : (Nat→Bool) → Nat → Bool
```

```
iseven = fix ff;
```

```
▸ iseven : Nat → Bool
```

```
iseven 7;
```

```
▸ false : Bool
```

- Example 2:

```
ff = λieio:{iseven:Nat→Bool, isodd:Nat→Bool}.
       {iseven = λx:Nat.
                   if iszero x then true
                   else ieio.isodd (pred x),
        isodd = λx: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)

```
divergeT = λ_:Unit. fix (λx:T.x);
```

$\blacktriangleright$ $\texttt{diverge}_T$: Unit → T

Lists

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

$\rightarrow$ $\mathbb{B}$ List — *Extends $\lambda_{\rightarrow}$ (9-1) with booleans (8-1)*

New syntactic forms

$$
\begin{array}{llll}
\mathtt{t} & ::= & \dots & \textit{terms:} \\
 & & \mathtt{nil[T]} & \textit{empty list} \\
 & & \mathtt{cons[T]\ t\ t} & \textit{list constructor} \\
 & & \mathtt{isnil[T]\ t} & \textit{test for empty list} \\
 & & \mathtt{head[T]\ t} & \textit{head of a list} \\
 & & \mathtt{tail[T]\ t} & \textit{tail of a list} \\
\mathtt{v} & ::= & \dots & \textit{values:} \\
 & & \mathtt{nil[T]} & \textit{empty list} \\
 & & \mathtt{cons[T]\ v\ v} & \textit{list constructor} \\
\mathtt{T} & ::= & \dots & \textit{types:} \\
 & & \mathtt{List\ T} & \textit{type of lists}
\end{array}
$$

New evaluation rules $\boxed{\mathtt{t} \longrightarrow \mathtt{t'}}$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{cons[T]\ t_1\ t_2} \longrightarrow \mathtt{cons[T]\ t_1'\ t_2}} \quad \text{(E-Cons1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{cons[T]\ v_1\ t_2} \longrightarrow \mathtt{cons[T]\ v_1\ t_2'}} \quad \text{(E-Cons2)}$$

$$\mathtt{isnil[S]\ (nil[T])} \longrightarrow \mathtt{true} \quad \text{(E-IsnilNil)}$$

$$\mathtt{isnil[S]\ (cons[T]\ v_1\ v_2)} \longrightarrow \mathtt{false} \quad \text{(E-IsnilCons)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{isnil[T]\ t_1} \longrightarrow \mathtt{isnil[T]\ t_1'}} \quad \text{(E-Isnil)}$$

$$\mathtt{head[S]\ (cons[T]\ v_1\ v_2)} \longrightarrow \mathtt{v_1} \quad \text{(E-HeadCons)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{head[T]\ t_1} \longrightarrow \mathtt{head[T]\ t_1'}} \quad \text{(E-Head)}$$

$$\mathtt{tail[S]\ (cons[T]\ v_1\ v_2)} \longrightarrow \mathtt{v_2} \quad \text{(E-TailCons)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{tail[T]\ t_1} \longrightarrow \mathtt{tail[T]\ t_1'}} \quad \text{(E-Tail)}$$

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\Gamma \vdash \mathtt{nil\ [T_1]} : \mathtt{List\ T_1} \quad \text{(T-Nil)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{List\ T_1}}{\Gamma \vdash \mathtt{cons[T_1]\ t_1\ t_2} : \mathtt{List\ T_1}} \quad \text{(T-Cons)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{List\ T_{11}}}{\Gamma \vdash \mathtt{isnil[T_{11}]\ t_1} : \mathtt{Bool}} \quad \text{(T-Isnil)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{List\ T_{11}}}{\Gamma \vdash \mathtt{head[T_{11}]\ t_1} : \mathtt{T_{11}}} \quad \text{(T-Head)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{List\ T_{11}}}{\Gamma \vdash \mathtt{tail[T_{11}]\ t_1} : \mathtt{List\ T_{11}}} \quad \text{(T-Tail)}$$

Homework

- Read Chapter 11.
- Do Exercise 11.11.2.