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.

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

New syntactic forms

T	::=	...	*types:*
		A	*base type*

A, B, C, ...

λx:A. x;

<fun>: A→A

λx:B. x;

<fun>: B→B

λf:A→A. λx:A. f(f(x));

<fun>: (A→A)→A→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 $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

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

New derived forms

$$\mathtt{t}_1 ; \mathtt{t}_2 \stackrel{\text{def}}{=} (\lambda \mathtt{x}{:}\mathtt{Unit}.\mathtt{t}_2)\ \mathtt{t}_1 \quad \text{where } \mathtt{x} \notin FV(\mathtt{t}_2)$$

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

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

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

  - New typing rules

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

Derived Form: Sequencing t_1 ; t_2

- Derived form (λ^I): syntactic sugar

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

- **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 (λx: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

$t ::= \ldots$ *terms:*

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

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

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

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

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

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

verification

Let Bindings

- To give names to some of its subexpressions.

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

New syntactic forms

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

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

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

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

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

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

- 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 $\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

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

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

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

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

Tuples

Generalization: binary ➔ n-ary products

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

New syntactic forms

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

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

$$\{\mathtt{v}_i\ ^{i\in 1..n}\}.\mathtt{j} \longrightarrow \mathtt{v}_j \quad \text{(E-ProjTuple)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.\mathtt{i} \longrightarrow \mathtt{t}_1'.\mathtt{i}} \quad \text{(E-Proj)}$$

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}_j'}{\begin{array}{l}\{\mathtt{v}_i\ ^{i\in 1..j-1}, \mathtt{t}_j, \mathtt{t}_k\ ^{k\in j+1..n}\}\\ \longrightarrow \{\mathtt{v}_i\ ^{i\in 1..j-1}, \mathtt{t}_j', \mathtt{t}_k\ ^{k\in j+1..n}\}\end{array}} \quad \text{(E-Tuple)}$$

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

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\mathtt{T}_i\ ^{i\in 1..n}\}}{\Gamma \vdash \mathtt{t}_1.\mathtt{j} : \mathtt{T}_j} \quad \text{(T-Proj)}$$

Records

Generalization: n-ary products ➔ labeled records

$\rightarrow$ {} *Extends* $\lambda_{\rightarrow}$ *(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 $\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

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.

→ + *Extends* $\lambda_\to$ *(9-1)*

New syntactic forms

t ::=	...	*terms:*
	inl t	*tagging (left)*
	inr t	*tagging (right)*
	case t of inl x⇒t | inr x⇒t	*case*
v ::=	...	*values:*
	inl v	*tagged value (left)*
	inr v	*tagged value (right)*
T ::=	...	*types:*
	T+T	*sum type*

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

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

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

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

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

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

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

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

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

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

Sums (with Unique Typing)

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

New syntactic forms

$$\begin{array}{lll} t ::= & \dots & \textit{terms:} \\ & \texttt{inl}\ t\ \boxed{\texttt{as}\ T} & \textit{tagging (left)} \\ & \texttt{inr}\ t\ \boxed{\texttt{as}\ T} & \textit{tagging (right)} \end{array}$$

$$\begin{array}{lll} v ::= & \dots & \textit{values:} \\ & \texttt{inl}\ v\ \boxed{\texttt{as}\ T} & \textit{tagged value (left)} \\ & \texttt{inr}\ v\ \boxed{\texttt{as}\ T} & \textit{tagged value (right)} \end{array}$$

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

$$\begin{array}{l} \texttt{case}\ (\texttt{inl}\ v_0\ \boxed{\texttt{as}\ T_0}) \\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \\ \quad \longrightarrow [x_1 \mapsto v_0]t_1 \end{array} \qquad \text{(E-CaseInl)}$$

$$\begin{array}{l} \texttt{case}\ (\texttt{inr}\ v_0\ \boxed{\texttt{as}\ T_0}) \\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \\ \quad \longrightarrow [x_2 \mapsto v_0]t_2 \end{array} \qquad \text{(E-CaseInr)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inl}\ t_1\ \boxed{\texttt{as}\ T_2} \longrightarrow \texttt{inl}\ t_1'\ \boxed{\texttt{as}\ T_2}} \qquad \text{(E-Inl)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inr}\ t_1\ \boxed{\texttt{as}\ T_2} \longrightarrow \texttt{inr}\ t_1'\ \boxed{\texttt{as}\ T_2}} \qquad \text{(E-Inr)}$$

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

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

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

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

t	::=	...	*terms:*
		$\langle l{=}t\rangle$ as T	*tagging*
		case t of $\langle l_i{=}x_i\rangle \Rightarrow t_i$ $^{i\in 1..n}$	*case*
T	::=	...	*types:*
		$\langle l_i{:}T_i\ ^{i\in 1..n}\rangle$	*type of variants*

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

t	::=	...	*terms:*
		fix t	*fixed point of t*

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

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

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

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

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

New derived forms

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

- 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);
▸ divergeT : Unit → T
```

Lists

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

→ 𝔹 List *Extends λ→ (9-1) with booleans (8-1)*

New syntactic forms

t ::=	...		*terms:*
	nil[T]		*empty list*
	cons[T] t t		*list constructor*
	isnil[T] t		*test for empty list*
	head[T] t		*head of a list*
	tail[T] t		*tail of a list*
v ::=	...		*values:*
	nil[T]		*empty list*
	cons[T] v v		*list constructor*
T ::=	...		*types:*
	List T		*type of lists*

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{cons[T]}\ t_1\ t_2 \longrightarrow \texttt{cons[T]}\ t_1'\ t_2} \quad \text{(E-CONS1)}$$

$$\frac{t_2 \longrightarrow t_2'}{\texttt{cons[T]}\ v_1\ t_2 \longrightarrow \texttt{cons[T]}\ v_1\ t_2'} \quad \text{(E-CONS2)}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

Homework

- Read Chapter 11.
- Do Exercise 11.11.2.

11.11.2 EXERCISE [★]: Rewrite your definitions of `plus`, `times`, and `factorial` from Exercise 11.11.1 using `letrec` instead of `fix`. □