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 *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 $\boxed{\Gamma \vdash t : T}$

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

New derived forms

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

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

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

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

  - 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

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

- **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

  checking if the term t has the type T

  - Useful for documentation and pinpointing error sources
  - Useful for controlling type printing
  - Useful for specializing types (after learning subtyping)

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

New syntactic forms

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

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

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

$$\mathtt{v}_1\ \mathtt{as\ T} \longrightarrow \mathtt{v}_1 \quad \text{(E-ASCRIBE)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{as\ T} \longrightarrow \mathtt{t}_1'\ \mathtt{as\ T}} \quad \text{(E-ASCRIBE1)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{as\ T} : \mathtt{T}} \quad \text{(T-ASCRIBE)}$$

verification

Let Bindings

- To give names to some of its subexpressions.

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

New syntactic forms

$$\texttt{t} ::= \ldots \qquad \textit{terms:}$$

$$\texttt{let x=t in t} \qquad \textit{let binding}$$

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

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

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

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

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

- Is "let binding" a derived form?

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

- Desugaring needs to be performed with type infromation

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

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

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

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

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

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

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

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

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

Records

Generalization: n-ary products ➔ labeled records

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

New syntactic forms

$$
\begin{array}{llll}
\mathtt{t} & ::= & \ldots & \textit{terms:} \\
 & & \{\mathtt{l}_i\mathtt{=t}_i{}^{\,i\in 1..n}\} & \textit{record} \\
 & & \mathtt{t.l} & \textit{projection}
\end{array}
$$

$$
\begin{array}{llll}
\mathtt{v} & ::= & \ldots & \textit{values:} \\
 & & \{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..n}\} & \textit{record value}
\end{array}
$$

$$
\begin{array}{llll}
\mathtt{T} & ::= & \ldots & \textit{types:} \\
 & & \{\mathtt{l}_i\mathtt{:T}_i{}^{\,i\in 1..n}\} & \textit{type of records}
\end{array}
$$

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

$$\{\mathtt{l}_i\mathtt{=v}_i{}^{\,i\in 1..n}\}\mathtt{.l}_j \longrightarrow \mathtt{v}_j \qquad \text{(E-ProjRcd)}$$

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

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

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

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

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

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.

$\rightarrow$ $+$ *Extends* $\lambda_{\rightarrow}$ *(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 $\boxed{\Gamma \vdash t : T}$

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

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

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

Sums (with Unique Typing)

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

New syntactic forms

$$t ::= \dots \quad \textit{terms:}$$
$$\texttt{inl}\ t\ \texttt{as}\ T \quad \textit{tagging (left)}$$
$$\texttt{inr}\ t\ \texttt{as}\ T \quad \textit{tagging (right)}$$

$$v ::= \dots \quad \textit{values:}$$
$$\texttt{inl}\ v\ \texttt{as}\ T \quad \textit{tagged value (left)}$$
$$\texttt{inr}\ v\ \texttt{as}\ T \quad \textit{tagged value (right)}$$

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

$$\texttt{case}\ (\texttt{inl}\ v_0\ \texttt{as}\ T_0)\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_0]t_1 \quad \text{(E-CASEINL)}$$

$$\texttt{case}\ (\texttt{inr}\ v_0\ \texttt{as}\ T_0)\ \texttt{of}\ \texttt{inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_0]t_2 \quad \text{(E-CASEINR)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inl}\ t_1\ \texttt{as}\ T_2 \longrightarrow \texttt{inl}\ t_1'\ \texttt{as}\ T_2} \quad \text{(E-INL)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{inr}\ t_1\ \texttt{as}\ T_2 \longrightarrow \texttt{inr}\ t_1'\ \texttt{as}\ T_2} \quad \text{(E-INR)}$$

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

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

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

Variant

- Generalization: Sums → Labeled variants
  - T1 + T2 → <l1:T1, l2:T2>
  - inl t as T1+T2 → <l1=t> as <l1:T1, l2:T2>
- 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
```

→ <> *Extends* $\lambda_{\to}$ *(9-1)*

New syntactic forms

t	::=	...	*terms:*
		<l=t> as T	*tagging*
		case t of $<\mathtt{l}_i=\mathtt{x}_i>\Rightarrow \mathtt{t}_i\ ^{i\in 1..n}$	*case*
T	::=	...	*types:*
		$<\mathtt{l}_i:\mathtt{T}_i\ ^{i\in 1..n}>$	*type of variants*

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

$$\text{case } (<\mathtt{l}_j=\mathtt{v}_j> \text{ as } \mathtt{T}) \text{ of } <\mathtt{l}_i=\mathtt{x}_i>\Rightarrow \mathtt{t}_i\ ^{i\in 1..n} \longrightarrow [\mathtt{x}_j \mapsto \mathtt{v}_j]\mathtt{t}_j \quad \text{(E-CaseVariant)}$$

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\text{case } \mathtt{t}_0 \text{ of } <\mathtt{l}_i=\mathtt{x}_i>\Rightarrow \mathtt{t}_i\ ^{i\in 1..n} \longrightarrow \text{case } \mathtt{t}_0' \text{ of } <\mathtt{l}_i=\mathtt{x}_i>\Rightarrow \mathtt{t}_i\ ^{i\in 1..n}} \quad \text{(E-Case)}$$

$$\frac{\mathtt{t}_i \longrightarrow \mathtt{t}_i'}{<\mathtt{l}_i=\mathtt{t}_i> \text{ as } \mathtt{T} \longrightarrow <\mathtt{l}_i=\mathtt{t}_i'> \text{ as } \mathtt{T}} \quad \text{(E-Variant)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_j : \mathtt{T}_j}{\Gamma \vdash <\mathtt{l}_j=\mathtt{t}_j> \text{ as } <\mathtt{l}_i:\mathtt{T}_i\ ^{i\in 1..n}> : <\mathtt{l}_i:\mathtt{T}_i\ ^{i\in 1..n}>} \quad \text{(T-Variant)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : <\mathtt{l}_i:\mathtt{T}_i\ ^{i\in 1..n}> \qquad \text{for each } i \quad \Gamma, \mathtt{x}_i:\mathtt{T}_i \vdash \mathtt{t}_i : \mathtt{T}}{\Gamma \vdash \text{case } \mathtt{t}_0 \text{ of } <\mathtt{l}_i=\mathtt{x}_i>\Rightarrow \mathtt{t}_i\ ^{i\in 1..n} : \mathtt{T}} \quad \text{(T-Case)}$$

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)

→ fix *Extends λ→ (9-1)*

New syntactic forms

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

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

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

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

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

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

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 (λx:T. x)

$$\text{diverge}_T = \lambda\_:\text{Unit}.\ \text{fix}\ (\lambda x:T.x);$$

$$\blacktriangleright\ \text{diverge}_T : \text{Unit} \to T$$

Lists

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

$\to \mathbb{B}$ List *Extends $\lambda_{\to}$ (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 $\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Homework

- Read Chapter 11.
- Do Exercise 11.11.1.

11.11.1 EXERCISE [★★]: Define `equal`, `plus`, `times`, and `factorial` using `fix`. □