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* $\lambda_{\rightarrow}$ *(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

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

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

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

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

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

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

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

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

verification

Let Bindings

- To give names to some of its subexpressions.

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

New syntactic forms

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

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

$$\texttt{let } x\texttt{=}v_1 \texttt{ in } t_2 \longrightarrow [x \mapsto v_1]t_2 \qquad \text{(E-LetV)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{let } x\texttt{=}t_1 \texttt{ in } t_2 \longrightarrow \texttt{let } x\texttt{=}t_1' \texttt{ in } t_2} \qquad \text{(E-Let)}$$

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

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \texttt{let } x\texttt{=}t_1 \texttt{ in } t_2 : T_2} \qquad \text{(T-Let)}$$

- Is “let binding” a derived form?

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

- Desugaring is not on terms but on typing derivations

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

⇩

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

Pairs

- To build compound data structures.

→ × *Extends λ→ (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 $t \longrightarrow t'$

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

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

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

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

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

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

New typing rules $\Gamma \vdash t : T$

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

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

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

Tuples

Generalization: binary ➔ n-ary products

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

New syntactic forms

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 $\boxed{t \longrightarrow t'}$

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

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

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

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

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

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

Records

Generalization: n-ary products ➔ labeled records

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

New syntactic forms

$\texttt{t} ::= \ldots$		*terms:*
	$\{\texttt{l}_i\texttt{=t}_i\ ^{i\in 1..n}\}$	*record*
	$\texttt{t.l}$	*projection*
$\texttt{v} ::= \ldots$		*values:*
	$\{\texttt{l}_i\texttt{=v}_i\ ^{i\in 1..n}\}$	*record value*
$\texttt{T} ::= \ldots$		*types:*
	$\{\texttt{l}_i\texttt{:T}_i\ ^{i\in 1..n}\}$	*type of records*

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

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

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

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

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

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

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

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 $\quad t \longrightarrow t'$

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

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

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

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

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

New typing rules $\quad \Gamma \vdash t : T$

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

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

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

Sums (with Unique Typing)

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

New syntactic forms

t ::=	...		*terms:*
	inl t as T		*tagging (left)*
	inr t as T		*tagging (right)*
v ::=	...		*values:*
	inl v as T		*tagged value (left)*
	inr v as T		*tagged value (right)*

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

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

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

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

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

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
```

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

New syntactic forms

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

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

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

$$\frac{\mathtt{t_0} \longrightarrow \mathtt{t_0'}}{\mathtt{case\ t_0\ of\ <l_i=x_i>\Rightarrow t_i}\ ^{i\in 1..n} \longrightarrow \mathtt{case\ t_0'\ of\ <l_i=x_i>\Rightarrow t_i}\ ^{i\in 1..n}} \quad \text{(E-Case)}$$

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

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

$$\frac{\Gamma \vdash \mathtt{t_j} : \mathtt{T_j}}{\Gamma \vdash \mathtt{<l_j=t_j>\ as\ <l_i:T_i}\ ^{i\in 1..n}\mathtt{>} : \mathtt{<l_i:T_i}\ ^{i\in 1..n}\mathtt{>}} \quad \text{(T-Variant)}$$

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

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)

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

New syntactic forms

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

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

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

$$\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2) \longrightarrow [\mathtt{x} \mapsto (\mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2))]\mathtt{t}_2 \qquad \text{(E-FixBeta)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{fix}\ \mathtt{t}_1 \longrightarrow \mathtt{fix}\ \mathtt{t}_1'} \qquad \text{(E-Fix)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \rightarrow \mathtt{T}_1}{\Gamma \vdash \mathtt{fix}\ \mathtt{t}_1 : \mathtt{T}_1} \qquad \text{(T-Fix)}$$

New derived forms

$$\mathtt{letrec}\ \mathtt{x}{:}\mathtt{T}_1 = \mathtt{t}_1\ \mathtt{in}\ \mathtt{t}_2 \stackrel{\text{def}}{=} \mathtt{let}\ \mathtt{x} = \mathtt{fix}\ (\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_1)\ \mathtt{in}\ \mathtt{t}_2$$

- 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)

```
diverge_T = λ_:Unit. fix (λx:T.x);
▸ diverge_T : Unit → T
```

Lists

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

→ 𝔹 List — *Extends* $\lambda_{\rightarrow}$ *(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 11.11.1.

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

Next class is programming exercise.
Please download arith and fullsimple.
Bring your laptop.