

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 λ_{\dots} (9-1)

New syntactic forms

T ::= ...
A

*types:
 base type*

A, B, C, ...



$\lambda x:A. x;$
 $\langle \text{fun} \rangle: A \rightarrow A$

$\lambda x:B. x;$
 $\langle \text{fun} \rangle: B \rightarrow B$

$\lambda f:A \rightarrow A. \lambda x:A. f(f(x));$
 $\langle \text{fun} \rangle: (A \rightarrow A) \rightarrow A \rightarrow A$



The Unit Type

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

→ Unit

Extends λ_{\dots} (9-1)

New syntactic forms

$t ::= \dots$
Unit

terms:
constant unit

$v ::= \dots$
Unit

values:
constant unit

$T ::= \dots$
Unit

types:
unit type

New typing rules

$\Gamma \vdash \text{unit} : \text{Unit}$

$\Gamma \vdash t : T$

(T-UNIT)

New derived forms

$t_1 ; t_2 \stackrel{\text{def}}{=} (\lambda x : \text{Unit}. t_2) t_1$
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)
 - $t ::= \dots$
 $t_1 ; t_2$
 - New valuation relation rules

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

$$\text{unit} ; t_2 \rightarrow t_2 \quad (\text{E-SEQNEXT})$$

- New typing rules

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



Derived Form: Sequencing $t_1 ; t_2$

- Derived form (λ^I): syntactic sugar

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

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 $t_1 ; t_2$ with $(\lambda x : \text{Unit}. t_2) t_1$. Then

- $t \rightarrow_E t'$ iff $e(t) \rightarrow_I e(t')$
- $\Gamma \vdash^E t : T$ 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

→ **as**

Extends λ_{\rightarrow} (9-1)

New syntactic forms

$t ::= \dots$
 t as T

terms:
ascription

New typing rules

$\Gamma \vdash t : T$

$$\frac{\Gamma \vdash t_1 : T}{\Gamma \vdash t_1 \text{ as } T : T}$$

(T-ASCRIBE)

New evaluation rules

v_1 as $T \rightarrow v_1$

$t \rightarrow t'$
(E-ASCRIBE)

$$\frac{t_1 \rightarrow t'_1}{t_1 \text{ as } T \rightarrow t'_1 \text{ as } T}$$

(E-ASCRIBE1)

verification



Let Bindings

- To give names to some of its subexpressions.

→ **let**

Extends λ_{\rightarrow} (9-1)

New syntactic forms

$t ::= \dots$

let $x=t$ in t

*terms:
let binding*

$$\frac{t_1 \rightarrow t'_1}{\text{let } x=t_1 \text{ in } t_2 \rightarrow \text{let } x=t'_1 \text{ in } t_2} \quad (\text{E-LET})$$

New evaluation rules

let $x=v_1$ in $t_2 \rightarrow [x \mapsto v_1]t_2$

$t \rightarrow t'$
(E-LETV)

New typing rules

$\Gamma \vdash t : T$

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



- Is “let binding” a derived form?

Yes, 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

$$\begin{array}{c}
 \begin{array}{c} \vdots \\ \hline \Gamma \vdash t_1 : T_1 \end{array} \quad \begin{array}{c} \vdots \\ \hline \Gamma, x:T_1 \vdash t_2 : T_2 \end{array} \\
 \hline
 \Gamma \vdash \text{let } x=t_1 \text{ in } t_2 : T_2 \quad \text{T-LET}
 \end{array}$$

↓

$$\begin{array}{c}
 \begin{array}{c} \vdots \\ \hline \Gamma, x:T_1 \vdash t_2 : T_2 \end{array} \quad \begin{array}{c} \vdots \\ \hline \Gamma \vdash t_1 : T_1 \end{array} \\
 \hline
 \Gamma \vdash (\lambda x:T_1.t_2) t_1 : T_2 \quad \text{T-APP}
 \end{array}$$



Pairs

- To build compound data structures.

→ ×

Extends λ_{\rightarrow} (9-1)

New syntactic forms

$t ::= \dots$
 $\{t, t\}$
 $t.1$
 $t.2$

terms:
 pair
 first projection
 second projection

$v ::= \dots$
 $\{v, v\}$

values:
 pair value

$T ::= \dots$
 $T_1 \times T_2$

types:
 product type

New evaluation rules

$\{v_1, v_2\}.1 \rightarrow v_1$

(E-PAIRBETA1)

$\{v_1, v_2\}.2 \rightarrow v_2$

(E-PAIRBETA2)

$\frac{t_1 \rightarrow t'_1}{t_1.1 \rightarrow t'_1.1}$

(E-PROJ1)

$t \rightarrow t'$

$\frac{t_1 \rightarrow t'_1}{t_1.2 \rightarrow t'_1.2}$

(E-PROJ2)

$\frac{t_1 \rightarrow t'_1}{\{t_1, t_2\} \rightarrow \{t'_1, t_2\}}$

(E-PAIR1)

$\frac{t_2 \rightarrow t'_2}{\{v_1, t_2\} \rightarrow \{v_1, t'_2\}}$

(E-PAIR2)

New typing rules

$\Gamma \vdash t : T$

$\frac{\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2}$

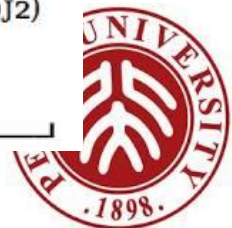
(T-PAIR)

$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.1 : T_{11}}$

(T-PROJ1)

$\frac{\Gamma \vdash t_1 : T_{11} \times T_{12}}{\Gamma \vdash t_1.2 : T_{12}}$

(T-PROJ2)



Tuples

Generalization: binary \rightarrow n-ary products

$\rightarrow \{\}$

Extends λ_{\dots} (9-1)

New syntactic forms

$t ::= \dots$ *terms:*
 $\{t_i^{i \in 1..n}\}$ *tuple*
 $t.i$ *projection*

$v ::= \dots$ *values:*
 $\{v_i^{i \in 1..n}\}$ *tuple value*

$T ::= \dots$ *types:*
 $\{T_i^{i \in 1..n}\}$ *tuple type*

New evaluation rules

$\{v_i^{i \in 1..n}\}.j \rightarrow v_j$ (E-PROJTUPLE) $t \rightarrow t'$

$$\frac{t_1 \rightarrow t'_1}{t_1.i \rightarrow t'_1.i}$$
 (E-PROJ)

$$\frac{t_j \rightarrow t'_j}{\{v_i^{i \in 1..j-1}, t_j, t_k^{k \in j+1..n}\} \rightarrow \{v_i^{i \in 1..j-1}, t'_j, t_k^{k \in j+1..n}\}}$$
 (E-TUPLE)

New typing rules

$\Gamma \vdash t : T$

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

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



Records

Generalization: n-ary products \rightarrow labeled records

$\rightarrow \{\}$

Extends λ_{\dots} (9-1)

New syntactic forms

$t ::= \dots$
 $\{\lambda_i = t_i \mid i \in 1..n\}$
 $t.l$

terms:
record
projection

$v ::= \dots$
 $\{\lambda_i = v_i \mid i \in 1..n\}$

values:
record value

$T ::= \dots$
 $\{\lambda_i : T_i \mid i \in 1..n\}$

types:
type of records

New evaluation rules

$\{\lambda_i = v_i \mid i \in 1..n\}.l_j \rightarrow v_j$

$t \rightarrow t'$
 (E-PROJRCD)

$\frac{t_1 \rightarrow t'_1}{t_1.l \rightarrow t'_1.l}$

(E-PROJ)

$\frac{t_j \rightarrow t'_j}{\{\lambda_i = v_i \mid i \in 1..j-1, \lambda_j = t_j, \lambda_k = t_k \mid k \in j+1..n\} \rightarrow \{\lambda_i = v_i \mid i \in 1..j-1, \lambda_j = t'_j, \lambda_k = t_k \mid k \in j+1..n\}}$

(E-RCD)

New typing rules

$\Gamma \vdash t : T$

$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{\lambda_i = t_i \mid i \in 1..n\} : \{\lambda_i : T_i \mid i \in 1..n\}}$

(T-RCD)

$\frac{\Gamma \vdash t_1 : \{\lambda_i : T_i \mid i \in 1..n\}}{\Gamma \vdash t_1.l_j : T_j}$

(T-PROJ)

Question: $\{\text{partno}=5524, \text{cost}=30.27\} = \{\text{cost}=30.27, \text{partno}=5524\}$?



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 λ_{\rightarrow} (9-1)

| | | |
|--|--------------------|---|
| <p><i>New syntactic forms</i></p> <p>$t ::= \dots$ <i>terms:</i></p> <p style="padding-left: 20px;">$\text{inl } t$ <i>tagging (left)</i></p> <p style="padding-left: 20px;">$\text{inr } t$ <i>tagging (right)</i></p> <p style="padding-left: 20px;">$\text{case } t \text{ of } \text{inl } x \Rightarrow t \mid \text{inr } x \Rightarrow t$ <i>case</i></p> <p>$v ::= \dots$ <i>values:</i></p> <p style="padding-left: 20px;">$\text{inl } v$ <i>tagged value (left)</i></p> <p style="padding-left: 20px;">$\text{inr } v$ <i>tagged value (right)</i></p> <p>$T ::= \dots$ <i>types:</i></p> <p style="padding-left: 20px;">$T+T$ <i>sum type</i></p> <p><i>New evaluation rules</i></p> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\text{case } (\text{inl } v_0) \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow [x_1 \mapsto v_0]t_1$ <i>(E-CASEINL)</i> </div> <div style="border: 1px solid gray; padding: 5px;"> $\text{case } (\text{inr } v_0) \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow [x_2 \mapsto v_0]t_2$ <i>(E-CASEINR)</i> </div> | $t \rightarrow t'$ | <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\frac{t_0 \rightarrow t'_0}{\text{case } t_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \rightarrow \text{case } t'_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2}$ <i>(E-CASE)</i> </div> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\frac{t_1 \rightarrow t'_1}{\text{inl } t_1 \rightarrow \text{inl } t'_1}$ <i>(E-INL)</i> </div> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\frac{t_1 \rightarrow t'_1}{\text{inr } t_1 \rightarrow \text{inr } t'_1}$ <i>(E-INR)</i> </div> <p><i>New typing rules</i> $\Gamma \vdash t : T$</p> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{inl } t_1 : T_1+T_2}$ <i>(T-INL)</i> </div> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 10px;"> $\frac{\Gamma \vdash t_1 : T_2}{\Gamma \vdash \text{inr } t_1 : T_1+T_2}$ <i>(T-INR)</i> </div> <div style="border: 1px solid gray; padding: 5px;"> $\frac{\Gamma \vdash t_0 : T_1+T_2 \quad \Gamma, x_1:T_1 \vdash t_1 : T \quad \Gamma, x_2:T_2 \vdash t_2 : T}{\Gamma \vdash \text{case } t_0 \text{ of } \text{inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 : T}$ <i>(T-CASE)</i> </div> |
|--|--------------------|---|



Sums (with Unique Typing)

→ +

Extends λ_{\rightarrow} (11-9)

New syntactic forms

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

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

New evaluation rules

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

$t \rightarrow t'$

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

$\frac{t_1 \rightarrow t'_1}{\text{inl } t_1 \text{ as } T_2 \rightarrow \text{inl } t'_1 \text{ as } T_2}$ (E-INL)

$\frac{t_1 \rightarrow t'_1}{\text{inr } t_1 \text{ as } T_2 \rightarrow \text{inr } t'_1 \text{ as } T_2}$ (E-INR)

New typing rules

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

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

$\Gamma \vdash t : T$



Variant

- Generalization: Sums \rightarrow Labeled variants
 - $T1 + T2 \rightarrow \langle l1:T1, l2:T2 \rangle$
 - $\text{inl } t \text{ as } T1+T2 \rightarrow \langle l1=t \rangle \text{ as } \langle l1:T1, l2:T2 \rangle$
- Example:

```
Addr = <physical:PhysicalAddr, virtual:VirtualAddr>;
```

```
a = <physical=pa> as Addr;
```

▶ $a : \text{Addr}$

```
getName =  $\lambda a:\text{Addr}.$ 
```

```
  case a of
```

```
    <physical=x>  $\Rightarrow$  x.firstlast
```

```
  | <virtual=y>  $\Rightarrow$  y.name;
```

▶ $\text{getName} : \text{Addr} \rightarrow \text{String}$



→ <>

Extends λ_{\rightarrow} (9-1)

New syntactic forms

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

$T ::= \dots$ *types:*
 $\langle l_i : T_i^{i \in 1..n} \rangle$ *type of variants*

New evaluation rules

$t \rightarrow t'$

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

$$\frac{t_0 \rightarrow t'_0}{\text{case } t_0 \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i^{i \in 1..n} \rightarrow \text{case } t'_0 \text{ of } \langle l_i=x_i \rangle \Rightarrow t_i^{i \in 1..n}}$$
 (E-CASE)

$$\frac{t_i \rightarrow t'_i}{\langle l_i=t_i \rangle \text{ as } T \rightarrow \langle l_i=t'_i \rangle \text{ as } T}$$
 (E-VARIANT)

New typing rules

$\Gamma \vdash t : T$

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

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

$V = \langle !:T \rangle$

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: $\text{fix } f = f (\text{fix } f)$

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

→ **fix**

Extends λ_{\rightarrow} (9-1)

New syntactic forms

$t ::= \dots$

fix t

terms:

fixed point of t

New typing rules

$\Gamma \vdash t : T$

$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_1}{\Gamma \vdash \text{fix } t_1 : T_1}$$

(T-FIX)

New evaluation rules

$t \rightarrow t'$

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

$$\frac{t_1 \rightarrow t'_1}{\text{fix } t_1 \rightarrow \text{fix } t'_1} \quad (\text{E-FIX})$$

New derived forms

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



- 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));
```

▶ $ff : (\text{Nat} \rightarrow \text{Bool}) \rightarrow \text{Nat} \rightarrow \text{Bool}$

```
iseven = fix ff;
```

▶ $iseven : \text{Nat} \rightarrow \text{Bool}$

```
iseven 7;
```

▶ $false : \text{Bool}$



- 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)};
```

- ▶ $ff : \{iseven:Nat \rightarrow Bool, isodd:Nat \rightarrow Bool\} \rightarrow \{iseven:Nat \rightarrow Bool, isodd:Nat \rightarrow Bool\}$

```
r = fix ff;
```

- ▶ $r : \{iseven:Nat \rightarrow Bool, isodd:Nat \rightarrow Bool\}$

```
iseven = r.iseven;
```

- ▶ $iseven : Nat \rightarrow Bool$

```
iseven 7;
```

- ▶ $false : Bool$



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

x as T

$\text{fix } (\lambda x:T. x)$

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

▶ $\text{diverge}_T : \text{Unit} \rightarrow T$



Lists

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

Extends λ_{\rightarrow} (9-1) with booleans (8-1)

| | | |
|--|---|--|
| <p>→ B List</p> <p><i>New syntactic forms</i></p> <p>$t ::= \dots$</p> <ul style="list-style-type: none"> $nil[T]$ $cons[T] \ t \ t$ $isnil[T] \ t$ $head[T] \ t$ $tail[T] \ t$ <p>$v ::= \dots$</p> <ul style="list-style-type: none"> $nil[T]$ $cons[T] \ v \ v$ <p>$T ::= \dots$</p> <ul style="list-style-type: none"> $List \ T$ <p><i>New evaluation rules</i></p> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$t \rightarrow t'$</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$cons[T] \ t_1 \ t_2 \rightarrow cons[T] \ t'_1 \ t_2$ (E-CONS1)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$cons[T] \ v_1 \ t_2 \rightarrow cons[T] \ v_1 \ t'_2$ (E-CONS2)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$isnil[S] \ (nil[T]) \rightarrow true$ (E-ISNILNIL)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$isnil[S] \ (cons[T] \ v_1 \ v_2) \rightarrow false$ (E-ISNILCONS)</div> | <p><i>terms:</i></p> <p><i>empty list</i></p> <p><i>list constructor</i></p> <p><i>test for empty list</i></p> <p><i>head of a list</i></p> <p><i>tail of a list</i></p> <p><i>values:</i></p> <p><i>empty list</i></p> <p><i>list constructor</i></p> <p><i>types:</i></p> <p><i>type of lists</i></p> | <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$t_1 \rightarrow t'_1$</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$isnil[T] \ t_1 \rightarrow isnil[T] \ t'_1$ (E-ISNIL)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$head[S] \ (cons[T] \ v_1 \ v_2) \rightarrow v_1$ (E-HEADCONS)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$head[T] \ t_1 \rightarrow head[T] \ t'_1$ (E-HEAD)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$tail[S] \ (cons[T] \ v_1 \ v_2) \rightarrow v_2$ (E-TAILCONS)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$tail[T] \ t_1 \rightarrow tail[T] \ t'_1$ (E-TAIL)</div> <p><i>New typing rules</i></p> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash t : T$</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash nil \ [T_1] : List \ T_1$ (T-NIL)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : List \ T_1$ $\Gamma \vdash cons[T_1] \ t_1 \ t_2 : List \ T_1$ (T-CONS)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash t_1 : List \ T_{11}$ $\Gamma \vdash isnil[T_{11}] \ t_1 : Bool$ (T-ISNIL)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash t_1 : List \ T_{11}$ $\Gamma \vdash head[T_{11}] \ t_1 : T_{11}$ (T-HEAD)</div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-bottom: 10px;">$\Gamma \vdash t_1 : List \ T_{11}$ $\Gamma \vdash tail[T_{11}] \ t_1 : List \ T_{11}$ (T-TAIL)</div> |
|--|---|--|



Homework

- Read Chapter 11.
- Do Exercise 11.11.1.

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

