OCaml

案例

链表去重

```
let rec compress = function
    | a :: (b :: _ as t) -> if a = b then compress t else a :: compress t
    | smaller -> smaller;;

# compress ["a"; "a"; "a"; "a"; "b"; "c"; "c"; "a"; "a"; "d"; "e"; "e"; "e";
"e"];;
- : string list = ["a"; "b"; "c"; "a"; "d"; "e"]
```

元素个数统计及

```
# let encode list =
    List.map (fun l -> (List.length l, List.hd l)) (pack list);;
val encode : 'a list -> (int * 'a) list = <fun>

# encode ["a"; "a"; "a"; "a"; "b"; "c"; "c"; "a"; "a"; "d"; "e"; "e"; "e";
"e"];;
- : (int * string) list =
[(4, "a"); (1, "b"); (2, "c"); (2, "a"); (1, "d"); (4, "e")]
```

重复元素个数

```
# let replicate list n =
    let rec prepend n acc x =
      if n = 0 then acc else prepend (n-1) (x :: acc) x in
    let rec aux acc = function
      | [] -> acc
      | h :: t -> aux (prepend n acc h) t in
    (* This could also be written as:
       List.fold_left (prepend n) [] (List.rev list) *)
    aux [] (List.rev list);;

# replicate ["a"; "b"; "c"] 3;;
- : string list = ["a"; "a"; "a"; "b"; "b"; "b"; "c"; "c"; "c"]
```

链表元素排列组合

```
# let rec permutation list =
    let rec extract acc n = function
      | [] -> raise Not_found
      | h :: t -> if n = 0 then (h, acc @ t) else extract (h :: acc) (n - 1) t
    in
    let extract_rand list len =
      extract [] (Random.int len) list
    in
```

```
    let rec aux acc list len =
      if len = 0 then acc else
        let picked, rest = extract_rand list len in
        aux (picked :: acc) rest (len - 1)
    in
    aux [] list (List.length list);;

# permutation ["a"; "b"; "c"; "d"; "e"; "f"];;
- : string list = ["a"; "e"; "f"; "b"; "d"; "c"]
```

质数检测

```
# let is_prime n =
    let n = abs n in
    let rec is_not_divisor d =
      d * d > n || (n mod d <> 0 && is_not_divisor (d + 1)) in
    n <> 1 && is_not_divisor 2;;
```

二叉树

```
# type 'a binary_tree =
    | Empty
    | Node of 'a * 'a binary_tree * 'a binary_tree;;
type 'a binary_tree = Empty | Node of 'a * 'a binary_tree * 'a binary_tree

# let example_tree =
    Node ('a', Node ('b', Node ('d', Empty, Empty), Node ('e', Empty, Empty)),
         Node ('c', Empty, Node ('f', Node ('g', Empty, Empty), Empty)));;
val example_tree : char binary_tree =
  Node ('a', Node ('b', Node ('d', Empty, Empty), Node ('e', Empty, Empty)),
   Node ('c', Empty, Node ('f', Node ('g', Empty, Empty), Empty)))
# let example_int_tree =
    Node (1, Node (2, Node (4, Empty, Empty), Node (5, Empty, Empty)),
         Node (3, Empty, Node (6, Node (7, Empty, Empty), Empty)));;
val example_int_tree : int binary_tree =
  Node (1, Node (2, Node (4, Empty, Empty), Node (5, Empty, Empty)),
   Node (3, Empty, Node (6, Node (7, Empty, Empty), Empty)))
```

修改变量

```
# let r = ref 0;;
val r : int ref = {contents = 0}
# r := 10;;
- : unit = ()
# !r;;
- : int = 10
#
```

快速排序

```
# let rec quicksort = function
    | [] -  > []
      | x::xs -> let smaller, larger = List.partition(fun y -> y < x) xs in
        quickso  rt smaller@(x::quicksort larger);;
val quicksort : 'a list -> 'a list = <fun>
# quicksort [1; 7; 5; 8; 2; 9; 4];;
- : int list = [1; 2; 4; 5; 7; 8; 9]
# 
```

```
# type mypair = {a: int; b: int};;
type mypair = { a : int; b : int; }
# {a = 1; b = 2};;
- : mypair = {a = 1; b = 2}
# {a = 1};;
Error: Some record fields are undefined: b
# 
```

```
# type foo =
    | Nothing
    | Int of int
    | Pair of int * int
    | String of string;;
type foo = Nothing | Int of int | Pair of int * int | String of string
# Nothing;;
- : foo = Nothing
# Int 1;;
- : foo = Int 1
# Pair (1, 2);;
- : foo = Pair (1, 2)
# String "hello";;
- : foo = String "hello"
# 
```

无类型算术表达式

语法树结点个数 size(t)：

$$
\begin{array}{lcl}
\mathit{size}(\mathtt{true}) & = & 1 \\
\mathit{size}(\mathtt{false}) & = & 1 \\
\mathit{size}(\mathtt{0}) & = & 1 \\
\mathit{size}(\mathtt{succ}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{pred}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{iszero}\ \mathtt{t}_1) & = & \mathit{size}(\mathtt{t}_1) + 1 \\
\mathit{size}(\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3) & = & \mathit{size}(\mathtt{t}_1) + \mathit{size}(\mathtt{t}_2) + \mathit{size}(\mathtt{t}_3) + 1
\end{array}
$$

语法书结点深度：

$$
\begin{aligned}
depth(\texttt{true}) &= 1 \\
depth(\texttt{false}) &= 1 \\
depth(\texttt{0}) &= 1 \\
depth(\texttt{succ } t_1) &= depth(t_1) + 1 \\
depth(\texttt{pred } t_1) &= depth(t_1) + 1 \\
depth(\texttt{iszero } t_1) &= depth(t_1) + 1 \\
depth(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &= \max(depth(t_1), depth(t_2), depth(t_3)) + 1
\end{aligned}
$$

Lambda 演算

三种规约方式：

Full beta-reduction：

$$
\begin{array}{ll}
 & \texttt{id (id (}\lambda\texttt{z. }\underline{\texttt{id z}}\texttt{))} \\
\longrightarrow & \texttt{id }\underline{\texttt{(id (}\lambda\texttt{z.z))}} \\
\longrightarrow & \underline{\texttt{id (}\lambda\texttt{z.z)}} \\
\longrightarrow & \lambda\texttt{z.z} \\
\not\longrightarrow &
\end{array}
$$

正则顺序：每次选择最左、最外侧的redex

$$
\begin{array}{ll}
 & \underline{\texttt{id (id (}\lambda\texttt{z. id z))}} \\
\longrightarrow & \underline{\texttt{id (}\lambda\texttt{z. id z)}} \\
\longrightarrow & \lambda\texttt{z. }\underline{\texttt{id z}} \\
\longrightarrow & \lambda\texttt{z.z} \\
\not\longrightarrow &
\end{array}
$$

传名：严格从外侧归约，函数抽象中不进行归约

$$
\begin{array}{ll}
 & \underline{\texttt{id (id (}\lambda\texttt{z. id z))}} \\
\longrightarrow & \underline{\texttt{id (}\lambda\texttt{z. id z)}} \\
\longrightarrow & \lambda\texttt{z. id z} \\
\not\longrightarrow &
\end{array}
$$

传值：函数参数先归约，也就是说先参数求值再计算函数

$$
\begin{array}{ll}
 & \texttt{id}\ \underline{(\texttt{id}\ (\lambda z.\ \texttt{id}\ z))} \\
\longrightarrow & \underline{\texttt{id}\ (\lambda z.\ \texttt{id}\ z)} \\
\longrightarrow & \lambda z.\ \texttt{id}\ z \\
\not\longrightarrow &
\end{array}
$$

Church encodes

```
tru = λt. λf. t;
fls = λt. λf. f;
```

```
test = λl. λm. λn. l m n;
```

- $and = \lambda b.\lambda c.b\ c\ fls$
- $or = \lambda b.\lambda c.b\ tru\ c$
- $not = \lambda b.b\ fls\ tru$

```
pair = λf.λs.λb. b f s;
fst = λp. p tru;
snd = λp. p fls;
```

$$
\begin{array}{l}
c_0 = \lambda s.\ \lambda z.\ z; \\
c_1 = \lambda s.\ \lambda z.\ s\ z; \\
c_2 = \lambda s.\ \lambda z.\ s\ (s\ z); \\
c_3 = \lambda s.\ \lambda z.\ s\ (s\ (s\ z)); \\
\text{etc.}
\end{array}
$$

```
scc = λn. λs. λz. s (n s z);
```

- $plus = \lambda m.\lambda n.\lambda s.\lambda z.m\ s\ (n\ s\ z)$
- $times = \lambda m.\lambda n.m\ (plus\ n)\ c_0$

- $iszero = \lambda m. m\ (\lambda x. fls)\ tru$
- $zz = pair\ c_0\ c_0$
- $ss = \lambda p. pair\ (snd\ p)\ (plus\ c_1\ (snd\ p))$
- $prd = \lambda m. fst\ (m\ ss\ zz)$
- $sub = \lambda m. \lambda n. m\ prd\ n$

```
fix = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y));
```

替换规则

$$
\begin{array}{lcll}
[x \mapsto s]x & = & s & \\
[x \mapsto s]y & = & y & \text{if } y \neq x \\
[x \mapsto s](\lambda y.t_1) & = & \lambda y.\ [x \mapsto s]t_1 & \text{if } y \neq x \text{ and } y \notin FV(s) \\
[x \mapsto s](t_1\ t_2) & = & [x \mapsto s]t_1\ [x \mapsto s]t_2 &
\end{array}
$$

简单类型表达式

引理：类型关系的倒置 (Inversion of the type relation)

1. If $\texttt{true} : R$, then $R = \texttt{Bool}$.
2. If $\texttt{false} : R$, then $R = \texttt{Bool}$.
3. If $\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : R$, then $t_1 : \texttt{Bool}$, $t_2 : R$, and $t_3 : R$.
4. If $0 : R$, then $R = \texttt{Nat}$.
5. If $\texttt{succ}\ t_1 : R$, then $R = \texttt{Nat}$ and $t_1 : \texttt{Nat}$.
6. If $\texttt{pred}\ t_1 : R$, then $R = \texttt{Nat}$ and $t_1 : \texttt{Nat}$.
7. If $\texttt{iszero}\ t_1 : R$, then $R = \texttt{Bool}$ and $t_1 : \texttt{Nat}$. $\square$

性质：

可靠性/安全性/Soundness/Safty

- Well-typed terms求值不会出错
- 不会到达stuck state

由两个定理来保证可靠性：

- Progress: A well-typed term is not stuck. 要么是一个value，要么存在一条规则进行求值
- Preservation: 一个well-typed term经过若干次求值得到的新term也是well-typed的

例：推断 (λx:Bool .x) true的类型

$$\dfrac{\dfrac{\dfrac{\mathtt{x{:}Bool} \in \mathtt{x{:}Bool}}{\mathtt{x{:}Bool} \vdash \mathtt{x} : \mathtt{Bool}}\ \textsc{T-Var}}{\vdash \lambda\mathtt{x{:}Bool.x} : \mathtt{Bool}\to\mathtt{Bool}}\ \textsc{T-Abs} \qquad \dfrac{}{\vdash \mathtt{true} : \mathtt{Bool}}\ \textsc{T-True}}{\vdash (\lambda\mathtt{x{:}Bool.x})\ \mathtt{true} : \mathtt{Bool}}\ \textsc{T-App}$$

引理：类型关系的倒置

1. If $\Gamma \vdash \mathtt{x} : \mathsf{R}$, then $\mathtt{x}{:}\mathsf{R} \in \Gamma$.
2. If $\Gamma \vdash \lambda\mathtt{x}{:}\mathsf{T}_1.\ \mathtt{t}_2 : \mathsf{R}$, then $\mathsf{R} = \mathsf{T}_1 \to \mathsf{R}_2$ for some R_2 with $\Gamma, \mathtt{x}{:}\mathsf{T}_1 \vdash \mathtt{t}_2 : \mathsf{R}_2$.
3. If $\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathsf{R}$, then there is some type T_{11} such that $\Gamma \vdash \mathtt{t}_1 : \mathsf{T}_{11} \to \mathsf{R}$ and $\Gamma \vdash \mathtt{t}_2 : \mathsf{T}_{11}$.
4. If $\Gamma \vdash \mathtt{true} : \mathsf{R}$, then $\mathsf{R} = \mathtt{Bool}$.
5. If $\Gamma \vdash \mathtt{false} : \mathsf{R}$, then $\mathsf{R} = \mathtt{Bool}$.
6. If $\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathsf{R}$, then $\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool}$ and $\Gamma \vdash \mathtt{t}_2, \mathtt{t}_3 : \mathsf{R}$. □

简单扩展

基本类型

使用抽象的A,B,C这样的符号表示不同的基本类型

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

New syntactic forms

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

例：

λx:A. x

λx:B. x

λf:A→A. λx:A. f(f(x))

The Unit Types

$\rightarrow$ `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

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

$t_1;t_2$ 可以看作 $(\lambda x : Unit. t_2)t_1$ 的缩写，是一种派生型(derived form)，又称为语法糖(syntactic sugar)

通配符(Wildcard)

- 一种语法糖
- 表示不会在函数抽象中使用到的参数
- 用通配符"_"表示这样的参数
- $\lambda\_ : S.t$ 是 $\lambda x : S.t$ 的缩写，其中 x 不在 t 中出现

类型归属(Ascription)

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

New syntactic forms

`t ::= ...`		*terms:*
`t 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 $\boxed{\Gamma \vdash t : T}$

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

Let 绑定 (Let Bindings)

– $let\ x = t_1\ in\ t_2 \overset{\text{def}}{=} (\lambda x{:}T_1.t_2)\ t_1$?

– T_1的类型从type checker得到

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

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

Pairs

$\to \times$ *Extends $\lambda_{\to}$ (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

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

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

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

记录

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

New syntactic forms

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

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

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

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

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

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

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

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

Sums

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

New syntactic forms

$\mathtt{t}$ ::=	...	*terms:*
	$\mathtt{inl\ t\ as\ T}$	*tagging (left)*
	$\mathtt{inr\ t\ as\ T}$	*tagging (right)*
$\mathtt{v}$ ::=	...	*values:*
	$\mathtt{inl\ v\ as\ T}$	*tagged value (left)*
	$\mathtt{inr\ v\ as\ T}$	*tagged value (right)*

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

$$\begin{array}{l}\mathtt{case\ (inl\ v_0\ as\ T_0)}\\ \mathtt{of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2}\\ \qquad \longrightarrow [\mathtt{x}_1 \mapsto \mathtt{v}_0]\mathtt{t}_1\end{array} \qquad \text{(E-CaseInl)}$$

$$\begin{array}{l}\mathtt{case\ (inr\ v_0\ as\ T_0)}\\ \mathtt{of\ inl\ x_1 \Rightarrow t_1\ |\ inr\ x_2 \Rightarrow t_2}\\ \qquad \longrightarrow [\mathtt{x}_2 \mapsto \mathtt{v}_0]\mathtt{t}_2\end{array} \qquad \text{(E-CaseInr)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{inl\ t_1\ as\ T_2} \longrightarrow \mathtt{inl\ t_1'\ as\ T_2}} \qquad \text{(E-Inl)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{inr\ t_1\ as\ T_2} \longrightarrow \mathtt{inr\ t_1'\ as\ T_2}} \qquad \text{(E-Inr)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \vdash \mathtt{inl\ t_1\ as\ T_1+T_2} : \mathtt{T}_1\mathtt{+T}_2} \qquad \text{(T-Inl)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_2}{\Gamma \vdash \mathtt{inr\ t_1\ as\ T_1+T_2} : \mathtt{T}_1\mathtt{+T}_2} \qquad \text{(T-Inr)}$$

Variants

将sum types加上标签，可以泛化为variants

$T_1 + T_2 \Longrightarrow < l_1 : T_1, l_2 : T_2 >$

$inl\ t\ as\ T_1 + T_2 \Longrightarrow < l_1 = t > as < l_1 : T_1, l_2 : T_2 >$

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

New syntactic forms

$$\begin{array}{llll} \texttt{t} & ::= & \ldots & \textit{terms:} \\ & & \texttt{<l=t> as T} & \textit{tagging} \\ & & \texttt{case t of <l}_i\texttt{=x}_i\texttt{>}\Rightarrow\texttt{t}_i\ ^{i\in 1..n} & \textit{case} \end{array}$$

$$\begin{array}{llll} \texttt{T} & ::= & \ldots & \textit{types:} \\ & & \texttt{<l}_i\texttt{:T}_i\ ^{i\in 1..n}\texttt{>} & \textit{type of variants} \end{array}$$

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

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

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

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

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

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

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

链表

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

New syntactic forms

$\mathtt{t} ::= \ldots$		*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*
$\mathtt{v} ::= \ldots$		*values:*
	`nil[T]`	*empty list*
	`cons[T] v v`	*list constructor*
$\mathtt{T} ::= \ldots$		*types:*
	`List T`	*type of lists*

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

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{cons[T]}\ \mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{cons[T]}\ \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-Cons1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{cons[T]}\ \mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{cons[T]}\ \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-Cons2)}$$

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

$$\mathtt{isnil[S]}\ (\mathtt{cons[T]}\ \mathtt{v}_1\ \mathtt{v}_2) \longrightarrow \mathtt{false} \quad \text{(E-IsnilCons)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{isnil[T]}\ \mathtt{t}_1 \longrightarrow \mathtt{isnil[T]}\ \mathtt{t}_1'} \quad \text{(E-Isnil)}$$

$$\mathtt{head[S]}\ (\mathtt{cons[T]}\ \mathtt{v}_1\ \mathtt{v}_2) \longrightarrow \mathtt{v}_1 \quad \text{(E-HeadCons)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{head[T]}\ \mathtt{t}_1 \longrightarrow \mathtt{head[T]}\ \mathtt{t}_1'} \quad \text{(E-Head)}$$

$$\mathtt{tail[S]}\ (\mathtt{cons[T]}\ \mathtt{v}_1\ \mathtt{v}_2) \longrightarrow \mathtt{v}_2 \quad \text{(E-TailCons)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{tail[T]}\ \mathtt{t}_1 \longrightarrow \mathtt{tail[T]}\ \mathtt{t}_1'} \quad \text{(E-Tail)}$$

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

$$\Gamma \vdash \mathtt{nil}\ [\mathtt{T}_1] : \mathtt{List}\ \mathtt{T}_1 \quad \text{(T-Nil)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{List}\ \mathtt{T}_1}{\Gamma \vdash \mathtt{cons}[\mathtt{T}_1]\ \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{List}\ \mathtt{T}_1} \quad \text{(T-Cons)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{List}\ \mathtt{T}_{11}}{\Gamma \vdash \mathtt{isnil}[\mathtt{T}_{11}]\ \mathtt{t}_1 : \mathtt{Bool}} \quad \text{(T-Isnil)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{List}\ \mathtt{T}_{11}}{\Gamma \vdash \mathtt{head}[\mathtt{T}_{11}]\ \mathtt{t}_1 : \mathtt{T}_{11}} \quad \text{(T-Head)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{List}\ \mathtt{T}_{11}}{\Gamma \vdash \mathtt{tail}[\mathtt{T}_{11}]\ \mathtt{t}_1 : \mathtt{List}\ \mathtt{T}_{11}} \quad \text{(T-Tail)}$$

引用

$\rightarrow$ `Unit` `Ref` *Extends* $\lambda_{\rightarrow}$ *with* `Unit` *(9-1 and 11-2)*

Syntax

`t ::=`		*terms:*
	`x`	*variable*
	`λx:T.t`	*abstraction*
	`t t`	*application*
	`unit`	*constant* `unit`
	`ref t`	*reference creation*
	`!t`	*dereference*
	`t:=t`	*assignment*
	l	*store location*
`v ::=`		*values:*
	`λx:T.t`	*abstraction value*
	`unit`	*constant* `unit`
	l	*store location*
`T ::=`		*types:*
	`T→T`	*type of functions*
	`Unit`	*unit type*
	`Ref T`	*type of reference cells*
`Γ ::=`		*contexts:*
	$\varnothing$	*empty context*
	`Γ, x:T`	*term variable binding*
μ `::=`		*stores:*
	$\varnothing$	*empty store*
	$\mu, l = \mathtt{v}$	*location binding*
Σ `::=`		*store typings:*
	$\varnothing$	*empty store typing*
	$\Sigma, l\mathtt{:T}$	*location typing*

Evaluation $\boxed{\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'}$

$$\frac{\mathtt{t_1} \mid \mu \longrightarrow \mathtt{t_1'} \mid \mu'}{\mathtt{t_1\ t_2} \mid \mu \longrightarrow \mathtt{t_1'\ t_2} \mid \mu'} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t_2} \mid \mu \longrightarrow \mathtt{t_2'} \mid \mu'}{\mathtt{v_1\ t_2} \mid \mu \longrightarrow \mathtt{v_1\ t_2'} \mid \mu'} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x:T_{11}.t_{12}})\ \mathtt{v_2} \mid \mu \longrightarrow [\mathtt{x} \mapsto \mathtt{v_2}]\mathtt{t_{12}} \mid \mu \quad \text{(E-AppAbs)}$$

$$\frac{l \notin dom(\mu)}{\mathtt{ref\ v_1} \mid \mu \longrightarrow l \mid (\mu, l \mapsto \mathtt{v_1})} \quad \text{(E-RefV)}$$

$$\frac{\mathtt{t_1} \mid \mu \longrightarrow \mathtt{t_1'} \mid \mu'}{\mathtt{ref\ t_1} \mid \mu \longrightarrow \mathtt{ref\ t_1'} \mid \mu'} \quad \text{(E-Ref)}$$

$$\frac{\mu(l) = \mathtt{v}}{!l \mid \mu \longrightarrow \mathtt{v} \mid \mu} \quad \text{(E-DerefLoc)}$$

$$\frac{\mathtt{t_1} \mid \mu \longrightarrow \mathtt{t_1'} \mid \mu'}{\mathtt{!t_1} \mid \mu \longrightarrow \mathtt{!t_1'} \mid \mu'} \quad \text{(E-Deref)}$$

$$l\mathtt{:=v_2} \mid \mu \longrightarrow \mathtt{unit} \mid [l \mapsto \mathtt{v_2}]\mu \quad \text{(E-Assign)}$$

$$\frac{\mathtt{t_1} \mid \mu \longrightarrow \mathtt{t_1'} \mid \mu'}{\mathtt{t_1:=t_2} \mid \mu \longrightarrow \mathtt{t_1':=t_2} \mid \mu'} \quad \text{(E-Assign1)}$$

$$\frac{\mathtt{t_2} \mid \mu \longrightarrow \mathtt{t_2'} \mid \mu'}{\mathtt{v_1:=t_2} \mid \mu \longrightarrow \mathtt{v_1:=t_2'} \mid \mu'} \quad \text{(E-Assign2)}$$

continued…

Typing $\boxed{\Gamma \mid \Sigma \vdash \mathtt{t : T}}$

$$\frac{\mathtt{x:T} \in \Gamma}{\Gamma \mid \Sigma \vdash \mathtt{x : T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x:T_1} \mid \Sigma \vdash \mathtt{t_2 : T_2}}{\Gamma \mid \Sigma \vdash \lambda\mathtt{x:T_1.t_2 : T_1 \rightarrow T_2}} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t_1 : T_{11} \rightarrow T_{12}} \qquad \Gamma \mid \Sigma \vdash \mathtt{t_2 : T_{11}}}{\Gamma \mid \Sigma \vdash \mathtt{t_1\ t_2 : T_{12}}} \quad \text{(T-App)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{unit : Unit} \quad \text{(T-Unit)}$$

$$\frac{\Sigma(l) = \mathtt{T_1}}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref\ T_1}} \quad \text{(T-Loc)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t_1 : T_1}}{\Gamma \mid \Sigma \vdash \mathtt{ref\ t_1 : Ref\ T_1}} \quad \text{(T-Ref)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t_1 : Ref\ T_{11}}}{\Gamma \mid \Sigma \vdash \mathtt{!t_1 : T_{11}}} \quad \text{(T-Deref)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t_1 : Ref\ T_{11}} \qquad \Gamma \mid \Sigma \vdash \mathtt{t_2 : T_{11}}}{\Gamma \mid \Sigma \vdash \mathtt{t_1:=t_2 : Unit}} \quad \text{(T-Assign)}$$

异常处理

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

New syntactic forms

$\mathtt{t} ::= \ldots$		*terms:*
	$\mathtt{raise}\ \mathtt{t}$	*raise exception*
	$\mathtt{try}\ \mathtt{t}\ \mathtt{with}\ \mathtt{t}$	*handle exceptions*

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

$$(\mathtt{raise}\ \mathtt{v}_{11})\ \mathtt{t}_2 \longrightarrow \mathtt{raise}\ \mathtt{v}_{11} \qquad \text{(E-AppRaise1)}$$

$$\mathtt{v}_1\ (\mathtt{raise}\ \mathtt{v}_{21}) \longrightarrow \mathtt{raise}\ \mathtt{v}_{21} \qquad \text{(E-AppRaise2)}$$

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

$$\mathtt{raise}\ (\mathtt{raise}\ \mathtt{v}_{11}) \longrightarrow \mathtt{raise}\ \mathtt{v}_{11} \qquad \text{(E-RaiseRaise)}$$

$$\mathtt{try}\ \mathtt{v}_1\ \mathtt{with}\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1 \qquad \text{(E-TryV)}$$

$$\mathtt{try}\ \mathtt{raise}\ \mathtt{v}_{11}\ \mathtt{with}\ \mathtt{t}_2 \longrightarrow \mathtt{t}_2\ \mathtt{v}_{11} \qquad \text{(E-TryRaise)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{try}\ \mathtt{t}_1\ \mathtt{with}\ \mathtt{t}_2 \longrightarrow \mathtt{try}\ \mathtt{t}_1'\ \mathtt{with}\ \mathtt{t}_2} \qquad \text{(E-Try)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{exn}}{\Gamma \vdash \mathtt{raise}\ \mathtt{t}_1 : \mathtt{T}} \qquad \text{(T-Exn)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{exn} \rightarrow \mathtt{T}}{\Gamma \vdash \mathtt{try}\ \mathtt{t}_1\ \mathtt{with}\ \mathtt{t}_2 : \mathtt{T}} \qquad \text{(T-Try)}$$

子类型

性质

引理：

- 如果 $S <: T_1 \rightarrow T_2$，那么 S 具有 $S_1 \rightarrow S_2$ 的形式，其中 $T_1 <: S_1$，$S_2 <: T_2$ 。
- 如果 $S <: \{l_i : T_i^{(i\in 1..n)}\}$，那么S具有 $\{k_j : S_j^{(j\in 1..m)}\}$ 的形式，其中 $\{l_i^{(i\in 1..n)}\} \subseteq \{k_j^{(j\in 1..m)}\}$ ，且对每个共同的标签 $l_i = k_j$ 都有 $S_j <: T_i$。

递归类型

利用运算符μ表示递归与重复的部分

【例】

- $NatList = \mu X. < nil : Unit, const\ \{Nat, X\} >;$
- 令 NatList 为一个无穷类型，满足方程 $X =< nil : Unit, const\ \{Nat, X\} >$

```
sumlist = fix (λs:NatList→Nat. λl:NatList.
                if isnil l then 0 else plus (hd l) (s (tl l)));
▸ sumlist : NatList → Nat
```

【例】

一个hungry函数，接收一个参数，然后返回一个函数，返回的函数仍然是一个hungry函数：

$Hungry = \mu A. Nat \rightarrow A;$

```
f = fix (λf: Nat→Hungry. λn:Nat. f);
▸ f : Hungry

f 0 1 2 3 4 5;
▸ <fun> : Hungry
```

【例】

$Stream = \mu A.Unit \to \{Nat, A\};$

```
hd = λs:Stream. (s unit).1;
▸ hd : Stream → Nat

tl = λs:Stream. (s unit).2;
▸ tl : Stream → Stream

upfrom0 = fix (λf: Nat→Stream. λn:Nat. λ_:Unit. {n,f (succ n)}) 0;
▸ upfrom0 : Stream

hd upfrom0;
▸ 0 : Nat

hd (tl (tl (tl upfrom0)));
▸ 3 : Nat
```

- Streams可以用来表示进程
- $Process = \mu A.Nat \to \{Nat, A\};$

```
p = fix (λf: Nat→Process. λacc:Nat. λn:Nat.
              let newacc = plus acc n in
              {newacc, f newacc}) 0;
▸ p : Process

curr = λs:Process. (s 0).1;
▸ curr : Process → Nat

send = λn:Nat. λs:Process. (s n).2;
▸ send : Nat → Process → Process
```

- 对象
  - 数据及数据上的操作
  - 纯函数式

```
Counter = μC. {get:Nat, inc:Unit→C, dec:Unit→C};

c = let create = fix (λf: {x:Nat}→Counter. λs: {x:Nat}.
                       {get = s.x,
                        inc = λ_:Unit. f {x=succ(s.x)},
                        dec = λ_:Unit. f {x=pred(s.x)} })
    in create {x=0};

▸ c : Counter

c1 = c.inc unit;
c2 = c1.inc unit;
c2.get;

▸ 2 : Nat
```

形式化定义

- 增加一对函数unfold和fold

$$
\begin{aligned}
\mathtt{unfold}[\mu X.T] &\quad : \quad \mu X.T \to [X \mapsto \mu X.T]T \\
\mathtt{fold}[\mu X.T] &\quad : \quad [X \mapsto \mu X.T]T \to \mu X.T
\end{aligned}
$$

$$
\begin{tikzcd}
\mu X.T \arrow[rr, bend left, "{\mathtt{unfold}[\mu X.T]}"] & & {[X \mapsto \mu X.T]T} \arrow[ll, bend left, "{\mathtt{fold}[\mu X.T]}"]
\end{tikzcd}
$$

【例】

$\mu X.\mathtt{<nil:Unit,cons:\{Nat,X\}>}$,

unfolds to

$\mathtt{<nil:Unit,\ cons:\{Nat,\ \mu X.<nil:Unit,cons:\{Nat,X\}>\}>}$.

多态

类型变量与替换。为了实现多态，可以使用一些变量作为占位符，需要的时候再替换成实际的类型。

类型的替换由两部分组成：

- 声明一个映射σ，将类型变量映射到具体的类型

- 应用一次映射T，得到一个实例σ T

【例】

$\sigma = [X \mapsto Bool, Y \mapsto Nat, Z \mapsto Nat \rightarrow Bool]$

$\sigma(X \rightarrow X) = Bool \rightarrow Bool$

System F

```
double = λX. λf:X→X. λa:X. f (f a);
▸ double : ∀X. (X→X) → X → X
```

```
doubleNat = double [Nat];
▸ doubleNat : (Nat→Nat) → Nat → Nat
doubleNatArrowNat = double [Nat→Nat];
▸ doubleNatArrowNat : ((Nat→Nat)→Nat→Nat) →
                        (Nat→Nat) → Nat → Nat
```

```
double [Nat] (λx:Nat. succ(succ(x))) 3;
▸ 7 : Nat
```

【例】多态链表

```
▸ nil : ∀X. List X
  cons : ∀X. X → List X → List X
  isnil : ∀X. List X → Bool
  head : ∀X. List X → X
  tail : ∀X. List X → List X
```

```
map = λX. λY.
        λf: X→Y.
          (fix (λm: (List X) → (List Y).
                  λl: List X.
                    if isnil [X] l
                      then nil [Y]
                      else cons [Y] (f (head [X] l))
                                    (m (tail [X] l))));
▸ map : ∀X. ∀Y. (X→Y) → List X → List Y

l = cons [Nat] 4 (cons [Nat] 3 (cons [Nat] 2 (nil [Nat])));
▸ l : List Nat

head [Nat] (map [Nat] [Nat] (λx:Nat. succ x) l);
▸ 5 : Nat
```

【例】二叉树

```
Tree = μX. <leaf:Unit, node:{Nat,X,X}>;
leaf = <leaf=unit> as Tree;

▸ leaf : Tree

node = λn:Nat. λt₁:Tree. λt₂:Tree. <node={n,t₁,t₂}> as Tree;

▸ node : Nat → Tree → Tree → Tree

isleaf = λl:Tree. case l of <leaf=u> ⇒ true | <node=p> ⇒ false;

▸ isleaf : Tree → Bool

label = λl:Tree. case l of <leaf=u> ⇒ 0 | <node=p> ⇒ p.1;

▸ label : Tree → Nat

left = λl:Tree. case l of <leaf=u> ⇒ leaf | <node=p> ⇒ p.2;

▸ left : Tree → Tree

right = λl:Tree. case l of <leaf=u> ⇒ leaf | <node=p> ⇒ p.3;

▸ right : Tree → Tree

append = fix (λf:NatList→NatList→NatList.
                λl1:NatList. λl2:NatList.
                if isnil l1 then l2 else
                cons (hd l1) (f (tl l1) l2));
```

```
▸ append : NatList → NatList → NatList

  preorder = fix (λf:Tree→NatList. λt:Tree.
                   if isleaf t then nil else
                   cons (label t)
                        (append (f (left t)) (f (right t))));

▸ preorder : Tree → NatList

  t1 = node 1 leaf leaf;
  t2 = node 2 leaf leaf;
  t3 = node 3 t1 t2;
  t4 = node 4 t3 t3;
  l = preorder t4;
  hd l;

▸ 4 : Nat

  hd (tl l);

▸ 3 : Nat

  hd (tl (tl l));

▸ 1 : Nat
```

求值规则

$\mathbb{B}$ *(untyped)*

Syntax

t ::=		*terms:*
	true	*constant true*
	false	*constant false*
	if t then t else t	*conditional*
v ::=		*values:*
	true	*true value*
	false	*false value*

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

$$\texttt{if true then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \longrightarrow \mathtt{t}_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \longrightarrow \mathtt{t}_3 \qquad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\texttt{if } \mathtt{t}_1 \texttt{ then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \longrightarrow \texttt{if } \mathtt{t}_1' \texttt{ then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3} \qquad \text{(E-If)}$$

$\mathbb{B}$ $\mathbb{N}$ *(untyped)* — *Extends* **B** *(3-1)*

New syntactic forms

t ::=	...	*terms:*
	0	*constant zero*
	succ t	*successor*
	pred t	*predecessor*
	iszero t	*zero test*
v ::=	...	*values:*
	nv	*numeric value*
nv ::=		*numeric values:*
	0	*zero value*
	succ nv	*successor value*

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{succ}\ t_1 \longrightarrow \texttt{succ}\ t_1'} \quad \text{(E-Succ)}$$

$$\texttt{pred}\ 0 \longrightarrow 0 \quad \text{(E-PredZero)}$$

$$\texttt{pred}\ (\texttt{succ}\ nv_1) \longrightarrow nv_1 \quad \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{pred}\ t_1 \longrightarrow \texttt{pred}\ t_1'} \quad \text{(E-Pred)}$$

$$\texttt{iszero}\ 0 \longrightarrow \texttt{true} \quad \text{(E-IszeroZero)}$$

$$\texttt{iszero}\ (\texttt{succ}\ nv_1) \longrightarrow \texttt{false} \quad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{iszero}\ t_1 \longrightarrow \texttt{iszero}\ t_1'} \quad \text{(E-IsZero)}$$

$\rightarrow$ *(untyped)*

Syntax

t ::=		*terms:*
	x	*variable*
	λx.t	*abstraction*
	t t	*application*
v ::=		*values:*
	λx.t	*abstraction value*

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

$\mathbb{B}$ *(typed)* — *Extends* **B** *(3-1)*

New syntactic forms

T ::=		*types:*
	Bool	*type of booleans*

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

$$\texttt{true} : \texttt{Bool} \quad \text{(T-True)}$$

$$\texttt{false} : \texttt{Bool} \quad \text{(T-False)}$$

$$\frac{t_1 : \texttt{Bool} \qquad t_2 : T \qquad t_3 : T}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T} \quad \text{(T-If)}$$

$\mathbb{B}$ $\mathbb{N}$ *(typed)* *Extends* **NB** *(3-2) and 8-1*

New syntactic forms

$\mathtt{T}$ ::= ... *types:*

$\mathtt{Nat}$ *type of natural numbers*

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

$$\mathtt{0 : Nat} \quad \text{(T-Zero)}$$

$$\frac{\mathtt{t_1 : Nat}}{\mathtt{succ\ t_1 : Nat}} \quad \text{(T-Succ)}$$

$$\frac{\mathtt{t_1 : Nat}}{\mathtt{pred\ t_1 : Nat}} \quad \text{(T-Pred)}$$

$$\frac{\mathtt{t_1 : Nat}}{\mathtt{iszero\ t_1 : Bool}} \quad \text{(T-IsZero)}$$

$\to$ *(typed)* *Based on* λ *(5-3)*

Syntax

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

$\mathtt{x}$ *variable*

$\lambda\mathtt{x{:}T.t}$ *abstraction*

$\mathtt{t\ t}$ *application*

$\mathtt{v}$ ::= *values:*

$\lambda\mathtt{x{:}T.t}$ *abstraction value*

$\mathtt{T}$ ::= *types:*

$\mathtt{T{\to}T}$ *type of functions*

Γ ::= *contexts:*

$\varnothing$ *empty context*

$\Gamma, \mathtt{x{:}T}$ *term variable binding*

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

$$\frac{\mathtt{t_1 \longrightarrow t_1'}}{\mathtt{t_1\ t_2 \longrightarrow t_1'\ t_2}} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t_2 \longrightarrow t_2'}}{\mathtt{v_1\ t_2 \longrightarrow v_1\ t_2'}} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x{:}T_{11}.t_{12}})\ \mathtt{v_2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v_2}]\mathtt{t_{12}} \quad \text{(E-AppAbs)}$$

Typing $\boxed{\Gamma \vdash \mathtt{t : T}}$

$$\frac{\mathtt{x{:}T} \in \Gamma}{\Gamma \vdash \mathtt{x : T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x{:}T_1} \vdash \mathtt{t_2 : T_2}}{\Gamma \vdash \lambda\mathtt{x{:}T_1.t_2 : T_1{\to}T_2}} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t_1 : T_{11}{\to}T_{12}} \qquad \Gamma \vdash \mathtt{t_2 : T_{11}}}{\Gamma \vdash \mathtt{t_1\ t_2 : T_{12}}} \quad \text{(T-App)}$$

子类型

$\rightarrow$ $<:$ Top *Based on $\lambda_{\rightarrow}$ (9-1)*

Syntax

t ::=		*terms:*
	x	*variable*
	$\lambda x{:}T.t$	*abstraction*
	$t\ t$	*application*
v ::=		*values:*
	$\lambda x{:}T.t$	*abstraction value*
T ::=		*types:*
	$\boxed{Top}$	*maximum type*
	$T{\rightarrow}T$	*type of functions*
Γ ::=		*contexts:*
	$\varnothing$	*empty context*
	$\Gamma, x{:}T$	*term variable binding*

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Subtyping $\boxed{S <: T}$

$$S <: S \quad \text{(S-Refl)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \quad \text{(S-Trans)}$$

$$S <: Top \quad \text{(S-Top)}$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1{\rightarrow}S_2 <: T_1{\rightarrow}T_2} \quad \text{(S-Arrow)}$$

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

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1{\rightarrow}T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11}{\rightarrow}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \quad \text{(T-Sub)}$$

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

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

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

$\rightarrow$ {} $<:$ *Extends $\lambda_{<:}$ (15-1) and simple record rules (15-2)*

New subtyping rules $\boxed{S <: T}$

$$\{l_i{:}T_i\ ^{i\in 1..n+k}\} <: \{l_i{:}T_i\ ^{i\in 1..n}\} \quad \text{(S-RcdWidth)}$$

$$\frac{\text{for each } i \quad S_i <: T_i}{\{l_i{:}S_i\ ^{i\in 1..n}\} <: \{l_i{:}T_i\ ^{i\in 1..n}\}} \quad \text{(S-RcdDepth)}$$

$$\frac{\{k_j{:}S_j\ ^{j\in 1..n}\} \text{ is a permutation of } \{l_i{:}T_i\ ^{i\in 1..n}\}}{\{k_j{:}S_j\ ^{j\in 1..n}\} <: \{l_i{:}T_i\ ^{i\in 1..n}\}} \quad \text{(S-RcdPerm)}$$

Iso-recursive

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

$$
\begin{array}{llll}
\texttt{t} & ::= & \ldots & \textit{terms:} \\
 & & \texttt{fold [T] t} & \textit{folding} \\
 & & \texttt{unfold [T] t} & \textit{unfolding} \\
\texttt{v} & ::= & \ldots & \textit{values:} \\
 & & \texttt{fold [T] v} & \textit{folding} \\
\texttt{T} & ::= & \ldots & \textit{types:} \\
 & & \texttt{X} & \textit{type variable} \\
 & & \mu\texttt{X.T} & \textit{recursive type}
\end{array}
$$

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

$$\texttt{unfold [S] (fold [T] } \texttt{v}_1\texttt{)} \longrightarrow \texttt{v}_1 \qquad \text{(E-UNFLDFLD)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{fold [T] } \texttt{t}_1 \longrightarrow \texttt{fold [T] } \texttt{t}_1'} \qquad \text{(E-FLD)}$$

$$\frac{\texttt{t}_1 \longrightarrow \texttt{t}_1'}{\texttt{unfold [T] } \texttt{t}_1 \longrightarrow \texttt{unfold [T] } \texttt{t}_1'} \qquad \text{(E-UNFLD)}$$

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

$$\frac{\texttt{U} = \mu\texttt{X.T}_1 \qquad \Gamma \vdash \texttt{t}_1 : [\texttt{X} \mapsto \texttt{U}]\texttt{T}_1}{\Gamma \vdash \texttt{fold [U] } \texttt{t}_1 : \texttt{U}} \qquad \text{(T-FLD)}$$

$$\frac{\texttt{U} = \mu\texttt{X.T}_1 \qquad \Gamma \vdash \texttt{t}_1 : \texttt{U}}{\Gamma \vdash \texttt{unfold [U] } \texttt{t}_1 : [\texttt{X} \mapsto \texttt{U}]\texttt{T}_1} \qquad \text{(T-UNFLD)}$$

约束求解

$$\frac{\texttt{x:T} \in \Gamma}{\Gamma \vdash \texttt{x} : \texttt{T} \ |_{\varnothing}\ \{\}} \qquad \text{(CT-VAR)}$$

$$\frac{\Gamma, \texttt{x:T}_1 \vdash \texttt{t}_2 : \texttt{T}_2 \ |_{\mathcal{X}}\ C}{\Gamma \vdash \lambda\texttt{x:T}_1\texttt{.t}_2 : \texttt{T}_1 \to \texttt{T}_2 \ |_{\mathcal{X}}\ C} \qquad \text{(CT-ABS)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \ |_{\mathcal{X}_1}\ C_1 \qquad \Gamma \vdash \texttt{t}_2 : \texttt{T}_2 \ |_{\mathcal{X}_2}\ C_2 \\ \mathcal{X}_1 \cap \mathcal{X}_2 = \mathcal{X}_1 \cap FV(\texttt{T}_2) = \mathcal{X}_2 \cap FV(\texttt{T}_1) = \varnothing \\ \texttt{X} \notin \mathcal{X}_1, \mathcal{X}_2, \texttt{T}_1, \texttt{T}_2, C_1, C_2, \Gamma, \texttt{t}_1, \text{or } \texttt{t}_2 \\ C' = C_1 \cup C_2 \cup \{\texttt{T}_1 = \texttt{T}_2 \to \texttt{X}\}\end{array}}{\Gamma \vdash \texttt{t}_1\ \texttt{t}_2 : \texttt{X} \ |_{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \{\texttt{X}\}}\ C'} \qquad \text{(CT-APP)}$$

$$\Gamma \vdash \texttt{0} : \texttt{Nat} \ |_{\varnothing}\ \{\} \qquad \text{(CT-ZERO)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T} \ |_{\mathcal{X}}\ C \qquad C' = C \cup \{\texttt{T} = \texttt{Nat}\}}{\Gamma \vdash \texttt{succ } \texttt{t}_1 : \texttt{Nat} \ |_{\mathcal{X}}\ C'} \qquad \text{(CT-SUCC)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T} \ |_{\mathcal{X}}\ C \qquad C' = C \cup \{\texttt{T} = \texttt{Nat}\}}{\Gamma \vdash \texttt{pred } \texttt{t}_1 : \texttt{Nat} \ |_{\mathcal{X}}\ C'} \qquad \text{(CT-PRED)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T} \ |_{\mathcal{X}}\ C \qquad C' = C \cup \{\texttt{T} = \texttt{Nat}\}}{\Gamma \vdash \texttt{iszero } \texttt{t}_1 : \texttt{Bool} \ |_{\mathcal{X}}\ C'} \qquad \text{(CT-ISZERO)}$$

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \ |_{\varnothing}\ \{\} \qquad \text{(CT-TRUE)}$$

$$\Gamma \vdash \texttt{false} : \texttt{Bool} \ |_{\varnothing}\ \{\} \qquad \text{(CT-FALSE)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \ |_{\mathcal{X}_1}\ C_1 \\ \Gamma \vdash \texttt{t}_2 : \texttt{T}_2 \ |_{\mathcal{X}_2}\ C_2 \qquad \Gamma \vdash \texttt{t}_3 : \texttt{T}_3 \ |_{\mathcal{X}_3}\ C_3 \\ \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \text{ nonoverlapping} \\ C' = C_1 \cup C_2 \cup C_3 \cup \{\texttt{T}_1 = \texttt{Bool}, \texttt{T}_2 = \texttt{T}_3\}\end{array}}{\Gamma \vdash \texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}_2 \ |_{\mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3}\ C'} \qquad \text{(CT-IF)}$$

$$
\begin{array}{lll}
\mathit{unify}(C) & = & \text{if } C = \varnothing, \text{ then } [\,] \\
 & & \text{else let } \{\texttt{S} = \texttt{T}\} \cup C' = C \text{ in} \\
 & & \quad \text{if } \texttt{S} = \texttt{T} \\
 & & \quad\quad \text{then } \mathit{unify}(C') \\
 & & \quad \text{else if } \texttt{S} = \texttt{X} \text{ and } \texttt{X} \notin FV(\texttt{T}) \\
 & & \quad\quad \text{then } \mathit{unify}([\texttt{X} \mapsto \texttt{T}]C') \circ [\texttt{X} \mapsto \texttt{T}] \\
 & & \quad \text{else if } \texttt{T} = \texttt{X} \text{ and } \texttt{X} \notin FV(\texttt{S}) \\
 & & \quad\quad \text{then } \mathit{unify}([\texttt{X} \mapsto \texttt{S}]C') \circ [\texttt{X} \mapsto \texttt{S}] \\
 & & \quad \text{else if } \texttt{S} = \texttt{S}_1 \to \texttt{S}_2 \text{ and } \texttt{T} = \texttt{T}_1 \to \texttt{T}_2 \\
 & & \quad\quad \text{then } \mathit{unify}(C' \cup \{\texttt{S}_1 = \texttt{T}_1, \texttt{S}_2 = \texttt{T}_2\}) \\
 & & \quad \text{else} \\
 & & \quad\quad \mathit{fail}
\end{array}
$$

→ ∀ *Based on* $\lambda_{\to}$ *(9-1)*

Syntax

t ::=		*terms:*
	x	*variable*
	λx:T.t	*abstraction*
	t t	*application*
	λX.t	*type abstraction*
	t [T]	*type application*
v ::=		*values:*
	λx:T.t	*abstraction value*
	λX.t	*type abstraction value*
T ::=		*types:*
	X	*type variable*
	T→T	*type of functions*
	∀X.T	*universal type*
Γ ::=		*contexts:*
	∅	*empty context*
	Γ, x:T	*term variable binding*
	Γ, X	*type variable binding*

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ [T_2] \longrightarrow t_1'\ [T_2]} \quad \text{(E-TApp)}$$

$$(\lambda X.t_{12})\ [T_2] \longrightarrow [X \mapsto T_2]t_{12} \quad \text{(E-TappTabs)}$$

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

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1 \to T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \lambda X.t_2 : \forall X.T_2} \quad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash t_1 : \forall X.T_{12}}{\Gamma \vdash t_1\ [T_2] : [X \mapsto T_2]T_{12}} \quad \text{(T-TApp)}$$