



# Type Systems

Lecture 4 Nov. 10th, 2004

Sebastian Maneth

<http://lampwww.epfl.ch/teaching/typeSystems/2004>



# Today: ... simple language extensions ...

1. Derived Forms
2. Labeled Records
3. Labeled Variants
4. Lists
5. Normalization

# 1. Derived Forms

**Idea** Give *more freedom* to the programmer by introducing **new syntactic forms  $f$**  to the surface language L.

If

- A. the **evaluation behavior** and
- B. the **typing behavior**

of  $f$  can be derived from those of L,  
then  $f$  is a **derived form of L**.

Derived forms give *more freedom* to the language designer, because the complexity of the internal language does not change.

→ type safety (**progress+preservation**) need NOT be reproved!

# 1. Derived Forms

**Example** Sequencing.

First, add new type **Unit** with unique constant value `uni t`, and

typing rule  $\Gamma \vdash \text{uni } t : \text{Unit}$

→ similar to `void` in languages like C or Java.

→ useful if we care about *side effects*, not result.

---

Sequencing:  $t_1 ; t_2$  = “evaluate  $t_1$ , throw away its trivial result, then evaluate  $t_2$ .”

Possible evaluation / typing rules

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2}$$

$$\text{uni } t ; t_2 \rightarrow t_2$$

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

# 1. Derived Forms

**Example** Sequencing.

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

- similar to `let` / application of an abstraction
- is there a lambda term with same typing??

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \mathbf{uni} \ t ; t_2 \rightarrow t_2$$

# 1. Derived Forms

**Example** Sequencing.

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

- similar to `let` / application of an abstraction
- is there a lambda term with same typing??

**YES!** Define  $t_1 ; t_2 := (\lambda x : \mathbf{Unit}. t_2) t_1$   $x \notin \text{FV}(t_2)$ , fresh!

---

$$\Gamma \vdash (\lambda x : \mathbf{Unit}. t_2) t_1 : T_2$$

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \mathbf{uni} \ t ; t_2 \rightarrow t_2$$

# 1. Derived Forms

**Example** Sequencing.

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

- similar to `let` / application of an abstraction
- is there a lambda term with same typing??

**YES!** Define  $t_1 ; t_2 := (\lambda x:\mathbf{Unit}. t_2) t_1$   $x \notin \text{FV}(t_2)$ , fresh!

$$\frac{\Gamma \vdash (\lambda x:\mathbf{Unit}. t_2) : \mathbf{Unit} \rightarrow T_2 \quad \Gamma \vdash t_1 : \mathbf{Unit}}{\Gamma \vdash (\lambda x:\mathbf{Unit}. t_2) t_1 : T_2}$$

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \text{uni } t ; t_2 \rightarrow t_2$$

# 1. Derived Forms

**Example** Sequencing.

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

- similar to `let` / application of an abstraction
- is there a lambda term with same typing??

**YES!** Define  $t_1 ; t_2 := (\lambda x:\mathbf{Unit}. t_2) t_1$   $x \notin \text{FV}(t_2)$ , fresh!

$$\frac{\frac{\Gamma, x:\mathbf{Unit} \vdash t_2 : T_2}{\Gamma \vdash (\lambda x:\mathbf{Unit}. t_2) : \mathbf{Unit} \rightarrow T_2} \quad \Gamma \vdash t_1 : \mathbf{Unit}}{\Gamma \vdash (\lambda x:\mathbf{Unit}. t_2) t_1 : T_2}$$

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \text{uni } t ; t_2 \rightarrow t_2$$



# 1. Derived Forms

**Example** Sequencing.

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

- similar to `let` / application of an abstraction
- is there a lambda term with same typing??

**YES!** Define  $t_1 ; t_2 := (\lambda x : \mathbf{Unit}. t_2) t_1$   $x \notin \text{FV}(t_2)$ , fresh!

$$\frac{\frac{\Gamma \vdash t_2 : T_2}{\Gamma, x : \mathbf{Unit} \vdash t_2 : T_2} \quad x \notin \text{FV}(t_2)}{\Gamma \vdash (\lambda x : \mathbf{Unit}. t_2) : \mathbf{Unit} \rightarrow T_2} \quad \Gamma \vdash t_1 : \mathbf{Unit}}{\Gamma \vdash (\lambda x : \mathbf{Unit}. t_2) t_1 : T_2}$$

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \text{uni } t ; t_2 \rightarrow t_2$$



# 1. Derived Forms

**Example** Sequencing.

Questions:

1. Can you prove that `;` is a **derived form** (= **A.** and **B.**)
2. Is `let` a derived form?

$$\frac{t_1 \rightarrow t_1'}{t_1 ; t_2 \rightarrow t_1' ; t_2} \quad \text{uni } t ; t_2 \rightarrow t_2$$

not needed anymore!!

**A.**  $t \rightarrow_{\text{ext}} t' \text{ iff } e(t) \rightarrow_{\text{int}} e(t') \quad e: \text{ext} \rightarrow \text{int}$

**B.**  $\Gamma \vdash_{\text{ext}} t : T \text{ iff } \Gamma \vdash_{\text{int}} e(t) : T$

## 2. Labeled Records

$\{x=5\}$  record of type  $\{x:\text{Nat}\}$

$\{\text{partno}=5524, \text{available}=\text{true}\}$   
 record of type  $\{\text{partno}:\text{Nat}, \text{available}:\text{Bool}\}$

selection:  $\{x=5, y=6\}.y \rightarrow 6$

### evaluation

$\{l_1=v_1, \dots, l_n=v_n\}.l_j \rightarrow v_j$       “if everything is value, you can select”

$$\frac{t_1 \rightarrow t_1'}{t_1.l \rightarrow t_1'.l}$$

“evaluate inside of selections, ...

$$t_j \rightarrow t_j'$$

..., from left to right.”

$\{l_1=v_1, \dots, l_{j-1}=v_{j-1}, l_j=t_j, \dots, l_n=t_n\} \rightarrow$   
 $\{l_1=v_1, \dots, l_{j-1}=v_{j-1}, l_j=t_j', \dots, l_n=t_n\}$       ( $\rightarrow$  *ordered*!)

## 2. Labeled Records

{x=5} record of type {x:Nat}

{partno=5524, available=true}  
record of type {partno:Nat, available:Bool}

selection: {x=5, y=6}.y → 6

---

typing

$$\frac{\Gamma \vdash t_1 : T_1, \dots, \Gamma \vdash t_n : T_n}{\Gamma \vdash \{l_1=t_1, \dots, l_n=t_n\} : \{l_1 : T_1, \dots, l_n : T_n\}}$$

$$\frac{\Gamma \vdash t_1 : \{l_1 : T_1, \dots, l_n : T_n\}}{\Gamma \vdash t_1.l_j : T_j}$$



## 2. Labeled Records

Note: our records are *ordered* :

$\{x=5, y=6\}$  is NOT the same as  $\{y=6, x=5\}$

→  $\{x:\text{Nat}, y:\text{Nat}\} \neq \{y:\text{Nat}, x:\text{Nat}\}$

Will change in the presence of **subtyping**.

(then, one will be a subtype of the other, i.e., terms of the one type can be used in any context where terms of the other are expected)

### 3. Labeled Variants

Often programs deal with *heterogeneous collections of values*.

e.g., a node of a binary tree can be internal or a leaf  
a list can be nil (empty) or consisting of a head and tail  
etc.

variant type:

Addr = <physical : Physical Addr, virtual : Virtual Addr>

a = <physical =pa> as Addr;

variant value

→ test which “internal” type a variant value has: case

```
getName = λa: Addr. case a of
  <physical =x> → x. firstLast
  | <virtual =y> → y. name
```

### 3. Labeled Variants

(like records: *ordered*!)

evaluation:

$$\text{case } \langle l_j = v_j \rangle \text{ as } T \text{ of } \langle l_i = x_i \rangle \rightarrow t_i \quad \rightarrow \quad [x_j \rightarrow v_j] t_j$$

$$\frac{t_0 \rightarrow t_0'}{\text{case } t_0 \text{ of } \langle l_i = x_i \rangle \rightarrow t_i \quad \rightarrow \quad \text{case } t_0' \text{ of } \langle l_i = x_i \rangle \rightarrow t_i}$$

$$\frac{t_i \rightarrow t_i'}{\langle l_i = t_i \rangle \text{ as } T \quad \rightarrow \quad \langle l_i = t_i' \rangle \text{ as } T}$$

typing:

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_j = t_j \rangle \text{ as } \langle l_i : T_i \rangle : \langle l_i : T_i \rangle}$$
$$\frac{\Gamma \vdash t_0 : \langle l_i : T_i \rangle \quad \text{for each } i \quad \Gamma, x_i : T_i \vdash t_i : T}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i = x_i \rangle \rightarrow t_i : T}$$



### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Options

`OptNat` = `<none: Unit, some: Nat>`

`Table` = `Nat → OptNat`      partial functions on numbers

→ how to define the empty table?

`emptyTable` = `λn: Nat. <none=unit> as OptNat`

→ how to update (m, v) of a table?

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Options

`OptNat` = `<none: Unit, some: Nat>`

`Table` = `Nat → OptNat`      partial functions on numbers

→ how to define the empty table?

`emptyTable` = `λn: Nat. <none=unit> as OptNat`

→ how to update (m, v) of a table?

`update` = `λt: Table. λm: Nat. λv: Nat. λn: Nat`  
`if equal n m then <some=v> as OptNat`  
`else t n`

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Options

`OptNat` = `<none: Unit, some: Nat>`

`Table` = `Nat → OptNat`      partial functions on numbers

→ how to define the empty table?

`emptyTable` = `λn: Nat. <none=unit> as OptNat`

→ how to update (m, v) of a table? (type `Table → Nat → Nat → Table`)

`update` = `λt: Table. λm: Nat. λv: Nat. λn: Nat`  
`if equal n m then <some=v> as OptNat`  
`else t n`

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Options

`OptNat` = `<none: Unit, some: Nat>`

`Table` = `Nat → OptNat`      partial functions on numbers

→ table lookup: (e.g., of entry '5')

```
x = case t(5) of
  <none=u> → 0
  | <some=v> → v
```

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Enumerations

**Weekday** = <monday: Uni t, tuesday: Uni t, wednesday: Uni t, thursday: Uni t, fri day: Uni t>

function that returns the next buisness day:

```
nextBui si nessDay = λw: Weekday. case w of
  <monday=x>   → <tuesday=uni t> as Weekday
  <tuesday=x>  → <wednesday=uni t> as Weekday
  ..
  <fri day=x>  → <monday=uni t> as Weekday
```

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Single-Field Variants

dollars2euros = λd: Float. timesfloat d 0.8145

euros2dollars = λd: Float. timesfloat d 1.2277

euros2dollars(dollars2euros 39.50) → 39.4984

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Single-Field Variants

dol I ars2euros = λd: Fl oat. ti mesfl oat d 0.8145

euros2dol I ars = λd: Fl oat. ti mesfl oat d 1.2277

euros2dol I ars(dol I ars2euros 39.50) → 39.4984

But, dol I ars2euros(dol I ars2euros 39.50)

 nonsense!

### 3. Labeled Variants

Some useful variants:

- a. Options
- b. Enumerations
- c. Single-Field Variants

#### a. Single-Field Variants

`DollarAmount` = `<dollars: Float>`

`EuroAmount` = `<euros: Float>;`

`dollar2euros` = `λd: DollarAmount.`

`case d of <dollars=x> →`

`<euros=timesfloat x 0.8145> as EuroAmount`

Type of `dollar2euros`: `DollarAmount → EuroAmount`



## 4. Lists

List is a new type constructor (similar to  $\rightarrow$ )

For a type T,

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

New syntactic forms:

evaluation:

ni I [T]  
cons[T] t1 t2  
i sni I [T] t  
head[T] t  
tai I [T] t

i sni I [S] (ni I [T])  $\rightarrow$  true  
i sni I [S] (cons[T] v<sub>1</sub> v<sub>2</sub>)  $\rightarrow$  false  
head[S] (cons[T] v<sub>1</sub> v<sub>2</sub>)  $\rightarrow$  v<sub>1</sub>  
tai I [S] (cons[T] v<sub>1</sub> v<sub>2</sub>)  $\rightarrow$  v<sub>2</sub>

+ usual cbv propagation rules

## 4. Lists

typing:

$$\Gamma \vdash \text{nil} [T_1]: \text{List } T_1$$
$$\frac{\Gamma \vdash t_1: T_1 \quad \Gamma \vdash t_2: \text{List } T_1}{\Gamma \vdash \text{cons}[T_1] t_1 t_2: \text{List } T_1}$$
$$\frac{\Gamma \vdash t_1: \text{List } T_1}{\begin{array}{l} \Gamma \vdash \text{isnil} [T_1] t_1: \text{Bool} \\ \Gamma \vdash \text{head}[T_1] t_1: T_1 \\ \Gamma \vdash \text{tail} [T_1] t_1: \text{List } T_1 \end{array}}$$

---

→ can you prove the **progress theorem** for **lambda+Bool+Lists**?

→ which type annotations can be removed? which not?

## 4. Lists

→ can you prove the **progress theorem** for **lambda+Bool+Lists**?

**NO!** `head[Bool] nil [Bool]` well-typed, but stuck!!

How to handle this: (1) split type List into **emptyList** and **nonemptyList**

(2) raise an **EXCEPTION**

most languages do (2).

**Exceptions** are straightforward to evaluate/type. Read/enjoy Chapter 14!!  
+ do the exercises

## 5. Normalization

$t$  is normalizable  $\Leftrightarrow$   $t$  has normal form  $(\exists t' : t \rightarrow^* t' \not\rightarrow)$

**Recall:** the (pure) lambda calculus is Turing complete!

In the (pure) **simply typed lambda calculus** every well-typed term  
is normalizable!!

---

Define: (1)  $R_A(t) \Leftrightarrow t$  normalizable

(2)  $R_{T_1 \rightarrow T_2}(t) \Leftrightarrow t$  normalizable and  $\forall s: R_{T_1}(s) \Rightarrow R_{T_2}(t s)$

easy **Lemma:** If  $t : T$  and  $t \rightarrow t'$  then  $R_T(t) \Leftrightarrow R_T(t')$

Proof.  $t$  is normaliz.  $\Leftrightarrow t'$  is normaliz. (because  $\rightarrow$  is deterministic!)

Hence, if  $T=A$  then we are done!

$T=T_1 \rightarrow T_2$ :  $\forall s: R_{T_1}(s) \Rightarrow R_{T_2}(t s) \Leftrightarrow \forall s: R_{T_1}(s) \Rightarrow R_{T_2}(t' s)$

induction (on  $T!$ ) + because  $t s \rightarrow t' s$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

(1)  $t = x_i, T = T_i$   
then  $[x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t = v_i$ , and  $R_T(v_i)$  because it is a value.

(2)  $t = \lambda x:S_1. s_2, T = S_1 \rightarrow S_2$ , and  $x_1:T_1, \dots, x_n:T_n, x:S_1 \vdash s_2 : S_2$   
 $\Rightarrow [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t$  is a value! (by INV.L.)

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

(1)  $t = x_i, T = T_i$   
then  $[x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t = v_i$ , and  $R_T(v_i)$  because it is value.

(2)  $t = \lambda x:S_1. s_2, T = S_1 \rightarrow S_2$ , and  $x_1:T_1, \dots, x_n:T_n, x:S_1 \vdash s_2 : S_2$   
 $\Rightarrow [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t$  is a value! (by INV.L.)

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$   
 $\Downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

$\downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

$\downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

$\uparrow^+$

$(\lambda x:S_1. [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] s_2) s$



## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

$\downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

$\uparrow^+$

by easy **Lemma**,  $R_{S_2}((\lambda x:S_1. [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] s_2) s)$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

---

Proof. by induction on the derivation  $\vdash$

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

$\downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

$\uparrow^+$

by easy **Lemma**,  $R_{S_2}(\lambda x:S_1. [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] s_2) s$

$= R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] \underbrace{(\lambda x:S_1. s_2)}_t) s$

## 5. Normalization

Lemma:  $x_1:T_1, \dots, x_n:T_n \vdash t : T$  and  
 $v_1:T_1, \dots, v_n:T_n$  closed values, then  $R_T([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$

Proof. by induction on the derivation  $\vdash$

to show:  $s:S_1$  and  $R_{S_1}(s)$  implies  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t) s$

$\downarrow$   
 $\Rightarrow s \rightarrow^* v$  for some closed value  $v$

By induction,  $R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] [x \rightarrow v] s_2)$

$\uparrow^+$

by easy **Lemma**,  $R_{S_2}((\lambda x:S_1. [x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] s_2) s)$

$$= R_{S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] \underbrace{(\lambda x:S_1. s_2)}_t) s \quad \text{(by definition of R)}$$

$$R_{S_1 \rightarrow S_2}([x_1 \rightarrow v_1] \dots [x_n \rightarrow v_n] t)$$