

```
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. $\frac{\Gamma \vdash t_1 : \text{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 : t_2 : T_2}$ $\Rightarrow \text{ similar to let / application of an abstraction} \Rightarrow \text{ is there a lambda term with same typing??}$ $\frac{t_1 \Rightarrow t_1'}{t_1 : t_2 \Rightarrow t_1' : t_2} \qquad \text{unit} : t_2 \Rightarrow t_2$

```
1. Derived Forms

Example Sequencing.

\frac{\Gamma \vdash t_1 : \mathsf{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 ; t_2 : T_2}

\Rightarrow \text{ similar to } 1 \text{ et } / \text{ application of an abstraction}

\Rightarrow \text{ is there a lambda term with same typing??}

YES! Define t_1 ; t_2 := (\lambda x : \mathsf{Unit}. t_2) \ t_1 \qquad x \not\in \mathsf{FV}(t_2), \text{ fresh!}

\frac{\mathsf{F} \vdash (\lambda x : \mathsf{Unit}. t_2) \ t_1 : \mathsf{T}_2}{\mathsf{T}_1 ; t_2 \Rightarrow \mathsf{t}_1' ; t_2} \quad \text{unit} \ t; t_2 \Rightarrow t_2
```

```
1. Derived Forms

Example Sequencing.

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

\rightarrow \text{ similar to I et } / \text{ application of an abstraction}

\Rightarrow \text{ is there a lambda term with same typing??}

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

\frac{\Gamma \vdash (\lambda x : \text{Unit. } t_2) : \text{Unit}}{\Gamma \vdash (\lambda x : \text{Unit. } t_2) \ t_1 : T_2}

\frac{t_1 \Rightarrow t_1'}{t_1 : t_2 \Rightarrow t_1' : t_2} \quad \text{unit} : t_2 \Rightarrow t_2
```

```
1. Derived Forms

Example Sequencing.

\frac{\Gamma \vdash t_1 : \text{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 : t_2 : T_2}

\Rightarrow \text{ similar to let } / \text{ application of an abstraction}

\Rightarrow \text{ is there a lambda term with same typing??}

YES! Define t_1 : t_2 := (\lambda x : \text{Unit. } t_2) \ t_1 \qquad x \not\in \text{FV}(t_2), \text{ fresh!}

\frac{\Gamma, x : \text{Unit} \vdash t_2 : T_2}{\Gamma \vdash (\lambda x : \text{Unit. } t_2) : \text{Unit. } T_2} \qquad \Gamma \vdash t_1 : \text{Unit}}{\Gamma \vdash (\lambda x : \text{Unit. } t_2) : t_1 : T_2}

\frac{t_1 \Rightarrow t_1'}{t_1 : t_2 \Rightarrow t_1' : t_2} \qquad \text{unit}  t ; t_2 \Rightarrow t_2
```

```
1. Derived Forms

Example Sequencing.

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

\rightarrow \text{ similar to } I \text{ et } / \text{ application of an abstraction}

\Rightarrow \text{ is there a lambda term with same typing??}

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

\frac{\Gamma \vdash t_2 : T_2}{\Gamma, x : \text{Unit } \vdash t_2 : T_2} \times \text{FV}(t_2)

\frac{\Gamma \vdash (\lambda x : \text{Unit. } t_2) : \text{Unit.} }{\Gamma \vdash (\lambda x : \text{Unit. } t_2) : \text{Unit.} } \xrightarrow{\Gamma \vdash t_1 : \text{Unit.}}

\frac{t_1 \Rightarrow t_1'}{t_1 ; t_2 \Rightarrow t_1' ; t_2} \quad \text{unit } ; t_2 \Rightarrow t_2
```

```
1. Derived Forms

Example Sequencing.

\frac{\Gamma \vdash t_1 : \mathsf{Unit}}{\Gamma \vdash t_1 ; t_2 : T_2}

\Rightarrow \text{ similar to } 1 \text{ et } / \text{ application of an abstraction}

\Rightarrow \text{ is there a lambda term with same typing??}

YES! Define t_1 : t_2 : t_2 : t_1 : t_2 : t_3 : t_4 : t_4 : t_4 : t_4 : t_4 : t_5 : t_4 : t_5 : t_4 : t_5 : t_6 : t_7 : t_8 : t_8 : t_7 : t_8 : t_8
```

```
1. Derived Forms

Example Sequencing.

Questions:

1. Can you prove that ; is a derived form (= A. and B.)

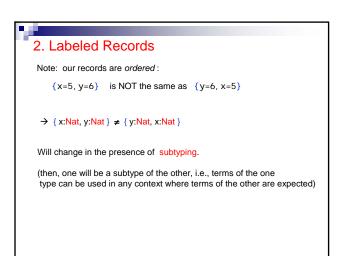
2. Is let a derived form?

\frac{t_1 \to t_1'}{t_1; t_2 \to t_1'; t_2} \quad \text{unit}; t_2 \to t_2 \\ \text{not needed anymore!}

A. t \to_{\text{ext}} t' \text{ iff } e(t) \to_{\text{int}} e(t') \quad e: \text{ ext} \to \text{int}

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

2. Labeled Records $\{x=5\} \text{ record of type } \{x:\text{Nat}\}$ $\{\text{partno=}5524, \text{ avai I abl e=true}\}$ $\text{ record of type } \{\text{partno:}\text{Nat, available:}\text{Bool}\}$ $\text{selection:} \quad \{x=5, y=6\} \cdot y \ \Rightarrow \ 6$ typing $\frac{\Gamma \vdash t_1: T_1, \ \dots, \ \Gamma \vdash t_n: T_n}{\Gamma \vdash \{I_1 = t_1, \dots, I_n = t_n\} \ : \ \{I_1: T_1, \dots, I_n: T_n\}}$ $\frac{\Gamma \vdash t_1: \{I_1: T_1, \dots, I_n: T_n\}}{\Gamma \vdash t_1: I_j: T_j}$



```
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 = \( \lambda n: \text{Nat.} < none=unit > \text{ as OptNat} \)

→ 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 buisiness day:

nextBui si nessDay = \( \lambda \text{: Weekday.} \) case w of
<monday=x> \( \rightarrow \text{: tuesday=uni t>} \) as Weekday
<tuesday=x> \( \rightarrow \text{: wednesday=uni t>} \) as Weekday

...
<fri day=x> \( \rightarrow \text{: monday=uni t>} \) 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

dollars2euros = λd: Float. timesfloat d 0.8145
euros2dollars = λd: Float. timesfloat d 1.2277
euros2dollars(dollars2euros 39.50) → 39.4984

But, dollars2euros(dollars2euros 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 = \( \lambda \text{d} \): DollarAmount.

\( \text{case d of } \) <dollars=x> \( \rightarrow \)
<euros=timesfloat x 0.8145> as EuroAmount
Type of dollar2euros: DollarAmount \( \rightarrow \) EuroAmount
```

```
4. Lists

List is a new type constructor (similar to →)

For a type T,
    List T describes finite-length lists whose elements are from T.

New syntactic forms:

ons[T]

isni![S](in![T]) → true
cons[T]

cons[T]

isni![S](cons[T]

isni![S](cons[T
```

```
4. Lists

typing:

F ⊢ ni I [T₁]: Li st T₁

F ⊢ t₁: T₁ F ⊢ t₂: Li st T₁

F ⊢ cons[T₁] t₁ t₂: Li st T₁

F ⊢ t₁: Li st T₁

F ⊢ i sni I [T₁] t₁: Bool

F ⊢ head[T₁] t₁: T₁

F ⊢ tai I [T₁] t₁: Li st T₁

→ 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 (∃t': t → t' )

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 \to 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→t' then R_T(t) \Leftrightarrow R_T(t')

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

Hence, if T=A then we are done!

T=T_1 \to 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 → t' s
```

5. Normalization Lemma: $x_1:T_1,...,x_n:T_n \vdash t:T$ and $v_1:T_1,...,v_n:T_n$ closed values, then $R_T([x_1 \rightarrow v_1]...[x_n \rightarrow v_n] t)$ Proof. by induction on the derivation \vdash (1) $t = x_i, T = T_i$ then $[x_1 \rightarrow v_1]..[x_n \rightarrow v_n] t = v_i$, 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 $x_1:T_1,...,x_n:T_n,x:S_1 \vdash S_2:S_2$ $\Rightarrow [x_1 \rightarrow v_1]...[x_n \rightarrow v_n] t$ is a value! (by INV.L.) to show: $s:S_1$ and $R_{S_1}(s)$ implies $R_{S_2}(([x_1 \rightarrow v_1]...[x_n \rightarrow v_n] t) s)$

```
5. Normalization

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

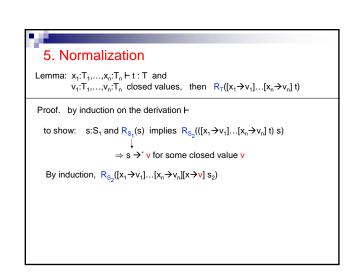
Proof. by induction on the derivation \vdash

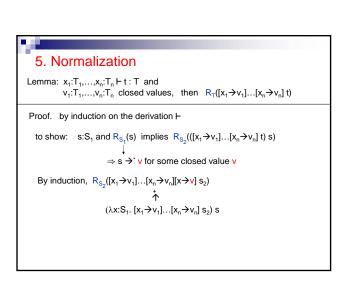
(1) t = x_i, T = T_i then [x_1 \rightarrow v_1]..[x_n \rightarrow v_n] t = v_i, and R_T(v_i) because it is value.

(2) t = \lambda x:S_1...S_2, T = S_1 \rightarrow S_2, and x_1:T_1,...,x_n:T_n,x:S_1 \vdash S_2:S_2 \Rightarrow [x_1 \rightarrow v_1]...[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]...[x_n \rightarrow v_n] t) s) \Rightarrow s \rightarrow v for some closed value v

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





```
5. Normalization

Lemma: x_1:T_1,...,x_n:T_n \vdash t: T and v_1:T_1,...,v_n:T_n closed values, then R_T([x_1 \rightarrow v_1]...[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]...[x_n \rightarrow v_n] t) s) \Rightarrow s \rightarrow v for some closed value v

By induction, R_{S_2}([x_1 \rightarrow v_1]...[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]...[x_n \rightarrow v_n] s_2) s
```